CAS automata elméleti felhasználásának didaktikai kérdései

doktori (PhD) értekezés

Dr. Maróti György

Debreceni Egyetem
Debrecen, 2004

Nyilatkozatok

Ezen értekezést a Debreceni Egyetem TTK **Matemaika és számítástudomány** Doktori Iskola **Matematika didaktika** programja keretében készítettem a Debreceni Egyetem TTK doktori (PhD) fokozatának elnyerése céljából.

Debrecen, 2004 május 2..

Dr. Maróti György
jelölt

Tanúsítom, hogy Dr. Maróti György doktorjelölt 2002-2004 között a fent megnevezett Doktori Iskola **Matematika didaktika** programjának keretében irányításommal végezte munkáját. Az értekezésben foglalt eredményekhez a jelölt önálló alkotó tevékenységével meghatározóan hozzájárult. Az értekezés elfogadását javasolom.

Debrecen, 2004 május 2

Dr. Csákány Béla
egyetemi tanár, témavezető

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani mindazoknak, akik a dolgozat elkészítésében szakmai vagy erkölcsi támogatást adtak.

Mindenek előtt volt professzoromnak és témavezetőmnek Dr. Csakány Béla egyetemi tanárnak tartozom köszönettel, aki pedagógiai munkásságával, egyéniségének, előadásainak sokszínűségével egy életre meghatározta matematikához való viszonyomat, és ezzel alapvetően befolyásolta életpályám alakulását. Erőt és önbizalmat adott azzal, hogy mentoromként vállalta dolgozatom témavezetését.

A dolgozat matematikai tartalma nagyrészt Dr. Imreh Balázs egyetemi docens munkássága és eredményei hatására alakult ki. Ő hívta fel a figyelmemet az irányítható automatákkal kapcsolatot vizsgálatokra és az ottani eredmények implementálhatóságának lehetőségeire.

Dr. Dömösi Pál egyetemi tanár régóta baráti figyelemmel kíséri botladozásaimat az automata elmélet és a komputer algebra határterületén. Szerteágazó és széleskörű ismereteinek megosztásával, bölcs tanácsaival nem kevéssel járult hozzá céljaim eléréséhez.

Végül köszönettel tartozom a Debreceni Egyetem Matematika és számítástudomány doktori iskolájának, aki a keretet biztosította erőfeszítéseimhez. Dr. Lajkó Károly támogató segítsége és jóindulatú tanácsai jelentették a biztonságos fogódzót az adminisztratív és szervezési feladatok útvesztőiben való eligazodáshoz.

Tartalom

1 Bevezetés

Az első komputer algebrai rendszerek (Computer Algebra System) az 1970-es években jelentek meg kvantummechanikai és csoportelméleti kutatásokhoz kapcsolódó számítások számítógépes támogatására. Az első sikerek hamar bővítettek a palettát. Új speciális célú CAS-okat fejlesztettek a többek között az általános relativitás, az algebrai geometria, a differenciálegyenletek és a nagy energiájú fizikai rendszerek vizsgálatára ([13]).

Ám hamarosan felmerült az általánosítás igénye és létrejöttek az általános célú komputer algebrai rendszerek, köztük a kanadai Waterloo egyetemen kifejlesztett Maple rendszer, mely nyitott architektúrájának köszönhetően napjaink egyik vezető CAS-ává fejlődött. A Maple architektúra jellemző vonása, hogy eljárásai a C nyelvű gyökér (Kernel) kivételével a Maple saját nyelvén készültek, és nagy részük a felhasználók által fejlesztett csomagokban (package) található. A rendszerhez eleddig fejlesztett csomagok száma meghaladja a kilencvenet ([16], [25]).

Ezek számát kívánta növelni az automata elméleti modellezést támogató `aut` csomag bevonva ezzel az automata elméletet a Maple-lel kezelhető területek körébe. Mint minden Maple csomag, az `aut` is adatstruktúrák és azok létrehozását, valamint a velük való különböző manipulációkat támogató eljárások összetartozó együttese. Az `aut` számos új adattípust vezet be, többek között a `automaton` típust véges nemdeterminisztikus automatákra, a `deterministic` és `complete` típust determinisztikus illetve teljes automatákra, a `regular` típust reguláris kifejezésekre. A csomag mintegy 35 eljárása gyakorlatilag lefedi a klasszikus automata elmélet eredményeit a jól ismert automata konstrukcióktól, az automaták minimalizációján és a reguláris kifejezések kezelésén keresztül a Kleene tételig, az automaták analíziséig és szintéziséig.

A jelen dolgozat célja megtalálni a választ két alapvető kérdésre.

1. *Használható-e, és ha igen, akkor milyen módon, az* `aut` *csomag automata elméleti kutatásokban?*
2. *Használható-e az aut csomag az automata elmélet oktatásának segédeszközeként, és ha igen, melyek azok a didaktikai módszerek, amelyek mentén a haladva a csomag megkönnyítheti az automata elmélet fogalmainak, tételeinek tanítását és segítséget nyújthat az automata elméleti problémamegoldás elsajátításában?*

A kijelölt céloknak megfelelően a dolgozat hármas felépítést mutat. A 2. fejezetben bemutatjuk az `aut` csomag használatát, miközben bevezetjük a szükséges automata elméleti fogalmakat ([8], [10], [12] és [17]) és a dolgozatban használatos jelöléseket. Kitérünk a különböző fogalmak Maple-beli

implementációjára, ismertetjük a legfontosabb automata konstrukciókat ([17]), mint például a részhalmaz konstrukció, a szorzat konstrukció stb., majd megvizsgáljuk az aut csomag konstrukciós motorjának, a Compose eljárásnak az algoritmusát és elvégezzük az algoritmus komplexitás vizsgálatát ([28]).

A 3. fejezet az aut csomag kutatási alkalmazásait tárgyalja. Elsőként egy általános tételt bizonyítunk **automata konstrukciók kompozícióira**. Vizsgálataink fókuszában azonban az automaták irányíthatóságának eldöntésére **Imreh és Steinby által kidolgozott algoritmus** ([4], [5], [14]) implementálása és vizsgálata áll. Megmutatjuk, hogy egy speciális adatszerkezet, az automata irányító táblájának felhasználásával az algoritmus nem csak az irányíthatóság eldöntésére képes, hanem alkalmassá tehető irányító szó meghatározására, sőt kommutatív automaták esetén minimális hosszúságú irányító szó előállítására is.

Az irányító tábla ismeretében lehetőség nyílik egy az irányítható automatákhoz társított automata bevezetésére, melyet μ**-automatának** nevezünk. Megmutatjuk, hogy a μ-automata irányíthatósága az automata irányíthatóságának szükséges és elégséges feltétele. A μ-automaták több bemenő jellel ám lényegesen rövidebb irányító szavakkal rendelkeznek. Nyitott kérdés, hogy a μ-automaták vizsgálata milyen szerepet játszhat az automaták irányíthatóságának vizsgálatában.

A dolgozat negyedik fejezete tárgyalja az aut csomag oktatásban való felhasználásának didaktikai kérdéseit. Részletesen elemezzük és Maple-lel illusztráljuk a reprezentációk oktatásban való használatára vonatkozó NCTM (National Council of Teachers of Mathematics) 2000-ben közzétett ajánlásában ([18], [19], [26]) megfogalmazott négyes szabályt (Rule of Four). Rámutatunk a különböző **reprezentációk** többszörösségére, a reprezentációk átfogalmazhatóságának és átjárhatóságának fontosságára, a realisztikus reprezentáció kognitív hatékonyságot befolyásoló szerepére. Végül a javaslatot teszünk egy a négyes szabálynál általánosabb didaktikai elv megfogalmazására, melyet a **többszörös reprezentáció elvének** nevezünk ([24]).

A dolgozat utolsó részében az itt javasolt didaktikai elvek konkrét alkalmazásaként kidolgozunk két Maple munkalapot, egyet a automata elméleti **fogalmak tanítására** ([21])és egy másikat az automataelméleti **probléma-megoldás tanítására** ([23]). A már jól ismert didaktika elvek ([2]), mit például az aktivitás elve, irányított felfedezéses tanulás elve stb., értelemszerű alkalmazása mellett javaslatot teszünk, és egyben példát is adunk a Maple munkalapok **didaktikai struktúrájának** felépítésére.

2 Az aut csomag bemutatása

A több éves fejlesztési folyamat eredményeként létrehozott aut automata elméleti csomag mindösszesen **35 eljárásból** áll, ami **2500 feletti forrássorban** ölt testet.

18 eljárás segíti az automaták létrehozását, a velük való manipulációt, valamint a különböző reprezentációkban megjelenő vizualizációt. A reguláris kifejezések kezelését 7 eljárás, míg az automata konstrukciókat további 8 eljárás támogatja.

Ez a fejezet rövid bevezetés az aut csomag használatába. A csomag egyes eljárásainak használata mellett itt vezetjük be azokat az alapfogalmakat és jelöléseket is, amelyeket a dolgozat későbbi fejezeteiben használni fogunk. Nem törekedhetünk minden eljárás átfogó bemutatására, mint ahogy a terjedelmi korlátok miatt nem törekedhetünk a bemutatandó eljárások összes opciójának ismertetésére sem. Amit bemutatunk az nem más, mint a csomag eljárásainak tipikus használata.

2.1 Fogalmak

Munkánkat az automata elméleti alapfogalmak definiálásával és azok Maple-beli implementációjának bemutatásával kezdjük.

Automaták

2.1.1 Definíció

Véges **automata** alatt egy $\Xi = [S, X, \delta, Q, F]$ rendezett ötöst értünk, ahol

1. S véges nem üres halmaz, az **automata állapotainak halmaza**.
2. X véges nem üres halmaz, melyet ábécé-nek nevezünk. Az X elemei az automata **bemenő jelei**.
3. $\delta : S \times X \rightarrow 2^S$ függvény, az automata **átmenet függvénye**.
4. Q állapotok (üres, vagy nem üres) halmaza az automata **kezdőállapotainak halmaza.**
5. F állapotok (üres, vagy nem üres) halmaza, melyet az automata **végállapot halmazának nevezünk**.

A Maple kezeli a halmaz adatstruktúrát így az állapothalmaz és a bemenő jelek ábrázolása nem okoz gondot.

```
> S:={a,b,c};
  X:={x,y};
```

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

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

Az átmenetfüggvény ábrázolására a Maple table adatstruktúráját használjuk.

```
➢ delta[a,x]:={a,b};
  delta[a,y]:={b};
  delta[b,e]:={a};
  delta[b,x]:={c};
```

$$\delta_{a,x} := \{a, b\}$$

$$\delta_{a,y} := \{b\}$$

$$\delta_{b,e} := \{a\}$$

$$\delta_{b,x} := \{c\}$$

Végül a az automata Maple implementációja egy öt elemű lista a fogalom matematikai definíciójának megfelelően. A második utasítás outputja azt mutatja, hogy a Maple felismeri Ξ -t, mint **automaton** típusú objektumot.

```
> Xi:=[S,X,delta,{a},{c}];
```

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

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

true

Automata fogalmunk nemdeterminisztikus**,** így helyénvaló a következő két fogalom bevezetése.

2.1.2 Definíció

A $\Xi = [S, X, \delta, Q, F]$ automatát **determinisztikusnak** nevezzük, ha $|Q| \le 1$, és minden $s \in S$ állapotra és $x \in X$ bemenő jelre $|\delta(s, x)| \le 1$ teljesül.

2.1.3 Definíció

A $\Xi = [S, X, \delta, Q, F]$ automatát **teljesnek** nevezzük, ha $1 \le |Q|$, és minden $s \in S$ állapotra és $x \in X$ bemenő jelre $1 \le |\delta(s, x)|$ teljesül.

Az **aut** csomagban mindkét, most bevezetett fogalom implementálva van.

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

false

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

false

Hogyan hozunk létre automatákat?

Az automaták létrehozása a definíción alapján nehézkes és felesleges lépéseket tartalmaz. Vegyük észre, hogy nincs szükség az állapothalmaz és a bemenő jelek halmazának megadására akkor, ha az átmenet függvényt leíró táblában megadjuk az összes releváns átmenetet. Ezt illusztrálja a **genaut** eljárás, melynek

paramétere egy olyan lista, ami leírja például azt, hogy a létrehozandó automatának legyen egy *a* állapota, melyből az `1` bemenő jel hatására a *b* állapotba kerül. Amit a genaut elvégez, az nem más, mint összegyűjti mindazon állapotokat és a bemenő jeleket, amelyek feltűnnek aktuális paraméterében, továbbá javaslatot tesz a kezdő és végállapotok halmazára. A genaut minden olyan állapotot, melynek neve *a*-val kezdődik kezdő állapotnak, az *f*-vel kezdődő nevűeket pedig végállapotnak tekint.

Az alábbi parancs egy Ξ automaton típusú objektumot hoz létre. A genaut outputja az automata minden komponensét megmutatja, kivéve az átmenet függvényt. Ezért hasznos a printaut eljárás, melynek segítségével képernyőre írhatjuk az automata **átmenet tábláját**. Ennek első oszlopában az állapotok, első sorában pedig a bemenő jelek láthatók. Az *s* állapot sorának és *x* bemenő jel oszlopának kereszteződésében pedig az átmenet függvény értéke, $\delta(s, x)$ található.

```
> Xi:=genaut([a,1,b,b,{0,1},b,b,0,c,c,0,f]);
```

$$\Xi := [\{f, a, b, c\}, \{0, 1\}, \delta, \{a\}, \{f\}]$$

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

true

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

STA \ INP	0	1
a	-	{*b*}
b	{*b*, *c*}	{*b*}
c	{*f*}	-
f	-	-

Set of initial states: , {*a*}

Set of final states: , {*f*}

Hogy írjuk le az automata működését?

A nemdeterminisztikus automata fogalom következménye, hogy az átmenet eredményeként létrejövő új állapot nincs egyértelműen meghatározva, ezért célszerűbb aktuális állapot helyett a **lehetséges állapotok aktuális halmazáról** (aktuális LAH) beszélni. Az első LAH maga a kezdőállapot-halmaz (=*Q*). Itt olvassa az automata az első bemenő jelet, melynek hatására létrejövő LAH nem más, mint $\delta(Q, x)$. Ezután következik a második bemenő olvasása és a következő LAH meghatározása, és így tovább. A számítás eredménye az input szó teljes feldolgozása után keletkező LAH. Ennek a működésnek a matematikai

kezelését teszi lehetővé az automata általánosított átmenetfüggvénye.

2.1.4 Definíció

A $\Delta : 2^S \times X^* \to 2^S$ **általánosított átmenetfüggvényt** a következő rekurzív definícióval adjuk meg. Az állapothalmaz bármely *P* részhalmazára és bármely *w* bemenő szóra valamint *x* bemenő jelre

1. $\Delta(P, \varepsilon) = P$
2. $\Delta(P, x)$ egyenlő a $\delta(s, \xi)$ uniójával, ahol $s \in P$.
3. $\Delta(P, wx) = \delta(\Delta(P, w), x)$

Az általánosított átmenet függvény Maple-beli implementációját a Delta eljárás valósítja meg. Nézzünk néhány példát a Delta működésére! A Delta első paramétere maga az automata, második paramétere egy állapot, vagy állapotok egy halmaza, harmadik paramétere pedig egy bemenő jel, vagy egy bemenő szó. A Delta outputja a keletkező LAH .

```
> Delta(Xi,b,`0`);
```

$$\{b, c\}$$

```
> Delta(Xi,a,`1001`);
```

$$\{b\}$$

Ha tudni akarjuk a működés különböző lépéseiben keletkező részeredményeket, akkor használnunk kell a Delta opciót. A tr2 opció hatására Delta megmutatja az input szó feldolgozása során keletkező LAH-okat, sőt minden egyes lépésben megmutatja a következő LAH kiszámításainak közbülső lépéseit is.

```
> Delta(Xi,a,`1001`,tr2, show);
-Delta begins with {a}
 .a,`1`->{b}
-Delta({a},`1`) = {b}
 .b,`0`->{b, c}
-Delta({b},`0`) = {b, c}
 .b,`0`->{b, c}
 .c,`0`->{f}
-Delta({b, c},`0`) = {f, b, c}
 .b,`1`->{b}
-Delta({f, b, c},`1`) = {b}
```

$$\Delta(\{a\}, 1001) = \{b\}$$

```
Result of Delta:
```

$$\{b\}$$

Egy másik szemléletes, bár ritkábban használt interpretációja az automaták működésének a **lehetséges futások halmaza** (LFH). Az ebben az interpretációban

az automata minden egyes lépésben véletlenszerű választásokat hajt végre. Működését rögtön választással, kezdi, véletlenszerűen kiválaszt ugyanis egy kezdőállapotot (mondjuk *a*-t) a kezdőállapotok halmazából. Ezután olvassa az első bemenő jelet (mondjuk *x*-et) és a következő állapotot a $\delta(a,x)$ halmazból választja ki. Így egy *k* hosszóságú *w* bemenő szó hatására állapotok egy *k+1* elemű listáját kapjuk, melyet ***w* szóhoz tartozó futásnak** nevezünk. Ha összegyűjtjük a *w* bemenő szóhoz az összes lehetséges futást, akkor megkapjuk a *w* szóhoz tartozó LFH-t.

A Delta a runs opcióval állítja elő a harmadik paramétereként megadott bemenő szóhoz tartozó LFH-t.

```
> Delta(Xi,{a},`10100`,runs);
```

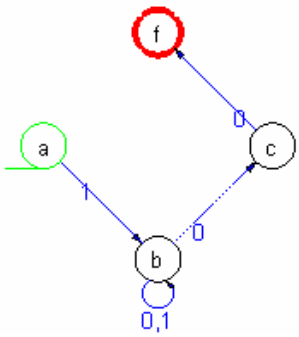
$$\{[a,b,b,b,b,b],[a,b,b,b,b,c],[a,b,b,b,c,f]\}$$

```
> Delta(Xi,{a},`01`,runs);
```

$$\{\ \}$$

Végül említést kell tenni az automaták legszemléletesebb reprezentációjáról, a átmenet gráfról. Az átmenet gráf szögpontjai az automata állapotaival, irányított élei pedig az automata bemenő jeleivel címkézettek. A $b \in \delta(a,x)$ reláció az átmenet gráfon úgy jelenik meg, hogy az *a* csúcsból *x*-szel címkézett él halad a gráf *b* szögpontjába. Az automata átmenet gráfját a plotaut eljárással jeleníthetjük meg.

```
> plotaut(Xi);
```



Szavak és nyelvek felismerése

Az automaták szavak és nyelvek felismerésére használatos eszközök.

2.1.5 Definíció

A véges nemüres X halmazt **ábécének** nevezzük Az X elemeit **jeleknek**, míg az X elemeiből álló véges sorozatokat **X feletti szavaknak** hívjuk. Azt a szót, ami egyetlen jelet sem tartalmaz **üres szónak** nevezzük és ε -nal jelöljük. Az X ábécé

feletti összes szó halmazát X*-gal jelöljük. Végül X* részhalmazait **X feletti nyelveknek** hívjuk.

2.1.6 Definíció

Azt mondjuk, hogy a $\Xi = [\, S\,,\, X\,,\, \delta\,,\, Q\,,\, F\,]$ automata **felismeri** (vagy **elfogadja**) a $w \in X^*$ bemenő szót, ha a $\Delta(\Xi, Q, w) \cap F$ halmaz nem üres. A Ξ automata által felismert szavak halmazát $L(\Xi)$-vel jelöljük, és azt mondjuk. hogy $L(\Xi)$ a Ξ automata által **felismert nyelv.**

Az accept eljárás egy logikai függvény, mely akkor és csak akkor ad igaz értéket, ha az első paramétereként megadott automata felismeri a második paramétereként megadott bemenő szót. A tr1 opció használata az eddigiekhez hasonlóan a számítás részleteit mutatja meg.

```
> accept(Xi,`100100`);
```

$$true$$

```
> accept(Xi,`100100`,tr1,show);
-Delta begins with {a}
-Delta({a},`1`) = {b}
-Delta({b},`0`) = {b, c}
-Delta({b, c},`0`) = {f, b, c}
-Delta({f, b, c},`1`) = {b}
-Delta({b},`0`) = {b, c}
-Delta({b, c},`0`) = {f, b, c}
```

$$\Delta(\{a\}, 100100) = \{f, b, c\}$$

```
Result of Delta:
```

$$\{f, b, c\}$$

$$set\ of\ final\ states:\ ,\ \{f\}$$

```
Result of accept:
```

$$true$$

Az accept eljárás annak eldöntésében ad segítséget, hogy egy adott szót az automata felismer-e vagy sem. Ám ha az automata által felismert nyelv valamely véges részhalmazára vagyunk kíváncsiak, akkor a langaut eljárást kell használnunk. Az alábbi utasításban található 0..6 tartomány a megjelenítendő szavak hosszát határozza meg, a growing és show opciók pedig megjelenítés módját szabályozzák.

```
> langaut(Xi,2..6,growing,show);
length 2:
```

$$\{\ \}$$

```
length 3:
```

$$\{100\}$$

length 4:

$$\{1000, 1100\}$$

length 5:

$$\{11100, 10000, 11000, 10100\}$$

length 6:

$$\{111000, 111100, 110000, 110100, 100100, 100000, 101000, 101100\}$$

Result of langaut:

$$\{111000, 11100, 111100, 110000, 110100, 100100, 100, 100000, 101000, 1000, 1100, 101100, 10000, 11000, 10100\}$$

Összefüggő automaták

Az automaták azon állapotai, amelyek nem érhetők el valamely kezdőállapotból, nem tudnak részt venni a szavak felismerésében, így tehát ebből a szempontból feleslegesek. Célszerű tehát a következő fogalmak bevezetése.

2.1.7 Definíció

Azt mondjuk, hogy a Ξ automata b állapota az a kezdőállapotból **elérhető**, ha létezik p bemenő szó, hogy $b \in \Delta(\Xi, a, p)$. A Ξ automatát **iniciálisan összefüggőnek** nevezzük, ha minden állapota elérhető valamelyik kezdőállapotból.

Könnyen megmutatható, hogy a kezdőállapotokból elérhető állapotok halmaza zárt az átmenetekre nézve.

2.1.8 Definíció

Tekintsük a $\Xi = [A, X, \delta, Q, F]$ automatát. Azt $\Phi = [B, X, \delta, Q, F \cap B]$ automatát, amelyre

1. B a Ξ kezdőállapotaiból elérhető állapotok halmaza,
2. Minden $b \in B$ -re és $x \in X$ -re $\delta(\Phi, b, x) = \delta(\Xi, b, x)$.

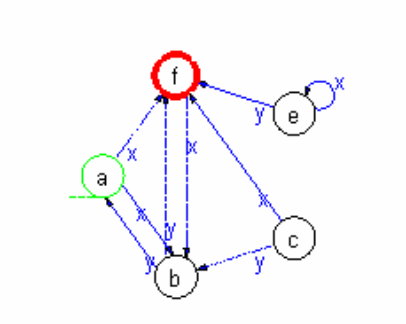
A Ξ automata **kezdőállapot halmaza által generált részautomatájának** nevezzük.

Azt `aut` csomag a kezdőállapot halmaz által generált részautomata előállítására a `ConIco` (Construct Initially Connected subautomaton) eljárást kínálja. Azt illusztráció kedvéért hozzunk létre a Ξ automatát, majd rajzoljuk fel a Ξ, és a ConIco(Xi) automaták átmenet gráfjait.

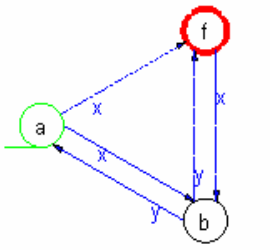
➢ `Xi:=genaut([a,x,{b,f}, b,y,a,c,x,f,c,y,b,f,x,b,b,y,f, e,x,e,e,y,f]);`

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

```
> plotaut(Xi);
```




```
> plotaut(ConIco(Xi));
```



Könnyen látható, hogy a kezdőállapot halmaz által generált részautomata iniciálisan összefüggő.

2.2 Automata konstrukciók

Automata elméleti konstrukció alatt olyan eljárást, algoritmust értünk, ami egy vagy több automatából kiindulva újabb automatát hoz létre. Amikor például a felismerhető nyelvek zártságát bizonyítjuk a reguláris műveletekre, akkor nem mást teszünk, minthogy az egyes nyelveket felismerő automatákból konstruálunk olyan automatákat, amelyek rendre a nyelvek unióját, szorzatát illetve iteráltját ismerik fel.

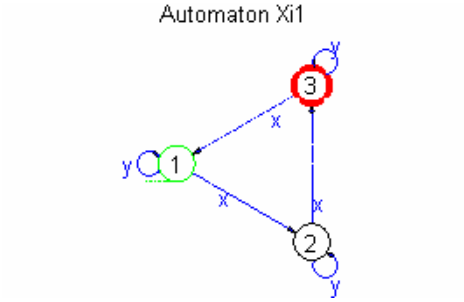
Az automata konstrukciók jól kidolgozott eljárásokat adnak a programozó kezébe. Ami izgalmassá teszi a kérdést, az nem más, mint az a felismerés, hogy ezek az automata konstrukciók egy közös módszerben gyökereznek. Ezt valósítja meg a 2.3 fejezetben ismertetendő `Compose` eljárás, melyről kiderül, hogy megfelelően paraméterezett hívásaival felépíthető a legtöbb automata elméleti konstrukció.

Ahhoz, hogy a bemutatásra kerülő automata konstrukciókat szemléltetni tudjuk hozzunk létre két automatát és a szemléletesség kedvéért rajzoljuk fel mindkét automata átmenet gráfját.

```
> Xi1:=renaut(genaut([[2,3,1],[1,2,3]]),inp=x);
```

$$\Xi 1 := [\{1, 2, 3\}, \{x, y\}, \delta, \{1\}, \{3\}]$$

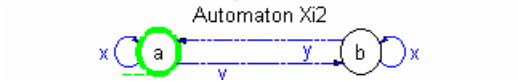
```
> plotaut(Xi1,"Automaton Xi1");
```




```
> Xi2:=genaut([a,y,b,b,y,a,a,x,a,b,x,b],a,a);
```

$$\Xi 2 := [\{a, b\}, \{x, y\}, \delta, \{a\}, \{a\}]$$

```
> plotaut(Xi2,"Automaton Xi2");
```



Nem nehéz annak ellenőrzése, hogy a $\Xi 1$ automata akkor és csak akkor ismeri fel a *w* bemenő szót, ha *w*-ben az *x* jelek száma kongruens 2 modulo 3. Az $\Xi 2$ pedig azokat és csak azokat a szavakat ismeri fel, amelyekben páros számú *y* szerepel. Illusztrációként adjuk ki a következő két utasítást.

```
> accept(Xi1,xyxxyyyxyx);
```

$$true$$

```
> langaut(Xi2,0..4,gro,sho);
length 0:
```

$$\{\varepsilon\}$$

```
length 1:
```

$$\{x\}$$

```
length 2:
```

$$\{xx, yy\}$$

```
length 3:
```

$$\{yyx, xxx, xyy, yxy\}$$

```
length 4:
```

$$\{xxxx, xxyy, xyxy, xyyx, yxxy, yxyx, yyxx, yyyy\}$$

```
Result of langaut:
```

$$\{\varepsilon, x, yyx, xxxx, xx, xxx, xxyy, yy, xyxy, xyyx, xyy, yxy, yxxy, yxyx, yyxx, yyyy\}$$

Az automata konstrukciók végrehajtásához az `aut` csomag `Construct` eljárása adja az interfészt. A `Construct` első paramétere a konstrukció neve (pl. product, set, union, iteration, stb.), míg a második paraméter azoknak az automatáknak a

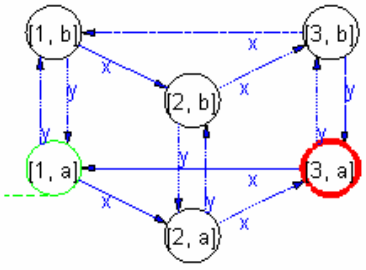
listája, amelyekre az első paraméterként meghatározott konstrukció elvégzendő.

Például, ha a **product** konstrukciót nevezzük meg első paraméterként, másodikként pedig a $\Xi 1$ és $\Xi 2$ automatákat, akkor a **Construct** eljárás olyan automatát épít, amelynek állapotai a $\Xi 1$ és $\Xi 2$ automaták állapotaiból képzett rendezett párok, a kezdő és végállapotok pedig rendre a $\Xi 1$ és $\Xi 2$ kezdő ill. végállapotaiból képzett rendezett párok halmaza. A konstruált automata szinkronizálja $\Xi 1$ és $\Xi 2$ működését oly módon, hogy a rendezett párok első komponensein a $\Xi 1$, második komponensein padig a $\Xi 2$ átmenetfüggvénye szerint működik. Megmutatható, hogy az így létrejövő automata az L($\Xi 1$) és az L($\Xi 2$) nyelvek metszetét ismeri fel.

```
> Phi1:=Construct(product,[Xi1,Xi2]);
```

$$\Phi 1 := [\{[1, a], [2, a], [3, a], [1, b], [2, b], [3, b]\}, \{x, y\}, \delta, \{[1, a]\}, \{[3, a]\}]$$

```
> plotaut(Phi1,[2,a]=[0,-.5],[3,a]=[1,0],[1,b]=[-
1,1],[2,b]=[0,0.5],[3,b]=[1,1]);
```



Ez az átmenet gráf szépen mutatja a szorzat (product) konstrukcióval létrehozott automata belső struktúráját. Figyeljük meg továbbá, hogy a **plotout** eljárás megengedi, hogy meghatározzuk a felrajzolandó automata állapotainak koordinátáit. Például a [2,a]=[0,-.5] opció azt írja elő, hogy a [2,a] állapotnak megfelelő kör (szögpont) középpontja a [0,-.5] koordinátájú pontra essen.

A felismerhető nyelvek szorzatát felismerő automata konstrukciója a **Construct** eljárás **concatenate** opciójával valósítható meg. Ezzel az opcióval a **Construct** olyan automatát hoz létre, melynek állapotai a kiindulási automaták állapothalmazai uniójának részhalmazai. Az eljárás feltételezi, hogy a konstrukcióban résztvevő automaták állapothalmazai diszjunktak. Ha mégsem így lenne, akkor megfelelő opció beállításával a **Construct** átnevezi az állapotokat.

A keletkező automata állapotai tehát halmazok, melyekben mindkét automatából szerepelhetnek állapotok. Az átmenetfüggvény úgy határozza meg az új állapotot, hogy az első automata állapotaira annak átmenetfüggvényét a második automata állapotaira pedig a második automata átmenetfüggvényét alkalmazva állítja össze

az új halmazt. Például a $\{2, a\}$ állapotra alkalmazva az x bemenő jelet a $\{3, a\}$ halmaz keletkezik, mert $\delta(\Xi 1, 2, x) = 3$ és $\delta(\Xi 2, a, x) = a$. Ha azonban ez a lépés olyan halmazt eredményez, amelyben szerepel $\Xi 1$ valamely végállapota, akkor az átmenetfüggvény ezt a halmazt kiegészíti a Xi2 kezdőállapotával. Erre a $\delta(\Phi 2, \{2, b\}, x) = \{3, a, b\}$ átmenet szolgáltat példát. A konstruált automata végállapotai azok a halmazok, amelyekben szerepel a második automata valamelyik végállapota.

```
> Phi2:=Construct(con,[Xi1,Xi2]);
```

$$\Phi 2 := [\{\{2\}, \{1\}, \{2, b\}, \{2, a\}, \{2, a, b\}, \{3, a\}, \{3, a, b\}, \{1, b\}, \{1, a\}, \{1, a, b\}\}, \{x, y\}, \delta, \{\{1\}\}, \{\{2, a\}, \{2, a, b\}, \{3, a\}, \{3, a, b\}, \{1, a\}, \{1, a, b\}\}]$$

```
> printaut(Phi2);
```

STA \ INP	x	y
$\{1\}$	$\{2\}$	$\{1\}$
$\{2\}$	$\{3, a\}$	$\{2\}$
$\{1, a\}$	$\{2, a\}$	$\{1, b\}$
$\{1, b\}$	$\{2, b\}$	$\{1, a\}$
$\{3, a\}$	$\{1, a\}$	$\{3, a, b\}$
$\{2, a\}$	$\{3, a\}$	$\{2, b\}$
$\{2, b\}$	$\{3, a, b\}$	$\{2, a\}$
$\{1, a, b\}$	$\{2, a, b\}$	$\{1, a, b\}$
$\{3, a, b\}$	$\{1, a, b\}$	$\{3, a, b\}$
$\{2, a, b\}$	$\{3, a, b\}$	$\{2, a, b\}$

Initial state:, $\{1\}$

Set of final states: ,

$\{\{2, a\}, \{2, a, b\}, \{3, a\}, \{3, a, b\}, \{1, a\}, \{1, a, b\}\}$

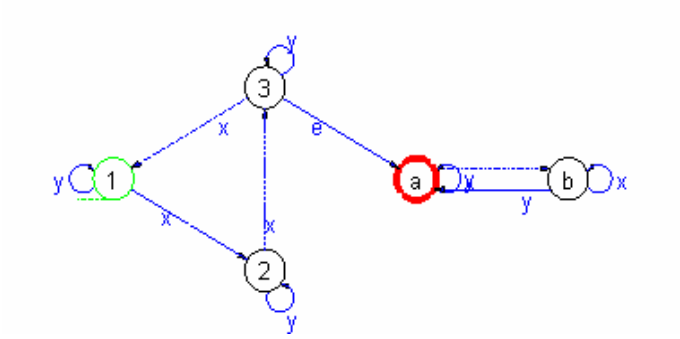
Figyeljük meg, hogy a **Construct** utasítás első paraméterében elegendő a konstrukció nevének első három karakterét megadni.

Az **aut** csomag kezeli az e-átmenetes automatákat. Ezek segítségével a felismerhető nyelvek unióját, szorzatát és iteráltját felismerő automata egyszerű és szemléletes módon megkonstruálható.

```
> Phi3:=Construct(e_c,[Xi1,Xi2]);
```

$$\Phi 3 := [\{1, 2, 3, a, b\}, \{x, y\}, \delta, \{1\}, \{a\}]$$

```
> plotaut(Phi3,2=[0.5,-
0.87],3=[0.5,0.87],a=[2,0],b=[3.5,0]);
```



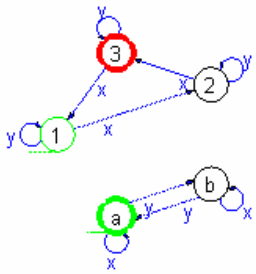
A $\Phi 3$ automatát az **e_concatenate** opcióval hoztuk létre. A keletkező automata átmenet gráfja szépen mutatja konstrukció lényegét, ami a két automata "sorba kötését" valósítja meg. A $\Phi 3$ úgy kezdi működését, mint a $\Xi 1$. Ám amikor a $\Xi 1$ végállapotába ér, akkor e-átmenettel, vagyis olvasás nélkül átmegy a $\Xi 2$ kezdőállapotába, ahol működését a $\Xi 2$ átmenetfüggvénye szerint folytatja. $\Phi 3$ végállapot halmaza megegyezik a második automata végállapot halmazával.

A következő parancs egy újabb példát ad automata konstrukcióra, nevezetesen az unió konstrukciót (**union**) mutatja be. Az unió konstrukció feltételezi, hogy a kiindulási automaták állapothalmazai diszjunktak, és képezi ezen két állapothalmaz unióját. Az kezdő és végállapotok halmaza ugyancsak az egyes automaták kezdő és végállapot halmazainak uniójaként áll elő. Az új automata működése úgy történik, hogy ha az automata $\Xi 1$ -beli állapotban van, akkor a $\Xi 1$ átmenetfüggvénye szerint, ha pedig $\Xi 2$ -beli állapotban van, akkor a $\Xi 2$ átmenetfüggvénye szerint vált állapotot. Nevével összhangban az unió konstrukcióval létrehozott automata a $\Xi 1$ és a $\Xi 2$ által felismert nyelvek unióját ismeri fel.

```
> Phi4:=Construct(uni,[Xi1,Xi2]);
```

$\Phi 4 := [\{1, 2, 3, a, b\}, \{x, y\}, \delta, \{1, a\}, \{3, a\}]$

```
> plotaut(Phi4,2=[.8, .58],b=[.8, -.58]);
```




```
> type(Phi4,deterministic);
```

false

Vegyük észre, hogy az unió konstrukció nemdeterminisztikus automatát hoz létre, még akkor is ha determinisztikus automatákra alkalmazzuk. Valóban, a keletkező automatának legalább két kezdőállapota van, ami önmagában elegendő a nemdeterminisztikussághoz. Konstruáljuk meg ezek után a ConDet eljrásással a $\Phi 4$ automatához a vele ekvivalens determinisztikus automatát.

```
> Phi5:=ConDet(Phi4);
```

$$\Phi 5 := [\{\{2,b\},\{2,a\},\{3,a\},\{1,b\},\{1,a\},\{3,b\}\},\{x,y\},\delta,$$
$$\{\{1,a\}\},\{\{2,a\},\{3,a\},\{1,a\},\{3,b\}\}]$$

```
> printaut(Phi5);
```

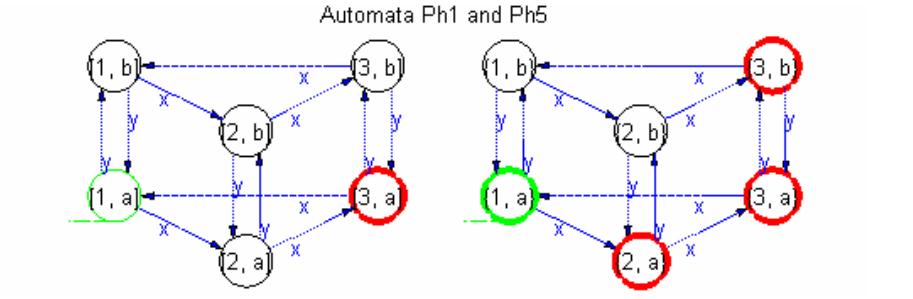
$$\begin{bmatrix} STA \setminus INP & x & y \\ \{3,a\} & \{1,a\} & \{3,b\} \\ \{1,a\} & \{2,a\} & \{1,b\} \\ \{2,a\} & \{3,a\} & \{2,b\} \\ \{2,b\} & \{3,b\} & \{2,a\} \\ \{1,b\} & \{2,b\} & \{1,a\} \\ \{3,b\} & \{1,b\} & \{3,a\} \end{bmatrix}$$

Initial state:, $\{1,a\}$

Set of final states:, $\{\{3,b\},\{2,a\},\{1,a\},\{3,a\}\}$

A átmenet tábla alapján szembeötlő, hogy minden állapot kételemű halmaz, mégpedig úgy, hogy az egyik elem $\Xi 1$ -ből, a másik pedig $\Xi 2$ -ből való. A következő utasítás a $\Phi 1$ és a $\Phi 5$ automaták átment gráfjait rajzolja fel, a gráfok szögpontjainak alkalmas elrendezésével. Az eredmény eléggé meglepő.

```
> p1:=plotaut(Phi5,{2,a}=[0,-.5],{3,a}=[1,0],{1,b}=[-
1,1],{2,b}=[0,0.5],{3,b}=[1,1]):
p2:=plotaut(Phi1,[1,a]=[-4,0],[2,a]=[-3,-.5],[3,a]=[-
2,0],[1,b]=[-4,1],[2,b]=[-3,0.5],[3,b]=[-2,1]):
plots[display]({p1,p2},scaling=unconstrained,title="Automata
Ph1 and Ph5");
```



Az ábrára pillantva azonnal eldönthető, hogy a két automata állapotaik elnevezésétől eltekintve megegyezik, más szóval mint algebrai struktúrák

izomorfak. Később látni fogjuk ez nem véletlen, ugyanis a 3.2 fejezetben bizonyítjuk, hogy determinisztikus automatákra az unió konstrukció és a részhalmaz konstrukció kompozíciójával létrehozott automata izomorf a szorzat konstrukció által létrehozott automatával.

Ezt a fejezetet implementált automata konstrukcióinak áttekintésével zárjuk. Az alábbiakban táblázatos formában összefoglaljuk az `aut` csomag mindazon eszközeit, amelyek automata konstrukciók elvégzését támogatják.

Az `aut` csomag konstrukciós eszközei

Nyelvi konstrukciók

`Construct(set,[Xi])`	Ekvivalens determinisztikus automata létrehozása (ld. ConDet)
`Construct(product,[Xi1,Xi2])`	Az $L(\Xi 1) \cap L(\Xi 2)$ nyelvet felismerő automata létrehozása
`Construct(union,[Xi1,Xi2])`	Az $L(\Xi 1) \cup L(\Xi 2)$ nyelvet felismerő automata létrehozása
`Construct(e_union,[Xi1,Xi2])`	Az $L(\Xi 1) \cup L(\Xi 2)$ nyelvet felismerő automata létrehozása e-átmenetes konstrukcióval
`Construct(concatenate,[Xi1,Xi2])`	Az $(L(\Xi 1)).(L(\Xi 2))$ nyelvet felismerő automata létrehozása
`Construct(e_c,[Xi1,Xi2])`	Az $(L(\Xi 1)).(L(\Xi 2))$ nyelvet felismerő automata létrehozása e-átmenetes konstrukcióval
`Construct(iterate,[Xi])`	Az $L(\Xi 1)*$ nyelvet felismerő automata létrehozása
`Construct(e_iterate,[Xi1,Xi2])`	Az $L(\Xi 1)*$ nyelvet felismerő automata létrehozása e-átmenetes konstrukcióval
`Construct(substitute,[Xi1,T])`	Nyelvek szubsztitúcióját felismerő automata létrehozása

Ekvivelencia konstrukciók

`ConDet(Xi)`	(CONstruct DETerministic automaton) ekvivalens determinsztikus automata létrehozása
`ConCom(Xi)`	(CONstruct COMplete automaton) ekvivalens teljes automata létrehozása

ConNet(Xi)	(CONstruct Non E-Transition) ekvivalens e-átmenet mentes automata létrehozása
ConMin(Xi	(CONstruct MINinmal automaton) ekvivalens minimális automata létrehozása
ConIco(Xi)	(CONstruct Initially COnnected automaton) iniciálisan összefüggő automata létrehozása

2.3 Egy heurisztikus algoritmus: a Compose eljárás

Az **aut** csomag nagyban támaszkodik az automata elmélet eredményeire. Ebben az értelemben nem tesz mást, minthogy implementálja az automata elméletben kidolgozott eljárásokat. Néhány esetben azonban szükség van olyan eljárás megírására, amelyhez az elmélet egyáltalán nem, vagy csak kevés kapaszkodót ad. Ilyenkor nincs más út, mint a heurisztika. Ilyen heurisztikus algoritmust valósít meg a **Compose** eljárás az **aut** csomag általános konstrukciós motorja.

Ebben a fejezetben is a 2.2 fejezetben létrehozott $\Xi 1$ és $\Xi 2$ közös kimenő jelekkel rendelkező automatákat használjuk mondanivalónk illusztrálására. Az **aut** csomagban megvalósított implementációs módszer, vagyis a **Compose** eljárás hívásának használatát erre a két automatára alkalmazott "módosított" szorzat konstrukción mutatjuk be. A módosítás abban áll, hogy amíg a szorzat konstrukciót a felismerhető nyelvek metszetének felismerésére használtuk, addig a módosított konstrukció a két nyelv unióját ismeri fel, miközben csak a szorzat konstrukcióval készült automata végállapot halmazát változtatja meg.

Az implementáció első lépése az új automata bemenő jeleinek specifikálása. Ez a halmaz megegyezik a $\Xi 1$ automata bemenő jeleinek halmazával.

```
> newinp:=Xi1[2];
```

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

Ezután meghatározzuk a konstruálandó automata kezdő állapotait oly módon, hogy vesszük a $\Xi 1$ és a $\Xi 2$ automaták kezdőállapot halmazainak Descartes szorzatát, és a keletkező halmazt a **newini** változóban tároljuk.

```
> newini:=Cartesian(Xi1[4],Xi2[4]);
```

$$newini := \{[1, a]\}$$

Az implementáció legfontosabb lépése az új automata működésének meghatározása. A **newtra** függvény három formális paramétere automaták egy kételemű listája, egy állapot (ami szintén egy kételemű lista), és egy bemenő jel. Figyeljük meg, hogy **newtra** azt írja elő, hogy az $a = [a_1, a_2]$ állapotból az új

automata az x bemenő jel hatására a [$\Delta(l_1, a_1, x)$, $\Delta(l_2, a_2, x)$] állapotba megy át. Vagyis a konstruálandó automata állapotainak első komponensén a $\Xi 1$, második komponensén pedig a $\Xi 2$ automata átmenetfüggvényének megfelelően működik.

```
> newtra:=(l,a,x)-
>{[Delta(l[1],a[1],x),Delta(l[2],a[2],x)]};
```

$$newtra := (l, a, x) \rightarrow \{[\Delta(l_1, a_1, x), \Delta(l_2, a_2, x)]\}$$

Az új automata végállapotait úgy kapjuk, a **newfin** változóba elhelyezzük mindazon rendezett párokat, amelyek legalább egyik komponense végállapot a neki megfelelő automatában.

```
> newfin:=Cartesian(Xi1[1],Xi2[5]) union
Cartesian(Xi1[5],Xi2[1]);
```

$$newfin := \{[1, a], [2, a], [3, a], [3, b]\}$$

Végül az implementációt a **Compose** eljárás hívásával tesszük teljessé. Első paramétere az a metódus, ami szerint dolgozik. Ez **define** illetve **generate** lehet. Ebben a fejezetben csak a **generate** metódussal foglalkozunk. A **Compose** második paramétere azon automaták listája, amelyekre a konstrukciót el akarjuk végezni. Harmadik paramétere pedig egy négyelemű lista, ami a konstruálandó automata egyes komponenseit specifikálja, az előkészítéseknek megfelelően.

```
> Compose(generate,[Xi1,Xi2],[newinp,newtra,newini,newfin]);
```

$$[\{[1, a], [2, a], [3, a], [1, b], [2, b], [3, b]\}, \{x, y\}, \delta, \{[1, a]\}, \{[1, a], [2, a], [3, a], [3, b]\}]$$

```
> printaut(%);
```

$STA \setminus INP$	x	y
$[1, a]$	$[2, a]$	$[1, b]$
$[1, b]$	$[2, b]$	$[1, a]$
$[2, a]$	$[3, a]$	$[2, b]$
$[2, b]$	$[3, b]$	$[2, a]$
$[3, a]$	$[1, a]$	$[3, b]$
$[3, b]$	$[1, b]$	$[3, a]$

$$Initial\ state:, [1, a]$$

$$Set\ of\ final\ states:, \{[1, a], [2, a], [3, a], [3, b]\}$$

A **Compose** eljárással való automata konstrukció a következő elveken alapul. Elsőként elképzeljük az új automata összes lehetséges állapotát, megnevezzük az **univerzumot**. A mi esetünkben az univerzum a kiindulási automaták

állapothalmazainak Descartes szorzata. Ezután megnevezzünk az univerzum egy vagy több konkrét elemét, azaz az univerzum egy részhalmazát (**newini**), melyet az konstruálandó automata kezdőállapot halmazának tekintünk. Végül megadjuk azt az eljárást (**newtra**), ami specifikálja, hogy a meglévő állapotokból hogyan kell új állapotokat generálni. A **Compose** kiindul a kezdőállapotok halmazából, és sorra generálja az univerzum mindazon állapotait, amelyek úgy keletkeznek, hogy a már legenerált állapotokra alkalmazza a bemenő jeleket a **newtra** eljárásnak megfelelően, és az átmenetek eredményeivel bővíti a legenerált állapotok halmazát. A keletkezett bővebb halmazra a **Compose** megismétli az eljárást mindaddig, amíg már nem tud új állapotot generálni. Megjegyezzük, hogy ez mindig bekövetkezik, mert a **Compose** egy véges univerzum részhalmazainak növekvő sorozatát állítja elő. Az eljárás vége egyben azt is biztosítja, hogy az univerzum olyan részhalmazához jutunk, ami zárt az átmenetekre nézve.

Működése során a **Compose** nyilvántartja a generált állapotok halmazát (GAH). Kezdetben a GAH nem más, mind a kezdőállapotok halmaza. Az eljárás minden egyes lépésében a GAH bővül, mindaddig, amíg a bővítés lehetséges. Másrészt a **Compose** egy vermet (stack) használ arra, hogy tárolja a feldolgozás során még nem érintett állapotokat. A **Compose** működése közben az aktuális lépés mindig úgy indul, hogy GAH tartalmazza az eddig generált összes állapotot, míg a verem a GAH azon állapotait tartalmazza, amelyekre eleddig nem került sor a generálás során.

Mindez jól megfigyelhető a következő utasítás outputján. A **Compose** ugyanis beszédessé válik, ha használjuk a **tr1** opciót, melynek hatására megjeleníti az eljárás során keletkező részeredményeket, míg a **state** opció a az egyes lépésekben keletkező GAH megjelenítését írja elő.

```
>
Compose(generate,[Xi1,Xi2],[Xi1[2],newtra,newini,newfin],tra
ce1,state):
```

Set of states in step 0 =, $\{[1, a]\}$

```
[[1, a]]->pop([1, a])->[]
 .delta([1, a],x) = [2, a]
 .delta([1, a],y) = [1, b]
```

Set of states in step 1 =, $\{[1, a], [2, a], [1, b]\}$

```
[[2, a], [1, b]]->pop([1, b])->[[2, a]]
 .delta([1, b],x) = [2, b]
 .delta([1, b],y) = [1, a]
```

Set of states in step 2 =, $\{[2, b], [1, a], [2, a], [1, b]\}$

```
[[2, a], [2, b]]->pop([2, b])->[[2, a]]
 .delta([2, b],x) = [3, b]
 .delta([2, b],y) = [2, a]
```

Set of states in step 3 =, $\{[2,b],[1,a],[2,a],[1,b],[3,b]\}$

```
[[2, a], [3, b]]->pop([3, b])->[[2, a]]
 .delta([3, b],x) = [1, b]
 .delta([3, b],y) = [3, a]
```

Set of states in step 4 =, $\{[2,b],[1,a],[2,a],[3,a],[1,b],[3,b]\}$

```
[[2, a], [3, a]]->pop([3, a])->[[2, a]]
 .delta([3, a],x) = [1, a]
 .delta([3, a],y) = [3, b]
```

Set of states in step 5 =, $\{[2,b],[1,a],[2,a],[3,a],[1,b],[3,b]\}$

```
[[2, a]]->pop([2, a])->[]
 .delta([2, a],x) = [3, a]
 .delta([2, a],y) = [2, b]
```

Set of states in step 6 =, $\{[2,b],[1,a],[2,a],[3,a],[1,b],[3,b]\}$

Ezek után megadjuk a **Compose** algoritmusának formális leírását.

```
1. Inicializáció
   GAH:=newini
   push(newini)
2. Mindaddig, amig a verem nem üres 3. és 4. lépés
ismétlése
3. s:=pop()
4. Minden x bemenő jelre
   do
      r:=newtra([Xi1,Xi2],s,x)
      if not r nem eleme GAH then
         GAH:=GAH union {r}
         push(r)
      fi
   od:
```

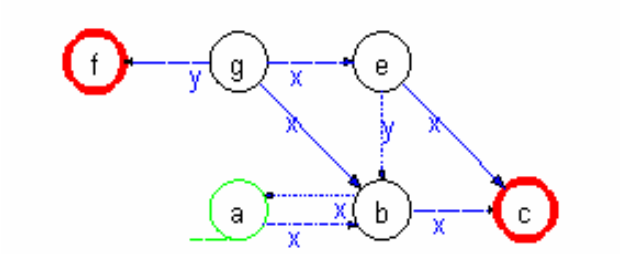
A **Compose** "brute force" heurisztikán alapuló algoritmusa biztosítja, hogy az univerzum elemei az eljárás során legfeljebb egyszer kerülnek megfogásra, ám akkor az algoritmus minden egyes bemenő jelre ellenőrizi a keletkező új állapotot. Eszerint, ha a $\Xi 1$ automata n_1 állapotú, a $\Xi 2$ automata pedig n_2 állapotú, akkor az algoritmus komplexitása a legrosszabb esetben O($n_1\, n_2\, m$), ahol m a közös ábécé elemeinek száma. A legrosszabb eset akkor következik be, ha a két automata direkt szorzata iniciálisan összefüggő.

A **Compose** rendkívül hasznosnak és nem utolsó sorban hatékonynak bizonyul. Ennek illusztrálására további két példát mutatunk felhasználására: az ekvivalens iniciálisan összefüggő automata, és ekvivalens teljes automata előállítását. Elsőként hozzuk létre a Ξ automatát és rajzoljuk fel az átmenet gráfját.
>

```
Xi:=genaut([a,x,b,b,x,{c,a},e,x,c,e,y,b,g,x,{e,b},g,y,f],a,{
f,c});
```

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

```
>  plotaut(Xi,a=[-1,0],b=[0,0],c=[1,0],g=[-1,1],e=[0,1],f=[-
2,1],plotpar=[scaling=unconstrained]);
```



Könnyű belátni, hogy az *e*, az *f* és a *g* állapotok feleslegesek, egyikük sem érhető el a kezdőállapotból. A **Compose** megfelelően paraméterezett hívásával elérhető, hogy megkapjuk a Ξ iniciálisan összefüggő részautomatáját, vagyis megszabadulhatunk az összes felesleges állapottól. A konstruálandó automatának az eredetivel azonos módon kell működnie, így annak kezdőállapota (newini), végállapota (newfin), bemenő jelei (newinp) és átmenetei (newtra) is megegyeznek az eredeti automata megfelelő komponenseivel.

```
> newinp:=Xi[2];
newini:=Xi[4];
newtra:=(l,a,x)->Delta(l[1],a,x);
newfin:=Xi[5];
```

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

$$newini := \{a\}$$

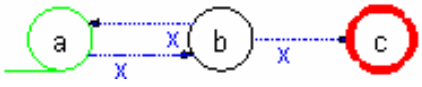
$$newtra := (l, a, x) \rightarrow \Delta(l_1, a, x)$$

$$newfin := \{f, c\}$$

```
> Phi:=Compose(generate,[Xi],[Xi[2],newtra,newini,newfin]);
```

$$\Phi := [\{a, b, c\}, \{x, y\}, \delta, \{a\}, \{c\}]$$

```
> plotaut(Phi,[1,0],line,plotpar=[scaling=unconstrained]);
```



Bár a Φ automatának két bemenő jele van (*x* és *y*), Φ mégsem tartalmaz *y*-átmenetet. Eszerint Φ nem teljes. Konstruáljuk hát meg a vele ekvivalens teljes Θ automatát egy új τ állapot hozzávételével. A Φ összes állapotát változatlanul hagyjuk, a nem definiált átmenetek, valamint a τ-ra vonatkozó átmenetek végeredményeként pedig magát a τ állapotat határozzuk meg. Azt

természetesen feltételezzük, hogy a kiindulási automata állapot halmazának τ nem eleme.

```
> newtra:=proc(l,a,x)
 if a=tau then {tau}
 elif Delta(l[1],a,x,set)={} then
 {tau}
 else
 Delta(l[1],a,x,set)
fi
end;
```

$newtra := \mathbf{proc}(l, a, x)$ **end proc**

if $a = \tau$ **then** $\{\tau\}$

elif $\Delta(l[1], a, x, set) = \{\ \}$ **then** $\{\tau\}$

else $\Delta(l[1], a, x, set)$

end if

A teljes automata kezdő és végállapota, valamint bemenő jeleinek halmaza az eredetivel megegyező.

```
> newinp:=Phi[2];
newini:=Phi[4];
newfin:=Phi[5];
```

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

$$newini := \{a\}$$

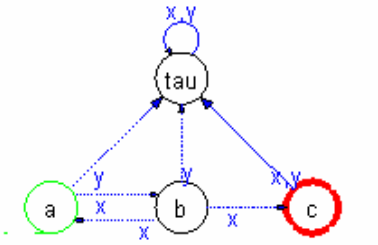
$$newfin := \{c\}$$

A következő két parancs magát a konstrukciót és az eredmény vizualizációját valósítja meg.

```
>
Theta:=Compose(generate,[Phi],[newinp,newtra,newini,newfin])
;
```

$$\Theta := [\{a, b, c\}, \{x, y\}, \delta, \{a\}, \{c\}]$$

```
> plotaut(Theta,line,c=[1,0],tau=[0,1]);
```



A két utolsó parancs outputja önmagáért beszél. A **Compose** helyesen végezte el a kijelölt feladatot.

3 Az aut csomag alkalmazásai

Ebben a fejezetben beszámolunk néhány az **aut** csomag használatához közvetlenül kötődő automata elméleti eredményéről. A CAS tudományos kutatásbeli szerepe napjainkra már nem szorul magyarázatra. Mind az általános célú, mind pedig célfeladatokra fejlesztett komputer algebrai rendszerek egyre szélesebb körben nyernek létjogosultságot a tudományos kutatás különböző területein. A jelen dolgozat egyik célja éppen az, hogy az automata elméleti kutatásokat is felsorakoztassa ezen területek sorába.

3.1 Automata konstrukciók kompozíciója

A dolgozat 2.2 fejezetében egy példa kapcsán azt találtuk, hogy determinisztikus automaták esetén az unió konstrukcióra alkalmazott részhalmaz konstrukció a szorzat konstrukcióval izomorf automatát eredményezett. Most ezt problémakört az általánosan esetre vizsgáljuk. Pontos definíciókat adunk az egyes automata konstrukciókra és egy általános összefüggést bizonyítunk azok kompozíciójára, melynek speciális eseteként megkapjuk a korábbi fejezetben felfedezett tulajdonságot.

3.1.1 Definíció

Tekintsük a $\Xi_1 = [A_1, X, \delta, Q_1, F_1]$ és a $\Xi_2 = [A_2, X, \delta, Q_2, F_2]$ automatákat. Azt mondjuk, hogy a $\Phi = [A, X, \delta, Q, F]$ automata a Ξ_1 és Ξ_2 automatákból **szorzat konstrukcióval** keletkezik, ha

1. $A = A_1 \times A_2$
2. $Q = Q_1 \times Q_2$
3. Bármely $a \in A$ állapotra és $x \in X$ bemenő jelre
   $\delta(\Phi, a, x) = \delta(\Xi_1, a_1, x) \times \delta(\Xi_2, a_2, x)$.

A Φ automata kezdőállapot halmaza által generált részautomatáját a Ξ_1 és Ξ_2 automaták **szorzat-automatájának** nevezzük és $\Xi_1 \times \Xi_2$ -vel jelöljük.

Megjegyezzük, hogy a fenti definíció a szorzat-automata végállapotára nem tesz semmilyen megszorítást, így különböző végállapot halmazok választásával más-más szorzat-automatához jutunk. Determinisztikus és teljes automatákra a szorzat konstrukció az automaták, mint algebrai struktúrák direkt szorzatát eredményezi.

3.1.2 Propozíció

Az $A_1 \times A_2$ minden $B_1 \times B_2$ részhalmazára, minden $x \in X$ bemenő jelre

és minden $w \in X^*$ bemenő szóra

1. $\Delta(\Xi_1 \times \Xi_2, B_1 \times B_2, x) = \Delta(\Xi_1, B_1, x) \times \Delta(\Xi_2, B_2, x)$, és
2. $\Delta(\Xi_1 \times \Xi_2, Q_1 \times Q_2, w) = \Delta(\Xi_1, Q_1, w) \times \Delta(\Xi_2, Q_2, w)$

Bizonyítás

Az első egyenlőség igazolásához legyen $[a_1, a_2]$ eleme a

$$\Delta(\Xi_1 \times \Xi_2, B_1 \times B_2, x)$$

Δ halmaznak. Ekkor létezik olyan $[b_1, b_2] \in B_1 \times B_2$, amelyre $[a_1, a_2]$ eleme $\delta(\Xi_1 \times \Xi_2, [b_1, b_2], x) = \delta(\Xi_1, b_1, x) \times \delta(\Xi_2, b_2, x)$ -nek, amiből $a_i \in \delta(\Xi_i, b_i, x)$ $(i = 1, 2)$ következik, ugyanakkor $\delta(\Xi_i, b_i, x) \subseteq \Delta(\Xi_i, B_i, x)$ $(i = 1, 2)$ és így

$$[a_1, a_2] \in \Delta(\Xi_1, B_1, x) \times \Delta(\Xi_2, B_2, x).$$

Fordítva, vegyük a $\Delta(\Xi_1, B_1, x) \times \Delta(\Xi_2, B_2, x)$ halmaz egy $[a_1, a_2]$ elemét. Ekkor alkalmas $b_i \in B_i$ -re $a_i \in \delta(\Xi_i, b_i, x)$ $(i = 1, 2)$. Ám ekkor $[b_1, b_2] \in B_1 \times B_2$ és $[a_1, a_2] \in \delta(\Xi_1, b_1, x) \times \delta(\Xi_2, b_2, x) = \delta(\Xi_1 \times \Xi_2, [b_1, b_2], x)$, ami viszont részhalmaza $\Delta(\Xi_1 \times \Xi_2, B_1 \times B_2, x)$-nek.

A 2. pontbeli egyenlőség bizonyítását a *w* hossza szerinti teljes indukcióval végezzük el. Az állítás az üres szóra triviális. Tegyük fel ez után, hogy az egyenlőség *w*-nél rövidebb szavakra teljesül, és legyen $w = px$. Ekkor

$$\begin{aligned}
&\Delta(\Xi_1 \times \Xi_2, Q_1 \times Q_2, w) = && \# \ w = px \\
&= \Delta(\Xi_1 \times \Xi_2, Q_1 \times Q_2, px) = && \# \ \Delta \text{ definíciója} \\
&= \Delta(\Xi_1 \times \Xi_2, \Delta(\Xi_1 \times \Xi_2, Q_1 \times Q_2, p), x) = && \# \text{ indukciós felt.} \\
&= \Delta(\Xi_1 \times \Xi_2, \Delta(\Xi_1, Q_1, p) \times \Delta(\Xi_2, Q_2, p), x) = && \# \text{ 1. egyenlőség} \\
&= \Delta(\Xi_1, \Delta(\Xi_1, Q_1, p), x) \times \Delta(\Xi_2, \Delta(\Xi_2, Q_2, p), x) && = \# \ \Delta \text{ definíciója} \\
&= \Delta(\Xi_1, Q_1, px) \times \Delta(\Xi_2, Q_2, px) && \# \ w = px \\
&= \Delta(\Xi_1, Q_1, w) \times \Delta(\Xi_2, Q_2, w)
\end{aligned}$$

Ezzel a második egyenlőséget is igazoltuk.

3.1.3 Definíció

Tegyük fel, hogy a $\Xi_1 = [A_1, X, \delta, Q_1, F_1]$ és a $\Xi_2 = [A_2, X, \delta, Q_2, F_2]$ automaták állapot halmazai diszjunktak. Azt mondjuk, hogy a $\Phi = [A, X, \delta, Q, F]$ automata a Ξ_1 és Ξ_2 automatákból **unió konstrukcióval** keletkezik, ha

1. $A = A_1 \cup A_2$
2. $Q = Q_1 \cup Q_2$
3. Bármely $a \in A$ állapotra és $x \in X$ bemenő jelre
   $\delta(\Phi, a, x) = \delta(\Xi_1, a, x)$ ha $a \in A_1$, és
   $\delta(\Phi, a, x) = \delta(\Xi_2, a, x)$ ha $a \in A_2$.

A Φ automata kezdőállapot halmaza által generált részautomatáját a Ξ_1 és Ξ_2 automaták **unió-automatájának** nevezzük és $\Xi_1 \cup \Xi_2$ -vel jelöljük.

Szemben a szorzat konstrukcióval, ami determinisztikus automatákra determinisztikus automatát eredményez, az unió konstrukció eredménye nemdeterminisztikus automata, ha a konstrukcióban résztvevő automaták kezdőállapot halmazai nem üresek.

3.1.4 Propozíció

Az A_i állapothalmaz minden B_i részhalmazára ($i = 1, 2$), minden $x \in X$ bemenő jelre és minden $w \in X^*$ bemenő szóra

1. $\Delta(\Xi_1 \cup \Xi_2, B_1 \cup B_2, x) = \Delta(\Xi_1, B_1, x) \cup \Delta(\Xi_2, B_2, x)$,
2. $\Delta(\Xi_1 \cup \Xi_2, Q_1 \cup Q_2, w) = \Delta(\Xi_1, Q_1, w) \cup \Delta(\Xi_2, Q_2, w)$

Bizonyítás

Az 1. egyenlőség az unió-automata átmenetfüggvénye definíciójának közvetlen következménye.
A második egyenlőség az üres szóra triviális. Ha pedig feltesszük, hogy a *p* szóra már teljesül, akkor

$\Delta(\Xi_1 \cup \Xi_1, Q_1 \cup Q_2, px)$ # Δ definíciója
$= \Delta(\Xi_1 \cup \Xi_1, \Delta(\Xi_1 \cup \Xi_1, Q_1 \cup Q_2, p), x)$ # indukciós feltevés
$= \Delta(\Xi_1 \cup \Xi_1, \Delta(\Xi_1, Q_1, p) \cup \Delta(\Xi_2, Q_2, p), x)$ # 1. egyenlőség
$= \Delta(\Xi_1, \Delta(\Xi_1, Q_1, p), x) \cup \Delta(\Xi_2, \Delta(\Xi_2, Q_2, p), x)$ # Δ definíciója

$= \Delta(\Xi_1, Q_1, px) \cup \Delta(\Xi_1, Q_1, px)$.

Ezzel az állítás bizonyítását befejeztük.

3.1.5 Definíció

Azt mondjuk, hogy a $\Phi = [B, X, \delta, Q, G]$ automata a $\Xi = [A, X, \delta, Q, F]$ automatából **részhalmaz konstrukcióval** keletkezik, ha

1. $B = 2^A$
2. Minden $b \in B$ állapotra és $x \in X$ bemenő jelre
$$\delta(\Phi, b, x) = \Delta(\Xi, b, x)$$.

A Φ automata kezdőállapot halmaza által generált részautomatáját a Ξ automata **részhalmaz-automatájának** nevezzük és 2^ Ξ - vel jelöljük.

3.1.6 Definíció

Azt mondjuk, hogy a $\Xi_1 = [A_1, X, \delta, Q_1, F_1]$ és a $\Xi_2 = [A_2, X, \delta, Q_2, F_2]$ automaták **izomorfak**, ha létezik olyan $\alpha : A_1 \to A_2$ bijekció, hogy minden $a \in A_1$ állapotra és $x \in X$ bemenő jelre $\delta(\Xi_1, a, x)\,\alpha = \delta(\Xi_2, a\,\alpha, x)$.

3.1.7 Tétel

Bármely Ξ_1 és Ξ_2 automatára 2^($\Xi_1 \times \Xi_2$) izomorf a 2^($\Xi_1 \cup \Xi_2$) automatával, vagyis a szorzat-automata részhalmaz-automatája izomorf az unió-automata részhalmaz-automatájával.

Bizonyítás

Legyen $\Xi_1 = [A_1, X, \delta, Q_1, F_1]$ és $\Xi_2 = [A_2, X, \delta, Q_2, F_2]$, és tegyük fel, hogy A_1 -nek és A_2 -nek nincs közös eleme. Az $A_1 \times A_2$ halmaz részhalmazaira bevezetjük a π_1 és π_2 projekciókat. Legyen $B \subseteq A_1 \times A_2$, ekkor

$B\,\pi_1 = \{a \mid a \in A_1$, létezik olyan $b \in A_2$, hogy $[a, b] \in B\}$

és

$B\,\pi_2 = \{b \mid b \in A_2$, létezik olyan $a \in A_a$, hogy $[a, b] \in B\}$.

Először azt mutatjuk meg, hogy a 2^($\Xi_1 \times \Xi_2$) automata minden *B* állapotára $B = B\,\pi_1 \times B\,\pi_2$.

Mivel 2^($\Xi_1 \times \Xi_2$) iniciálisan összefüggő, ezért alkalmas *w* bemenő szóra *B* = Δ ($\Xi_1 \times \Xi_2$, $Q_1 \times Q_2$, *w*), ami az 3.1.2 Propozíció 2. pontbeli egyenlősége szerint egyenlő $\Delta(\Xi_1, Q_1, w) \times \Delta(\Xi_2, Q_2, w)$ -val. Ugyanakkor $B = \Delta(\Xi_1, Q_1, w) \times \Delta(\Xi_2, Q_2, w)$ maga után vonja, hogy $B\,\pi_i = \Delta(\Xi_i, Q_i, w)$ ($i = 1, 2$), amiből pedig már következik, hogy $B = B\,\pi_1 \times B\,\pi_2$.

Vegyük ezután azt az α leképezést, ami a 2^($\Xi_1 \times \Xi_2$) automata minden B állapotához a 2^($\Xi_1 \cup \Xi_2$) automata $B\,\pi_1 \cup B\,\pi_2$ állapotát rendeli, tehát amelyre

$$B\,\alpha = B\,\pi_1 \cup B\,\pi_2 \ .$$

Megmutatjuk, hogy α bijekciót létesít a szorzat-automata részhalmaz-automatájának állapothalmaza és az unió-automata részhalmaz-automatájának állapothalmaza között.

Az α kölcsönösen egyértelműségének igazolásához tegyük fel, hogy a 2^($\Xi_1 \times \Xi_2$) automata *B* és *C* állapotaira $B\,\alpha = C\,\alpha$. Ebből

$$B\,\pi_1 \cup B\,\pi_2 = C\,\pi_1 \cup C\,\pi_2$$

következik, ám $B\,\pi_1$ és $C\,\pi_1$ az A_1 részhalmazai, míg $B\,\pi_2$ és $C\,\pi_2$ a tőle diszjunkt A_2 halmaz részhalmazai. Így szükségképpen $B\,\pi_i = C\,\pi_i$ ($i = 1, 2$), amiből kapjuk, hogy $B = C$.
Tekintsük most a 2^($\Xi_1 \cup \Xi_2$) automata egy *C* állapotát. Ekkor alkalmas *w* bemenő szóra $C = \Delta(\Xi_1 \cup \Xi_2, Q_1 \cup Q_2, w)$. Azt mutatjuk meg, hogy a

2^($\Xi_1 \times \Xi_2$) automata B = Δ ($\Xi_1 \times \Xi_2$, $Q_1 \times Q_2$, *w*) állapota a *C* állapot α melletti őse. Valóban

$$\begin{aligned}
&B\,\alpha && \#\ B \text{ választása}\\
&= \Delta\,(\,\Xi_1 \times \Xi_2\,,\ Q_1 \times Q_2\,,\ w)\ \alpha = && \#\ 3.1.2 \text{ Propozíció}\\
&= (\,\Delta(\Xi_1, Q_1, w) \times \Delta(\Xi_2, Q_2, w)\,)\ \alpha = && \#\ \alpha \text{ definíciója}\\
&= \Delta(\Xi_1, Q_1, w) \cup \Delta(\Xi_2, Q_2, w) = && \#\ 3.1.4 \text{ Propozíció}\\
&= \Delta(\Xi_1 \cup \Xi_2, Q_1 \cup Q_2, w) = && \#\ C \text{ választása}\\
&= C.
\end{aligned}$$

Ezzel megmutattuk, hogy α ráképezés.

Végül megmutatjuk, hogy α felcserélhető az átmenetekkel. Legyen a B a 2^($\Xi_1 \times \Xi_2$) egy állapota. Mint korábban már beláttuk $B = B\pi_1 \times B\pi_2$. Így tetszőleges $x \in X$ bemenő jelre

$\Delta_{(\Xi_1 \times \Xi_2, B, x)} =$ # $B = B\pi_1 \times B\pi_2$

$= \Delta_{(\Xi_1 \times \Xi_2, B\pi_1 \times B\pi_2, x)}\alpha =$ # 3.1.2 Propozíció

$= (\Delta(\Xi_1, B\pi_1, x) \times \Delta(\Xi_2, B\pi_2, x))\alpha =$ # α definíciója

$= \Delta(\Xi_1, B\pi_1, x) \cup \Delta(\Xi_2, B\pi_2, x) =$ # 3.1.4 Propozíció

$= \Delta(\Xi_1 \cup \Xi_2, B\pi_1 \cup B\pi_2, x)$ # $(B\pi_1 \times B\pi_2)\alpha = B\pi_1 \cup B\pi_2$

$= \Delta_{(\Xi_1 \cup \Xi_2, (B\pi_1 \times B\pi_2)\alpha, x)} =$ # $B = B\pi_1 \times B\pi_2$

$= \Delta_{(\Xi_1 \cup \Xi_2, B\alpha, x)}$.

A bizonyítás kész.

Mivel determinisztikus automatákra a szorzat-automata maga is determinisztikus, valamint determinisztikus automaták részhalmaz-automatája izomorf magával a kiindulási automatával, ezért a 3.1.7 tétel következményeként kapjuk, az alábbi állítást.

3.1.8 Következmény

Determinisztikus Ξ_1 és Ξ_2 automatára a szorzat-automata izomorf az unió-automata részhalmaz-automatájával.

3.2 Az Imreh-Steinby algoritmus

Előzmények

3.2.1 Definíció

Azt mondjuk, hogy a *w* bemenő szó **összefésüli** a Ξ determinisztikus automata *a* és *b* állapotait, ha $\Delta(\Xi,a,w) = \Delta(\Xi,b,w)$. Ha az *a* és *b* állapotokra létezik őket összefésülő szó, akkor *a*-t és *b*-t **összefésülhetőnek** nevezzük. Végül, ha a *w* bemenő szó az összes állapotpárt összefésüli, akkor *w*-t **irányító szónak** nevezzük. Magát a Ξ automatát pedig **irányíthatónak** mondjuk, ha létezik irányító szava.

Tekintsük a Ξ determinisztikus automatát és tegyük fel, hogy *w* irányító szó. Ekkor bármely *a* és *b* állapotra $\Delta(\Xi,a,w) = \Delta(\Xi,b,w)$, amiből az következik, hogy a $\Delta(\Xi, \Xi_1, w) = \{\Delta(\Xi,a,w) \mid a \in \Xi_1\}$ halmaz egyelemű. Fordítva,

ha $\Delta(\Xi, \Xi_1, w)$ egyetlen elemből áll, akkor a $\Delta(\Xi, a, w) = \Delta(\Xi, b, w)$ egyenlőségnek minden *a* és *b* állapotra teljesülnie kell. Megállapíthatjuk tehát, hogy a Ξ automata akkor és csak akkor irányítható, ha $\Delta(\Xi, \Xi_1, w)$ egyelmű halmaz.

Eszerint ha Φ a Ξ automata olyan részhalmaz automatája, amelyre

$\Phi_4 = \Xi_1$, vagyis a Φ kezdőállapota megegyezik Ξ állapothalmazával és

$\Phi_5 = \{\{a\} | a \in \Xi_1\}$, vagyis Φ végállapotai a Ξ állapothalmazának egyelemű részhalmazai,

akkor Φ pontosan a Ξ irányító szavait ismeri fel.

A Φ automata előállításához a **Compose** eljárást használjuk a 2.3 fejezetekben ismertetett módon.

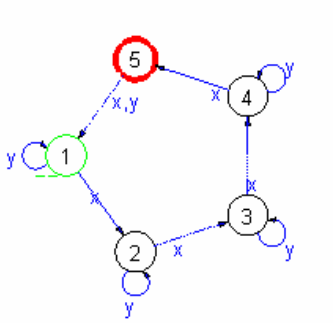
```
> with(aut):
> ConDir:=proc(A::automaton)
      local newtra,newfin, newini, newinp:
      newinp:=A[2]:
      newini:={A[1]}:
      newtra:=proc(L,a,x) {Delta(L[1],a,x,set)} end;
      newfin:=proc(l,x)::Boolean: evalb(nops(x)=1) end;
      Compose(generate,[A],[newinp,newtra,newini,newfin],\
      args[2..-1])
  end:
```

A **ConDir** eljárás működését az *n* állapotú Cerny típusú automatával illusztráljuk. Ennek az automatának *n* állapota van: *1,2,...,n*, valamint két bemenő jele: *x* és *y*. Az *i* állapotra és az *x* bemenő jelre $\delta(i, x) = (i \bmod n) + 1$, míg az *y* bemenő jel az *n* kivételével mindegyik állapotot fixen hagyja és $\delta(n, y) = 1$. A következő utasítás egy Maple függvényobjektumot hoz lérte, ami az *n* paraméter függvényében előállítja az *n* állapotú Cerny típusú automatát.

```
➢ Cerny:=n->renaut(genaut([[seq(i,i=2..n),1],\
  [seq(i,i=1..n-1),1]]),inp=x);
```

$$Cerny := n \rightarrow \text{renaut}(\text{genaut}([[\text{seq}(i, i = 2 .. n), 1], [\text{seq}(i, i = 1 .. n - 1), 1]]), inp = x)$$

```
> plotaut(Cerny(5));
```



Az n állapotú Cerny típusú automata irányítható. Valóban, irányító szava például az $y(x^{(n-1)}y)^{(n-2)}$ bemenő szó. Legyen ugyanis *a* tetszőleges állapot. Ekkor *i* = $\delta(a,y) \in \{1 .. n-1\}$.

Ebből azt kapjuk, hogy $\Delta(i, x^{(n-1)}y) = i-1$ ha i>1, különben pedig 1. Eszerint i>1 esetén $\Delta(i, (x^{(n-1)}y)^{(i-1)}) = 1$, továbbá minden *m* természetes számra $\Delta(1, (x^{(n-1)}y)^{m}) = 1$, amiből állításunk már adódik.

Az eddig elmondottak alapján a ...

```
> p:=`y(xxxxy)(xxxxy)(xxxxy)`;
```

$$p := y(xxxxy)(xxxxy)(xxxxy)$$

...bemenő szó irányító szava a Cerny(5) automatának. Mint az a következő utasítás outputján jól látható, a *p* szó az *n* állapotú Cerny típusú automata minden állapotát valóban az 1 állapotba viszi.

```
> for state to 5 do
 print('Delta'(' Xi',state,p)=Delta(Cerny(5),state,p))
od:
```

$$\Delta(\Xi, 1, y(xxxxy)(xxxxy)(xxxxy)) = 1$$
$$\Delta(\Xi, 2, y(xxxxy)(xxxxy)(xxxxy)) = 1$$
$$\Delta(\Xi, 3, y(xxxxy)(xxxxy)(xxxxy)) = 1$$
$$\Delta(\Xi, 4, y(xxxxy)(xxxxy)(xxxxy)) = 1$$
$$\Delta(\Xi, 5, y(xxxxy)(xxxxy)(xxxxy)) = 1$$

Ideje azonban kipróbálni a ConDir eljárást. A ConDir 3 állapotú Cerny típusú automatára hét állapotú automatát hoz létre, melynek működése jól megfigyelhető az accept eljárás outputján. Az *yxxy* irányító szava a Cerny(3) automatának, mivel a Φ automatát a {1,2,3} kezdőállapotból az {1} végállapotba viszi.

```
> Phi:=ConDir(Cerny(3));
```

$$\Phi := [\{\{1,2,3\},\{2\},\{3\},\{1\},\{1,2\},\{1,3\},\{2,3\}\},\{x,y\},\delta,$$
$$\{\{1,2,3\}\},\{\{2\},\{3\},\{1\}\}]$$

```
> accept(Phi,yxxy,tr1,sho,set);
-Delta begins with {{1, 2, 3}}
-Delta({{1, 2, 3}},y) = {{1, 2}}
-Delta({{1, 2}},x) = {{2, 3}}
-Delta({{2, 3}},x) = {{1, 3}}
-Delta({{1, 3}},y) = {{1}}
```

$$\Delta(\{\{1,2,3\}\},yxxy) = \{\{1\}\}$$

```
Result of Delta:
```

$$\{\{1\}\}$$

set of final states: , $\{\{2\},\{3\},\{1\}\}$

```
Result of accept:
```

true

Az eddig elmondottak szerint egy Ξ automata akkor és csak akkor irányítható, ha a ConDir(Ξ) automata által felismert nyelv nem üres. Eszerint a Ξ irányíthatósági problémája ekvivalens a ConDir(Ξ) automata ürességi problémájával, ez utóbbi pedig algoritmikusan eldönthető. Ehhez ugyanis elegendő ellenőrizni az automata állapotszámánál nem hosszabb szavak felismerhetőségét. Ez azonban exponenciális idő-bonyolultságú algoritmust eredményezne.

Most viszont abban a szerencsés helyzetben vagyunk, hogy nem egy tetszőleges automata, hanem ConDir eljárás által generált automata által felismert nyelv üres vagy nem üres voltát kell ellenőriznünk. Itt pedig kihasználhatjuk, azt a tényt, hogy a ConDir iniciálisan összefüggő automatát generál. Így ha létezik legalább egy végállapot, akkor az el is érhető a kezdőállapotból. Eszerint a ConDir által generált automata ürességi problémája konstans idő alatt ellenőrizhető, ugyanis nem kell mást tenni, mint megnézni, hogy a végállapotok halmaza üres-e. Ezt valósítja meg az isdirectable eljárás, melynek működését egy irányítható (Cerny(3)) és egy nem irányítható automatán mutatjuk meg.

Megjegyzés. A második automatát a randaut eljárással generáltuk, ami egy véletlen automatát hoz létre. Első paramétere a véletlen automata állapotainak számát, második paramétere, pedig a bemenő jelek ábécéjét határozza meg. A további két opció leszűkíti az randaut mozgásterét meghatározva, hogy a keletkező automata determinisztikus és teljes legyen. Felhívjuk továbbá a figyelmet arra, hogy az utasítássorozat ismételt végrehajtása a randaut használata miatt nem feltétlenül fogja ugyanazokat az outputokat eredményezni.

```
> isdirectable:=proc(A::automaton)
evalb(nops(ConDir(A)[5])>0)
end;
```

isdirectable :=

proc (*A*::*automaton*) evalb(0 < nops(ConDir(*A*)[5])) **end proc**

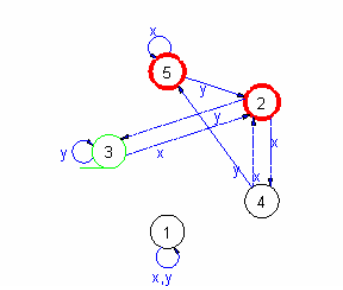
```
> isdirectable(Cerny(3));
```

true

```
> Xi:=randaut(5,{x,y},deterministic,complete);
```

$\Xi := [\{1, 2, 3, 4, 5\}, \{x, y\}, \delta, \{3\}, \{2, 5\}]$

```
> plotaut(Xi);
```




```
> isdirectable(Xi);
```

false

Annak ellenére, hogy az üresség ellenőrzése konstans idejű, az isdirectable eljárás nem hatékony. Ez abból adódik, hogy a ConDir exponenciális időben állítja elő a kívánt automatát. Valóban a ConDir időbonyolultsága csak konstansban tér el a Compose időbonyolultságától, ami O(*u***m*), ahol *m* a konstruálandó automata bemenő jeleinek száma, *u* pedig az univerzum számossága. Ez utóbbi pedig esetünkben 2^n, mert a ConDir a részhalmaz-automata egy részautomatáját állítja elő. Mindezt egybevetve az isdirectable eljárás időbonyolultsága O(2^n **m*), vagyis exponenciális.

Elméleti alapok

Imreh és Steinby 1955-ben publikált egy O(n^2 *m) időbonyolultságú algoritmust determinisztikus automaták irányíthatóságának eldöntésére. Az algoritmus alapötlete az alábbi észrevételeken alapul.

Legyen Ξ automata és jelölje $\mu(k, \Xi)$ azon *(a,b)* párok halmazát, mely párok legfeljebb *k* hosszuságú bemenő szóval összefésülhetők. A definíció alapján világos, hogy a $\mu(0, \Xi)$, $\mu(1, \Xi)$, $\mu(2, \Xi)$, ... halmazok növekvő sorozatot alkotnak, vagyis minden *k* természetes számra $\mu(k, \Xi) \subseteq \mu(k+1, \Xi)$.

Továbbá, nem nehéz belátni, hogy az a és b állapototkat az xp szó akkor és csak akkor fésüli össze, ha a $\delta(a, x)$ és a $\delta(b, x)$ állapotokat a p szó összefésüli. Ez alapján a $\mu(k, \Xi)$ kiszámítására az alábbi rekurzív formula adódik:

1. $\mu(0, \Xi) = \{(a,a) \mid a \in \Xi_1\}$,
2. $\mu(k+1, \Xi) = \mu(k, \Xi)$ union $\{(a,b) \mid$ létezik $x \in \Xi_2$, amelyre $(\delta(\Xi, a, x), \delta(\Xi, b, x))$ eleme $\mu(k, \Xi)$-nek $\}$

Végül, ha valamely k-ra $\mu(k, \Xi) = \mu(k+1, \Xi)$, akkor $\mu(k, \Xi) = \mu(\Xi)$, ahol $\mu(\Xi)$ az összefésülhető (a,b) párok halmaza.

Az eddig elmodottak szerint a Ξ automata irányítható voltának eldöntésére az alábbi eljárás adódik.

1. Állítsuk elő rendre a $\mu(0, \Xi)$, $\mu(1, \Xi)$, $\mu(2, \Xi)$, ... halmazokat mindaddig, amíg $\mu(k, \Xi) = \mu(k+1, \Xi)$ legelőször teljesül.
2. Ha ekkor $\mu(k, \Xi)$ $(= \mu(\Xi))$ az összes (a, b) állapotpárt tartalmazza, akkor Ξ irányítható, különben pedig nem.

Imreh és Steinby cikkükben egy nxn-es logikai mátrixot használnak annak jelzésére, hogy az állapothalmaz i-dik és j-dik eleme összefésülhető-e. Célszerű azonban ehelyett egy bővebb információt hordozó adatstruktúrát is választani, a Ξ automata irányító tábláját. Mint később látni fogjuk, ezt megtehetjük anélkül, hogy elrontanánk az algoritmus belső szerkezetét és ezáltal annak időbonyolultságát.

3.2.2 Definíció

Tekintsük a Ξ automata különböző a és b állapotait és legyen

T(a,b)={p| $p \in X^*$, p összefésüli a-t és b-t, és p hossza minimális}.

Képezzünk ezek után egy olyan két oszlopos táblázatot, melynek első oszlopába azokat az $[a,b]$ párokat írjuk, amelyekre T(a,b) nem üres, az $[a,b]$-val jelölt sor második oszlopába pedig magát T(a,b)-t írjuk. A keletkező táblázatot (ami megfelel a Maple tábla adatstruktúrájának) jelöljük $DirTab_{\Xi}$-vel, és nevezzük a Ξ automata **irányító táblájának**. Az $[a,b]$-vel jelölt sor második oszlopára a Maple szintaxisát követve $DirTab_{\Xi}$ [a,b]-val hivatkozunk.

Az irányító tábla tehát olyan táblázat, melyben a $DirTab_{\Xi}$ [a,b] akkor és csak akkor definiált, ha a T(a,b) halmaz nem üres, és ekkor $DirTab_{\Xi}$ [a,b] =T(a,b). Mivel irányítható automatára T(a,b) sosem üres ezért irányítható Ξ automatára

$DirTab_{\Xi}$ [a,b] minden állapot párra definiált. A következő két állítás az irányító tábla használhatóságát mutatja.

3.2.3 Propozíció

n állapotú irányítható automata egy irányító szava az irányító tábla ismeretében O(*n*) idő alatt előállítható.

Bizonyítás
Alkalmazzuk a következő algoritmust.

```
1.inicializálás
  Legyen B az automata állapothalmaza
  Legyen p az üres szó.
2.ciklus
  Mindaddig, amíg B nem egyelemű
    legyen a és b a B két tetszőleges, de különböző
eleme
    legyen q a DirTab_Ξ [a,b] egy tetszőleges eleme
    B := Δ(Ξ, B, q)   és   p := pq
3. output: p
```

Az algoritmus második lépésében az *a* és *b* állapotok, és a *q* választása miatt *q* összefésüli *a*-t és *b*-t. Megjegyezzük, hogy az előírt választás mindig lehetséges, mert a kiindulási automata irányítható.

Mivel *q* összefésüli *a*-t és *b*-t, ezért a $\Delta(\Xi, B, q)$ elemeinek száma legalább eggyel kisebb *B* elemeinek számánál. Tehát az eljárás legfeljebb *n* lépésben megáll, miközben a keletkező *p* szó az automata irányító szava lesz.

3.2.4 Propozíció

Kommutatív automata esetén az automata az irányító táblájának ismeretében előállítható minimális hosszúságú irányító szó.

Bizonyítás

Legyen $w= x_1\ x_2 \ldots x_k$ minimális hosszúságú irányító szava a Ξ kommutatív és irányítható automatának. A Ξ kommutativitása miatt bármely *x* és *y* bemenő jelre $\Delta(\Xi, A, xy) \subseteq \Delta(\Xi, A, x)$. Eszerint a

$$\Delta(\Xi, A, x_1),\ \Delta(\Xi, A, x_1 x_2),\ldots,\ \Delta(\Xi, A, x_1 x_2 \ldots x_k) = \Delta(\Xi, A, w)$$

halmazok csökkenő sorozatot alkotnak, ahol a tartalmazások *w* szó minimalitása miatt valódiak.

Eszerint az x_1 bemenő jel összefésüli az *A* halmaz legalább két állapotát,

mondjuk a_1 -et és b_1 -et, amiből az következik, hogy x_1 eleme a $DirTab_{\Xi}$ [a_1, b_1] halmaznak. Ugyanígy kapjuk, hogy az x_2 bemenő jel a $\Delta(\Xi, A, x_1)$ halmazban összefésül legalább két elemet (mondjuk a_2 -et és b_2 -et,) így , x_2 szükségképpen eleme a $DirTab_{\Xi}$ [a_2, b_2] halmaznak. Ezt a gondolatmenetet folytatva azt kapjuk, hogy a *w* szó előáll alkalmas, az irányító táblában szereplő bemenő jelek konkatenációjaként.

Az Imreh-Steinby algoritmus eredeti formájában használatos *M* logikai mátrix és a $DirTab_{\Xi}$ irányító tábla használata analóg módon történik. Kezdetben az *M* csupa hamis értékkel rendelkezik, míg a $DirTab_{\Xi}$ tábla kezdetben üres. Az eljárás során *M*[*i*,*j*] igazzá válik, valahányszor az *i* és *j* állapot párról kiderül, hogy összefésülhető, a $DirTab_{\Xi}$ [*i*,*j*] pedig definiálttá válik ugyanekkor, sőt a bejegyzésbe belekerül az összes *i*-t és *j*-t összefésülő minimális hosszúságú bemenő szó.

Eszerint az eljárás végrehajtásának bármely pontján *M*[*i*,*j*] akkor és csak akkor igaz, ha $DirTab_{\Xi}$ [*i*,*j*] definiált. És mivel a tábla adatstruktúra Maple-beli implementációja olyan, hogy a $DirTab_{\Xi}$ elemei konstans idő alatt elérhetők, ezért annak eldöntése, hogy $DirTab_{\Xi}$ definiált-e ugyanúgy konstans idő alatt dönthető el, mint az, hogy *M*[*i*,*j*] értéke igaz-e.

A $DirTab_{\Xi}$ tábla betöltheti tehát mindazt a szerepet, amit az *M* logikai mátrix betölt, miközben rendkívül hasznos további információt hordoz, nevezetesen az *i* és *j* állapotokat összefésülő összes minimális hosszú szót.

A módosított Imreh-Steinby algoritmus

Most ismertetjük az Imreh-Steinby algoritmust itt leírt módosított verzióját.

```
1.lépés. inicializálás
  T:=table([]): Verem:=üres
2.lépés:
  Az InvTab inverz átmenet tábla előállítása
3.lépés: Verem feltöltése
  minden a állapotra és x bemenő jelre
    az InvTab[a,x] minden kételemű {c,d} részhalmazára
    ha T[c,d] nem definiált, akkor
      push({c,d}): T[c,d]:={x}
    különben
      T[c,d]:=T[c,d] union {x}
4.lépés: Verem kiürítése
  mindaddig, amíg a verem nem üres
    {c,d}:=pop()
```

```
     minden x bemenő jelre
     az InvTab[c,x]×InvTab[d,x] minden kételemű {e,f}
részhalmazára
       ha T[e,f] nem definiált, akkor
         push({e,f}): T[e,f]:={xp | p eleme T[c,d]}
       különben
         T[e,f]:=T[e,f] union {xp | p eleme T[c,d]}
5.lépés: irányíthatóság eldöntése
  Ha T definiált elemeinek száma n*(n-1)/2, akkor
    az automata irányítható,
  különben
    nem.
```

3.2.5 Propozíció

A módosított Imreh-Steinby algoritmus időbonyolultsága O(m*n^2).

Bizonyítás

A számolás az eredeti algoritmus vizsgálatát végző, cikkbeli gondolatmenet értelemszerű adaptációjával történik. Az inicializálás konstans idő alatt elvégezhető. Az inverz átmenet tábla előállításához minden állapotot és minden bemenő jelet pontosan egyszer meg kell fogni, így a szükséges időigény O(n*m).

A T tábla indexei az automata állapothalmazának kételemű részhalmazai, így a T bejegyzéseinek száma legfeljebb n*(n-1)/2. Mivel miden ilyen halmazt az összes bemenő jelre meg kell vizsgálni, így a harmadik lépés időbonyolultsága O($m\,n^2$). A negyedik lépés időkorlátja megint csak O($m\,n^2$), mert bármely állapotpárt legfeljebb egyszer kell megvizsgálni minden *x* bemenő jelre. Végül az irányíthatóság eldöntése konstans idejű. Mindez O($m\,n^2$) időbonyolultságot eredményez.

3.2.6 Propozíció

A módosított Imreh-Steinby algoritmus végrehajtásával keletkező T tábla irányítható automata esetén nem más, mint az automata irányító táblája.

Bizonyítás

Valóban, a verem feltöltése lépésben egy [*c,d]* állapotpárt (c és d különböző!) összefésülő összes bemenő jel bele kerül a T[*c,d*] halmazban. Ha pedig verem kiürítése során feltesszük, hogy a ciklus indulásakor a T minden [c,d] bejegyzése igaz, hogy T[c, d] tartalmazza, a *c* és *d* állapotokat összefésülő összes minimális hosszúságú szót, akkor ez az állítás igaz marad az [e, f] állapot párra is. Ugyanis, *e* eleme InvTab[c, x]-nek, és így $\delta(e, x) = c$. Hasonlóan $\delta(f, x) = d$. Ezután

$\Delta(e, xp)$ =
= $\Delta(\delta(e, x), p)$ =
= $\Delta(c, p)$ =
= $\Delta(d, p)$ =
= $\Delta(\delta(f, x), p)$ =
= $\Delta(f, xp)$

Δ definíciója
$\delta(e, x) = c$
p összefésüli *c*-t és *d*-t
$\delta(f, x) = d$
Δ definíciója

tehát *xp* valóban összefésüli *e*-t és *f*-et.

Ha *xp* nem lenne minimális hosszú, akkor valamely *q* bemenő szóra $|xq|<|xp|$ és $\Delta(\Xi, e, xq) = \Delta(\Xi, f, xq)$ teljesülne, miből $\Delta(c, q) = \Delta(d, q)$ következne, és ez lehetetlen, mert *q* hossza határozottan kisebb, mint *p* hossza, miközben *p* minimális hosszú.

Már csak azt kell megmutatni, hogy az *e*-t és *f*-et összefésülő összes minimális hosszúságú szó benne lesz a T[*e,f*] halmazban. Tekintsük ehhez a *p* szót, és tegyük fel, hogy *p* minimális hosszú az *e*-t és *f*-et összefésülő szavak között. A *p* szó nem lehet üres szó, mert *e* és *f* különbözik. Ha *p* hossza 1, akkor a verem feltöltése lépésben belekerül a T[*e,f*] halmazba. Ellenkező esetben alkalmas *x* bemenő jelre és *q* bemenő szóra $p = xq$. Mivel *p* összefésüli *e*-t és *f*-et, a *q* szükségképpen összefésüli a $c = \delta(e, x)$ és $d = \delta(f, x)$ állapotokat, továbbá *p* minimális hossza biztosítja, hogy *q* is minimális hosszú a *c*-t és *d*-t összefésülő szavak között. Eszerint T[*c,d*] tartalmazza a *q* szót. A verem kiürítési lépésben minden kételemű állapothalmaz bekerül a verembe, így $\{c, d\}$ is. Tehát lesz olyan futása a ciklusnak, amikor a pop() utasítás a {c,d} halmazt eredményezi. Amikor a belső ciklus az *x* benemő jelet dolgozza fel, akkor T[*e,f*] halmazban az *p*= *xq* szó is belekerül. Ezzel állításunkat igazoltuk.

Implementáió

A módosított `ImrehSteinby` algoritmus implementációját a három állapotó Cerny-féle automatán mutatjuk be. Az algoritmus forrásszövege a dolgozat Appendixében található.

Az első parancs outputja *true*, jelezve, hogy a Cerny(3) automata valóban irányítható. Eljárásunk beszédesebbé válik, ha számos opciójából használunk néhányat. A `trace2` hatására az eljárás megjeleníti az automata inverz átmenet tábláját, jelzi a ciklusok futása során a verem aktuális állapotait, valamint mutatja az irányító tábla bejegyzéseinek időbeli alakulását.

```
> ImrehSteinby(Cerny(3));
```

true

```
> ImrehSteinby(Cerny(3),trace2,show);
.({1, 3},x)  : {{2, 3}} -> Inverse transition table:
```
$$[(1,x)=\{3\},(1,y)=\{1,3\},(2,x)=\{1\},(2,y)=\{2\},(3,x)=\{2\}]$$

```
Initialization:
..push({1, 3}): [0]
```
$$[\{1,3\}=\{y\}]$$

```
pop()->{1, 3} : [0]
.({1, 3},x)  : {{2, 3}} -> {{2, 3}}
..push({2, 3}): [1, {2, 3}]
```
$$[\{1,3\}=\{y\},\{2,3\}=\{xy\}]$$

```
.({1, 3},y)  : skip
pop()->{2, 3} : [0]
.({2, 3},x)  : {{1, 2}} -> {{1, 2}}
..push({1, 2}): [1, {1, 2}]
```
$$[\{1,2\}=\{xxy\},\{1,3\}=\{y\},\{2,3\}=\{xy\}]$$

```
.({2, 3},y)  : skip
pop()->{1, 2} : [0]
.({1, 2},x)  : {{1, 3}} -> {{1, 3}}
```
$$[\{1,2\}=\{xxy\},\{1,3\}=\{y,xxxy\},\{2,3\}=\{xy\}]$$

```
.({1, 2},y)  : {{1, 2}, {2, 3}} -> {{1, 2}, {2, 3}}
```
$$[\{1,2\}=\{yxxy,xxy\},\{1,3\}=\{y,xxxy\},\{2,3\}=\{xy,yxxy,yyxxy\}]$$

```
Result of ImrehSteinby:
```
$$true$$

A table opció hatására az eljárás outputja az automata irányító táblája, míg a word opció pedig egy irányító szót jelenít meg, ha az első paraméterként adott automata irányítható.

```
> ImrehSteinby(Cerny(3),table); # table
```
$$\text{table}([\{1,3\}=\{y,xxxy\},\{2,3\}=\{xy,yxxy,yyxxy\},$$
$$\{1,2\}=\{yxxy,xxy\}$$
$$])$$

```
> ImrehSteinby(Cerny(3),word);
```
$$yxxy$$

3.3 μ-automaták

Tekintsünk egy tetszőleges automatát, mondjuk a Cerny(4) automatát, és állítsuk elő az irányító tábláját.

```
> Xi:=Cerny(4);
```

$$\Xi := [\{1, 2, 3, 4\}, \{x, y\}, \delta, \{1\}, \{4\}]$$

```
> T:=ImrehSteinby(Xi,table);
```

$$T := \text{table}([\{1, 3\} = \{yxyxxxy, xyxxxy\}, \{1, 4\} = \{y, xxxxy\},$$
$$\{2, 3\} = \{yxxy, xxy\}, \{2, 4\} = \{xxyxxxy, yxxxy\},$$
$$\{1, 2\} = \{yxxxy, xxxy\},$$
$$\{3, 4\} = \{yxyxxxy, xy\}$$
$$])$$

Gyűjtsük össze az irányító táblában található minden egyes összefésülő bemenő szót.

```
> W:=`union`(map(x->rhs(x),op(op(T)))[]);
```

$$W := \{y, yxyxxxy, xy, xyxxxy, xxyxxxy, yxxxy, xxxxy, yxxy, xxy, xxxy\}$$

Hozzunk létre egy új automatát a Ξ automata állapothalmazán oly módon, hogy a *W* halmazban található szavakat tekintjük az új automata bemenő jeleinek.

```
> LL:=NULL:xc:=A:Code:=table([]):
for xi in W do
 Code[xc]:=xi:
 for xs in Xi[1] do
 LL:=LL,xs,xc,Delta(Xi,xs,xi)
 od:
 xc:=convert(convert([convert(xc,bytes)[]+1],bytes),name);
od:
eval(op(op(Code)));
```

$$[J = xxxy, D = xyxxxy, H = yxxy, E = xxyxxxy, G = xxxxy, A = y, F = yxxxy,$$
$$C = xy, B = yxyxxxy, I = xxy]$$

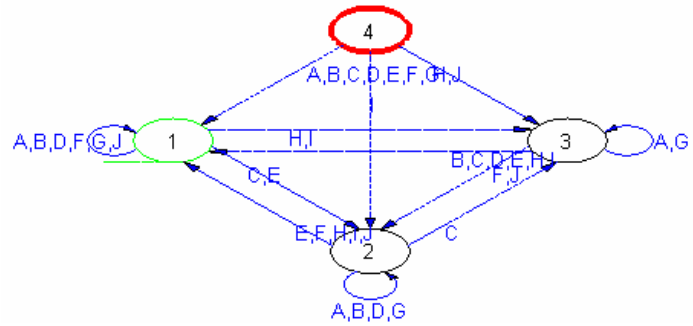
A fenti Maple ciklus létrehoz egy *Code* nevű kódtáblát, amelyben *W* halmaz minden elemére egy új bemenő jelet vezetünk be. Ezzel egy kölcsönösen egyértelmű megfeleltetést hoztunk létre a kiindulási Ξ automata irányító táblájában szereplő bemenő szavak, és az új automata bemenő jelei között.

A következő utasításban létrehozzuk az új automatát, melyet $\mu(\Xi)$ -vel jelölünk, majd megjelenítjük $\mu(\Xi)$ átmenet gráfját.

```
> mu('Xi'):=genaut([LL]);
```

$$\mu(\Xi) := [\{1, 2, 3, 4\}, \{I, D, F, C, H, J, A, B, E, G\}, \delta, \{1\}, \{4\}]$$

```
> plotaut(mu('Xi'));
```



Ideje azonban matematikai definíciót adni az eddig példán ismertetett automata konstrukcióra.

3.3.1 Definíció

Legyen Ξ automata, legyen továbbá

$W(Xi)=\{w|\ w \in X^*$, létezik olyan $a, b \in \Xi_1$, amelyre w eleme a $DirTab_{\Xi}[a,b]$ halmaznak$\}$.

Legyen továbbá Y ábécé, amelyre a $|Y|=|W(\Xi)|$ és $\phi: Y \to W(\Xi)$ kölcsönösen egyértelmű leképezés. Azt a $\mu(\Xi) = [\Xi_1, Y, \delta, \Xi_4, \Xi_5]$ automatát, amelyre a $\delta(\mu(\Xi), a, y) = \delta(\Xi, a, y\phi)$ egyenlőség minden a állapotra és $y \in Y$ bemenő jelre teljesül a Ξ automatához tartozó μ **-automatának**, vagy röviden a Ξ μ-automatájának nevezzük.

Megjegyezzük, hogy a μ-automata nem minden automatára létezik. Ugyanis, ha egy automata irányító táblája üres, akkor a definícióban szereplő $W(\Xi)$ halmaz is az. Ugyanakkor viszont minden irányítható automatának létezik μ-automatája.

A μ-automaták az eredeti automata számos tulajdonságát (pl. kommutativitás, nilpotensség, definitség) öröklik. Számunkra azonban legfontosabb a következő

3.3.2 Tétel

A Ξ automata akkor és csak akkor irányítható, ha $\mu(\Xi)$ létezik és irányítható.

Bizonyítás

Először is vegyük észre, hogy bármely $p = y_1\ y_2 \ldots y_k$ bemenő szóra és bármely a állapotra $\Delta(\mu(\Xi), a, p) = \Delta(\Xi, a, y_1\phi\ y_2\phi \ldots y_k\phi)$. Ez a w hossza szerinti teljes indukcióval könnyen bizonyítható.

Az állítás szükségességének bizonyításához tegyük fel Ξ irányítható. Ekkor az

1. Állítás szerint létezik Ξ -nek olyan w irántyó szava, ami előáll a Ξ irányító táblájában található összefésülő szavak konkatenációjaként. Legyen tehát *w* = $w_1\ w_2 \ldots w_k$, ahol $w_i \in \mathrm{W}(\Xi)$ ($i = 1$,...,*k*). A w_i ($i = 1$,...,*k*) szavak mindegyike előáll valamely *Y*-beli bemenő jel ϕ melletti képeként, tehát mindegyiknek van őse ϕ mellett.

Legyen $y_i = w_i \phi^{(-1)}$ ($i = 1$,...,*k*). Ekkor *p* = $y_1\ y_2 \ldots y_k$ irányító szó $\mu(\Xi)$ -ben. Valóban tetszőleges *a* és *b* állapotokra

$\Delta(\mu(\Xi), a, p)$ = # $\mu(\Xi)$ átmenetfüggvénye

$= \Delta(\Xi, a, y_1\ \phi\ y_2\ \phi \ldots y_3\ \phi) =$ # $y_i = w_i \phi^{(-1)}$

$= \Delta(\Xi, a, w_1\ w_2 \ldots w_k) =$ # $w = w_1\ w_2 \ldots w_k$

$= \Delta(\Xi, a, w) =$ # w irányító szava Xi-nek

$= \Delta(\Xi, b, w) =$ # $w = w_1\ w_2 \ldots w_k$

$= \Delta(\Xi, b, w_1\ w_2 \ldots w_k) =$ # $y_i = w_i \phi^{(-1)}$

$= \Delta(\Xi, b, y_1\ \phi\ y_2\ \phi \ldots y_3\ \phi) =$ # $\mu(\Xi)$ átmenetfüggvénye

$= \Delta(\mu(\Xi), b, p)$

ami bizonyítja, állításunkat.

Az elégségesség igazolásához tegyük fel, hogy a p = $y_1\ y_2 \ldots y_k$ irányító szava $\mu(\Xi)$ -nek. Megmutatjuk, hogy a *w* = $y_1\ \phi\ y_2\ \phi \ldots y_k\ \phi$ szó irányító szava Ξ -nek. Valóban legyen *a* és *b* tetszőleges állapot. Ekkor

$\Delta(\Xi, a, w)$ = # $w = y_1\ \phi\ y_2\ \phi \ldots y_k\ \phi$

$= \Delta(\Xi, a, y_1\ \phi\ y_2\ \phi \ldots y_3\ \phi) =$ # $\mu(\Xi)$ átmenetfüggvénye

$= \Delta(\mu(\Xi), a, p) =$ # p irányító szava $\mu(\Xi)$ -nek

$= \Delta(\mu(\Xi), b, p) =$ # $\mu(\Xi)$ átmenetfüggvénye

$= \Delta(\Xi, b, y_1\ \phi\ y_2\ \phi \ldots y_3\ \phi) =$ # $w = y_1\ \phi\ y_2\ \phi \ldots y_k\ \phi$

$= \Delta(\Xi, b, w)$,

amivel az álllítás igazolását befejeztük.

4 Didaktikai vizsgálatok

Az automata elméleti előadások és munkalapok kidolgozása során számos didaktikai elvvel szembesülhetünk. Különösen akkor, ha feladatunk éppen a vizsgálandó anyagrész didaktikai megközelítésű tárgyalása. A másik dolog, amivel szembesülni elkerülhetetlen, hogy a didaktikai elvek általában nem alkalmazhatóak közvetlenül, mechanikusan. Szembe kell tehát néznünk az egyes elvek újragondolásának feladatával, le kell vonni vizsgálat speciális tárgyából adódó következtetéseket.

4.1 A négyes szabály elemzése

A négyes szabályként megfogalmazott didaktikai elv a NCTM 2000 ([26]) standardok kötött látott napvilágot, és azóta számos didaktikai témájú könyvben is publikációban hivatkoznak rá (ld. [9], [18], [19]). Az elv gyakorlati alkalmazása többségében a kalkulus területén valósult meg, sőt bizonyos szerzők eleve a kalkulusra korlátozzák az elv megfogalmazását ([20]).

A függvénytan valóban kellemes terület. A függvény helyettesítési értékeinek egy sorozata példát ad numerikus reprezentációra, míg a függvény formulával való megadása illetve grafikonja a szimbolikus ill. a grafikus reprezentációt biztosítja. A függvénytulajdonságok természetes nyelven való megfogalmazása pedig biztosítja a leíró, verbális reprezentáció megjelenését.

Ez az idilli kép azonban árnyaltabbá, és komplexebbé válik, ha kilépünk a kalkulus keretiből. Ebben a fejezetben górcső alá vesszük a négyes szabály kalkuluson kívüli alkalmazásának következményeit. Megvizsgáljuk a lehetséges reprezentáció típusokat, rámutatunk a reprezentációk többszörösségére, bevezetjük a realisztikus reprezentáció fogalmát, végül a numerikus típus egy általánosítását adjuk. Mindezen lépések elvégzésével egy szélesebb körben alkalmazható didaktikai elvhez jutunk, melyet a négyes szabály általánosításának tekinthetünk.

1. Észrevétel: Reprezentációk többszörössége

A négyes szabály megfogalmazása

"every topic should be presented numerically, graphically, symbolically and verbally"

azt sugallja, hogy a matematikai problémákat (mindegyiket!) négy különféle módon, egy leíró, egy szimbolikus, egy grafikus és egy numerikus reprezentációval célszerű, sőt melegen ajánlott reprezentálni. Ugyanakkor egy adott matematikai fogalmat, problémát vagy jelenséget a négyes szabályban felsorolt reprezentációk bármelyikével általában többféle módon is reprezentálhatunk. Tehát az egyes reprezentációk nem egyértelműen

meghatározottak, és mint később látni fogjuk, sokszor célszerű több különböző grafikus vagy éppen több különböző szimbolikus reprezentációval segíteni a vizsgált jelenség megértését.

Példa

Tekintsük azt a sorozatot, amelyre

$$T_n = \frac{2\,4^n}{3} - \frac{2}{3}$$

A sorozat tehát képlettel adott, most tehát a kiindulási reprezentáció szimbolikus. A sorozat első 10 tagjának kiszámításával a sorozat egy numerikus reprezentációjához jutunk.

```
> T:=n->2*(4^n-1)/3;
```

$$T := n \to \frac{2}{3}4^n - \frac{2}{3}$$

```
> seq(T(i),i=1..10);
```

2, 10, 42, 170, 682, 2730, 10922, 43690, 174762, 699050

Megmutatható, hogy a T_n sorozat tagjai kielégítik az alábbi rekurzív definíciót:

$$T_1 = 2$$
$$T_{n+1} = 4\,T_n + 2$$

Az első egyenlőségről egyszerű behelyettesítéssel meggyőződhetünk, a másodikat pedig a szokásos Maple technikával mutatjuk meg. Képezzük a két oldal különbségét, majd az eredményt egyszerűsítjük.

```
> 4*T(n)+2-T(n+1);
```

$$\frac{8\,4^n}{3} - \frac{2\,4^{(n+1)}}{3}$$

```
> simplify(%);
```

$$0$$

A rekurzív formula az eredeti sorozat újabb szimbolikus reprezentációját adja. Írjuk most át a rekurzív definíciót oly módon, hogy a benne szereplő számokat bináris formájukban szerepeltetjük. A definíció a

$$T_1 = 10$$
$$T_{n+1} = 100\,T_n + 10$$

alakot ölti egy újabb szimbolikus reprezentációt szolgáltatva.

Az új rekurzív formula szerint a sorozat első tagja a 10 bináris szám. A másodikat úgy kapjuk, hogy ezt szorozzuk 100-zal, ami két nullának a szám végére írását jelenti, majd 10-et hozzáadunk, ami a szám végére írt két nullát 10-re változtatja.

Az elvégzett két lépés összességében azt eredményezi, hogy a sorozat következő tagja úgy keletkezik, hogy az aktuális tag után egyszerűen 10-et írunk. A második tag a 1010, a harmadik 101010, stb.. Összefoglalva azt mondhatjuk, hogy a sorozat tagjait bináris számokként az

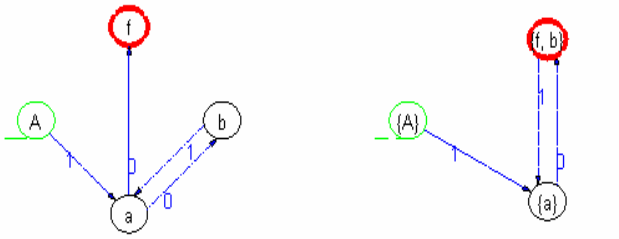
$$L = \{10, 1010, 101010, 10101010, \ldots\}$$

halmaz elemei írják le, amely L halmaz a {0,1} kételemű ábécé feletti nyelvnek is tekinthető, és mivel reguláris, felismerhető automatával. Az alábbi Maple utasítások két ilyen automata átmenet gráfját jelenítik meg. Ezzel egy csapásra két grafikus reprezentációt is szolgáltatva a kiindulási problémára.

```
> L={seq(convert(2*(4^i-1)/3,binary),i=1..5)};
```

$$L = \{10, 1010, 101010, 10101010, 1010101010\}$$

```
> with(aut):
  Xi:=genaut([A,1,a,a,0,{b,f},b,1,a],A):
  p1:=plotaut(Xi,[-2,0]):
  p2:=plotaut(ConDet(Xi),[2,0]):
  plots[display](p1,p2,scaling=unconstrained);
```



Foglaljuk össze, hogy mit is láttunk. Felvettünk egy sorozatot, amelyet első megjelenési formájában szimbolikusan reprezentáltunk, majd a rekurzív definíció decimális és bináris alakjával további két szimbolikus reprezentációt állítottunk elő. A bináris alakban adott számhalmaz nyelvként való felfogása burkolt áttérés a numerikus reprezentációról a szimbolikus reprezentációra. Így a nyelvként való reprezentációval a kiindulási halmaz negyedik szimbolikus reprezentációját adtuk meg. Végül az automata elmélet eredményeit felhasználva a sorozat két grafikus reprezentációjával fejeztük be mondanivalónkat.

Reprezentáció típusok

A fenti gondolatmenet azt példázza, hogy szó sincs arról, hogy létezne legjobb, vagy kiemelt reprezentáció. Éppen ellenkezőleg, mind a négy szimbolikus reprezentációnak megvolt a maga szerepe abban, hogy eljuthassunk a tárgyalt sorozat két grafikus reprezentációjához.

Látható tehát, hogy a négyes szabályban szereplő fogalmak (numerikus, grafikus, szimbolikus és verbális) valójában nem egy-egy reprezentációt, hanem

reprezentáció típusokat határoznak meg. A konkrét reprezentációk e típusok különféle megjelenési formái.

A reprezentáció választásának szabadsága

Ha valamiből több van, akkor felmerül a választás lehetősége és kényszere is. Milyen típusú, és az azonos típusúak közül melyik reprezentációt válasszuk egy probléma vizsgálata során?

Ez a kérdés tapasztalataim szerint komoly nehézségeket okozhat a tanulók egy részénél, ami abból az általános iskolában kialakuló szemléletből fakad, hogy a matematikai problémák megoldása, előre meghatározott, a feladat lényegéből következő szükségszerű és determinisztikus lépések sorozata. Tehát a feladat megoldásának egy meghatározott útja van, a tanulónak nincs más dolga, mint ezt az utat megtalálni. Vagy másként fogalmazva a kellő eréllyel belésulykolt megoldási lépéssorozatot az adott feladatra rekonstruálni.

Valóban nem könnyű szembesülni azzal, hogy a matematika egzakt volta nem jelenti a megoldáshoz vezető út unicitását. A megoldandó probléma tehát nem határozza meg egyértelműen azt az utat, amellyel eljuthatunk annak megoldásáig.

Hogyan válasszuk hát ki a megfelelő reprezentációt? A válasz: mi döntjük el. Szabadon választhatunk, ám azzal tisztában kell lennünk, hogy a különböző reprezentációk, sokszor még az azonos típusúak sem azonos módon támogatják gondolkodásunkat.

2. Észrevétel: Bázisreprezentáció

A különböző problémák más-más reprezentációban jelennek meg a megfogalmazásuk pillanatában. Eszerint a vizsgált jelenséget leíró lehetséges reprezentációk közül az egyik kiemelt szerepet játszik, nevezetesen az a reprezentáció, amelyen az adott fogalom, tulajdonság vagy probléma, alakot ölt, megfogalmazódik. Ezt a kitüntetett reprezentációt a probléma bázisreprezentációjának nevezzük.

Az új fogalom felhasználásával azt mondhatjuk, hogy az előző pontban bemutatott példa bázisreprezentációja a számsorozat képlettel megadása, tehát a vizsgált probléma bázisreprezentációja szimbolikus típusú.

3. Észrevétel: Reprezentációk tulajdonságai

A reprezentációk nem önmagukért való matematikai fogalmak. Egy új reprezentációt azért vezetünk be, hogy az a vizsgálat tárgyának valamely tulajdonságát kiemelje, könnyebbé, átláthatóvá, érthetőbbé tegye. Ehhez mindenek előtt biztosítani kell az egyes reprezentációk közötti átjárást. Ennek mindkét irányú megléte szükséges ahhoz, hogy az egyik reprezentációban vizsgált probléma a másik reprezentációban is megfogalmazható legyen, valamint hogy az átfogalmazott probléma megoldása az eredeti reprezentációban is leírható legyen.

Példa

Tekintsük a paraméteres egyenletrendszerrel adott

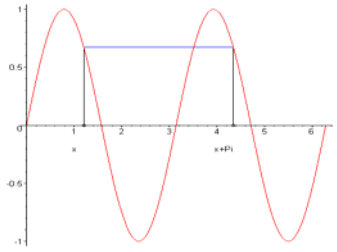
$$x = t$$
$$y = \sin(2\,t)$$

függvény görbéjét a Descartes féle koordinátarendszerben (ld. [7]). Mint tudjuk, a függvény periódusa π, és ez úgy jelenik meg a grafikus reprezentációban, hogy akármilyen értéket is veszünk, a függvény x helyen vett helyettesítési értéke megegyezik az $x+\pi$ helyen vett helyettesítési értékkel. Mindezt következő animáció is jól szemlélteti.

```
> p:=[t,sin(2*t),t=0..2*Pi];
```

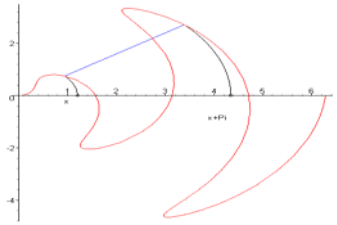
$$p := [t, \sin(2\,t), t = 0 .. 2\,\pi]$$

```
➢ period(p,1.2,cartesian);
```



Térjünk most át egy másik grafikus reprezentációra, és ábrázoljuk a görbét polár koordináta rendszerben!

```
➢ period(p,1.2,polar);
```



Bizony jelentősen megváltozott a függvény képe. Ám most minket elsősorban az érdekel, hogy a periodicitás milyen alakot ölt az új reprezentációban? Figyeljük meg, és egy kis töprengés után meg is győzhetjük magunkat erről, hogy a derékszögű koordinátarendszerben vízszintes szakasz egy origón átmenő egyenes szakaszává, míg a függőleges egyenes szakaszok origó körüli körök körívévé transzformálódnak a polár koordináta rendszerben. Ily módon függvényünk periodicitása nem mást jelent, mint hogy akármilyen értéket is választunk és rajzolunk egy sugarú és egy sugarú kört az origó körül, akkor ezen köröknek a függvénygörbével vett metszéspontjait összekötve origón átmenő egyenest kapunk.

Eléggé nyilvánvaló, hogy didaktikai szempontból a polár koordinátás reprezentációval két gond is van. Egyrészt nem könnyű (értsd nem könnyen érthető és kezelhető) az átmenet a két grafikus reprezentáció között, másrészt a vizsgált tulajdonság az eredeti reprezentációban egyszerűbben, tisztábban és ily módon könnyebben érthetően fogalmazódott meg mint az új reprezentációban. Akkor meg mi szükség van rá?

Ha tehát a vizsgált probléma a szinusz függvény periodicitása, akkor a polár koordináta rendszerben való grafikus ábrázolás kifejezetten szerencsétlen választás. Jogosan merül tehát fel a kérdés, hogy mik azok a feltételek, amelyek teljesülése esetén a reprezentáció didaktikailag jól használható?

Átfogalmazhatóság

Egy reprezentáció használhatóságának elengedhetetlen feltétele, hogy a vizsgált jelenségnek a bázisreprezentációban megfogalmazott lényegi tulajdonságai az új reprezentációban olyan alakot öltsenek, olyan alakra fogalmazódjanak át, amelynek megoldása már ismert, vagy az eredetinél könnyebben átlátható és megoldható. Ezen feltételek esetén azt mondjuk, hogy a vizsgált fogalom, tulajdonság vagy probléma az adott reprezentációra nézve átfogalmazható.

2. Példa

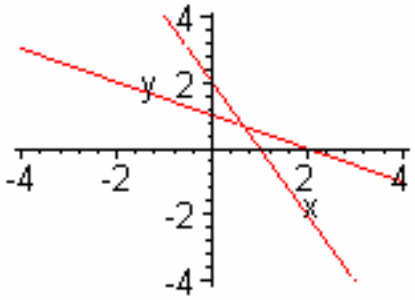
Tekintsük példaként a két ismeretlenes lineáris egyenletrendszer megoldásának feladatát. A szimbolikus típusú bázisreprezentációban megjelenő problémához könnyen adhatunk két grafikus reprezentációt (ld. [9]. Az elsőben az egyenleteket egyenes egyenletének felfogva a problémát két egyenessel szemléltetjük, míg a másodikban az x és az y változó együtthatóit, valamint a szabad tagokat két komponensű síkbeli vektorokként ábrázoljuk.

```
> M:=linalg[matrix](2,3,[1,2,2,2,1,2]):
M[1,1]*x+M[1,2]*y=M[1,3];M[2,1]*x+M[2,2]*y=M[2,3];
```

$$x + 2\,y = 2$$

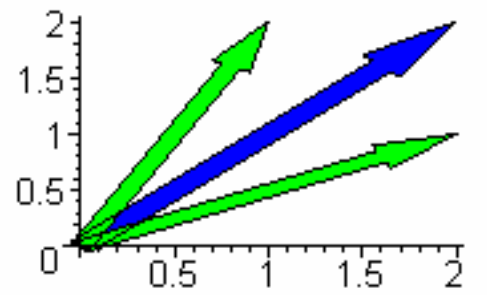
$$2\,x + y = 2$$

```
> plots[implicitplot]({%,%%},x=-4..4,y=-4..4);
```




```
> u1:=Vector([M[1,1],M[2,1]]):
  u2:=Vector([M[1,2],M[2,2]]):
```

```
u3:=Vector([M[1,3],M[2,3]]):
plots[display](plots[arrow]({u1,u2},color=green),\
               plots[arrow](u3,color=blue));
```



A bázisreprezentációban az egyenletrendszer megoldásának algebrai megkeresése a feladat. Ez a feladat második reprezentációra nézve átfogalmazható, hiszen ebben az eredeti feladat két egyenes metszéspontjának geometriai meghatározására redukálódik. Az egyenletrendszer megoldása a harmadik reprezentációra nézve is átfogalmazható. Itt ugyanis az egyenletrendszer megoldása egy vektor másik két vektor lineáris kombinációjaként való előállításának feladatára fogalmazódik át.

Ezen a ponton emlékeztetünk rá, hogy a szinusz függvény periodicitása a polár koordináta rendszerben való grafikus reprezentációra nézve nem átfogalmazható.

Végül figyeljük meg, hogy az átfogalmazhatóság reprezentációfüggő. Ez alatt az értendő, hogy ugyanaz a probléma egy adott reprezentációra nézve lehet átfogalmazható, míg más reprezentációra nézve nem. Mivel számunkra általában a probléma adott, ezért ahhoz keressük a reprezentációt, és nem fordítva. Tehát a feladat rendszerint az, hogy adott problémához olyan reprezentációt találjunk, amelyre nézve az átfogalmazható.

Átjárhatóság

A könnyű átjárhatóság a reprezentációk használhatóságának másik szükséges feltétele. Nem tekinthető ugyanis elfogadható reprezentációnak az, amelyre való átjárás túl bonyolult, nehezen elvégezhető és ily módon nem támogatja a megértést, hanem felesleges technikai nehézségeket támaszt.

Az előző példában könnyű áttérés adódik az egyes reprezentációk között. A szimbolikus reprezentációban megadott egyenletek maguk írják le az egyenesek egyenleteit. Ugyancsak könnyen végrehajtható az átjárás a szimbolikus reprezentációról a második grafikus reprezentációra is, hiszen nincs más dolgunk, mint az x változó együtthatóit egy vektor két koordinátájának felfogva, azt a derékszögű koordináta rendszerben ábrázolni. Majd ugyanezt végrehajtani az y változó együtthatóira és a szabad tagokra is.

Az átjárhatóság fontosságát kiemeli az a tény is, hogy az elemi matematika döntő

hányada nem mást tesz, minthogy a problémák különböző reprezentációi közötti átjárást gyakoroltatja, és kéri számon a tanulóktól. Amikor a tanuló szöveges feladatot old meg mondjuk két ismeretlenes lineáris egyenletrendszerrel, akkor valójában nem csinál mást, mint a megoldandó probléma leíró, szöveges reprezentációjáról áttér annak szimbolikus reprezentációjára. Ha ezekután ábrázolja a kapott egyeneseket és ezek metszéspontját határozza meg, akkor a szimbolikus reprezentációról áttér egy grafikus reprezentációra. Ugyanezt csináljuk a függvényábrázolás során is. És miközben áttérünk a grafikus reprezentációra a függvény különböző szimbolikus reprezentációban megadott tulajdonságait (párosság, periodicitás, konvexség, stb.) átfogalmazzuk a grafikus reprezentáció geometriai tulajdonságaira.

Ebben az értelemben tehát a különböző reprezentáció típusok közötti átjárás végrehajtásának képessége a tanulók matematikai tudásának mércéje.

4. Észrevétel: Realisztikus reprezentáció

Célszerű a vizsgált jelenség szempontjából jól használható reprezentációkra külön elnevezést bevezetni. Egy reprezentációt az adott problémára nézve realisztikusnak nevezünk,

1. ha a reprezentáció a probléma bázisreprezentációjáról átjárható, valamint
2. ha az adott probléma erre reprezentációra nézve átfogalmazható.

A realisztikus reprezentációk tehát nem mások, mint azok, amelyeknek a vizsgált matematikai fogalom, probléma, vagy jelenség szempontjából didaktikai hasznossága, ésszerűsége van.

Látható, hogy egy reprezentáció realisztikussága a vizsgált jelenségre nézve relatív fogalom. Ugyanaz a reprezentáció valamely problémára lehet realisztikus, míg egy másik jelenségre vonatkozva nem az.

5. Észrevétel: Kognitív hatékonyság

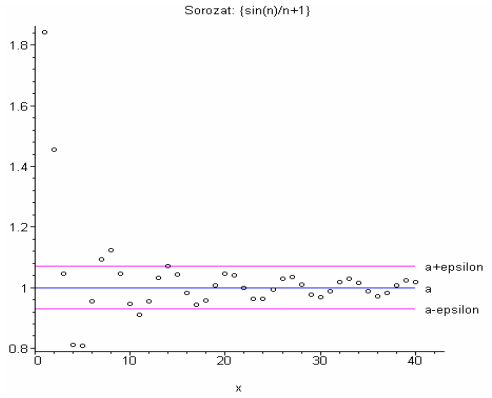
A kognitív hatékonyság fogalmát Ziegenbalg (ld.[16]) az algoritmusokra illetve a programozási nyelvekre vezeti be. A kognitív hatékonyság, vagyis az átláthatóság, a könnyen érthetőség, a szemléltető erő, a tanulók kognitív struktúrájához való varratmentes illeszkedés azonban értelemszerűen kiterjeszthető reprezentációkra is.

A különböző típusú reprezentációk, sőt azonos típusú reprezentációk más-más modelljeinek kognitív hatékonysága jelentősen eltérhet. Ugyanis a reprezentációk kognitív hatékonysága problémafüggő. Baj lenne, ha nem így lenne. A reprezentációkat pontosan ez a tulajdonságuk teszi alkalmassá arra, hogy hozzájáruljanak a gondolkodásunk fejlődéséhez, az eredményesebb problémamegoldáshoz.

3. Példa

Klasszikus példa, sok szerző tárgyalja is a kalkulus egyik legnehezebben tanítható fogalmát a sorozatok határértékét (ld. [4]). A határérték szimbolikus reprezentációja, a kvantorokkal megfogalmazott logikai formula, a tanulók számára reménytelenül nehezen érthető. Így azt kell mondanunk, hogy ebben az esetben a szimbolikus reprezentáció kognitív hatékonysága rendkívül alacsony. Kissé jobb a helyzet a verbális reprezentációval, amikor is azt fogalmazzuk meg, hogy akárhogy vesszük fel a határérték egy környezetét, mindannyiszor a sorozatnak csak véges sok tagja maradhat ki ebből a környezetből. Ám az igazán hatékony reprezentáció a Maple animációs technikájának felhasználásával adható. A konvergencia eljárás forrásszövege a dolgozat Appendixében található

- `konvergencia(sin(n)/n+1,0.07);`



Az esetek nagy részében a legjobb kognitív hatékonyságú reprezentáció grafikus típusú. Ez nem véletlen, hiszen a grafikus reprezentációk a gondolkodás ikonikus síkján fejtik ki hatásukat hatékonyan támogatva az induktív gondolkodást.

6. Észrevétel: Konkretizált reprezentáció

Mint ahogy azt már a bevezetőben említettük, a négyes szabály alkalmazása elsősorban függvénytan egyes fejezeteinek tárgyalására korlátozódik. Ennek megfelelően a numerikus reprezentációk jelesen valamely függvény helyettesítési értékeinek felsorolásában, esetleg szisztematikus, táblázatos bemutatásában öltenek testet. Példaként megemlítjük, hogy Fuchs [9]-ban a Heron iteráció bemutatásával ad numerikus reprezentációt a konvergencia fogalmára. Ugyanezt teszi Sárvári a Newton iterációval (ld. [27]).

Mi történik azonban, ha kilépünk a valós függvények köréből? Test feletti polinomgyűrű esetén a polinomok helyettesítési értékei testbeli elemek, amik lehetnek ugyan számok (számtest esetén), de nem feltétlenül azok. Beszélhetünk-e itt numerikus reprezentációról? Vagy tekintsük az automata elméletet. Elnevezhetem ugyan egy automata állapotait számokkal,...

```
> printaut(randaut(3,2));
```

STA \ INP	1
1	2
2	2
3	3

Initial state: , 1

Set of final states: , { 1, 2, 3 }

... de felvetődik a kérdés, hogy az ábrán látható átmenet tábla numerikus reprezentációnak tekinthető-e? Vagy lehet-e például az {x,y} kételemű ábécé feletti nyelvekre numerikus reprezentációt adni? Azt tudjuk a Turing gépek elméletéből, hogy a matematikai objektumokat binárisan kódolni tudjuk, ám az így keletkező numerikus reprezentáció messze nem realisztikus, így a mi tárgyalásunk szempontjából nem használható.

Szembe kell tehát néznünk azzal, hogy a numerikus reprezentáció bizonyos tárgykörökben, ill. bizonyos absztrakciós szint felett közvetlenül nem értelmezhető. Így ha nem akarunk a numerikus reprezentációról sem lemondani, sem pedig használatát pusztán kalkulusbeli tárgykörökre megszorítani, akkor meg kell próbáljuk azt általánosítani, azaz egy olyan általánosabb elvvel helyettesíteni, ami az eddig ismert esetekre a már megszokott numerikus reprezentációt adja, miközben szélesebb körben használható.

Nézzük csak meg, hogy mit is csinálunk akkor, amikor egy függvény helyettesítési értékét kiszámoljuk. Nem mást, minthogy a vizsgált általános jelenséget (jelen esetben a függvényt) konkrét számpéldákon keresztül mutatjuk be, konkrét példákkal reprezentáljuk. Tehát a vizsgált matematikai objektum vagy jelenség egy konkrét megjelenési formáját állítjuk elő, konkretizáljuk a problémát, miközben azt is feltételeztük, hogy az így keletkező konkretizált reprezentáció megjelenési formája már a tanuló számára ismert.

Figyeljük meg, hogy a konkretizált reprezentáció alacsonyabb absztrakciós szintet képviselő megjelenési formát, az általános jelenség egy-egy speciális esetét állítja elő. A konkretizált reprezentáció a numerikus reprezentáció általánosítása, hiszen valós számokon értelmezett matematikai objektumokra alkalmazva, valós számokat eredményez, és ily módon a vizsgált jelenség numerikus reprezentációját adja.

Összefoglalás

Mint a nagyszerű és kellően mély elvek általában, a négyes szabály is rendkívül egyszerű formában ölt testet. Alkalmazásának általános igénye arra készteti a felhasználóját, hogy részletesen gondolja át és rendszerezze reprezentációkkal kapcsolatos fogalmait.

Mint láttuk az egyes reprezentációs típusok sok esetben többszörösen jelennek meg egy probléma vizsgálata során. Ennek megfelelően szó sincs négy különböző reprezentációról, legfeljebb négy különböző reprezentációs típusba tartozó reprezentációk kellő számú, néha csak kettő néha akár öt-hat alkalmazásáról. Ennek megfelelően célszerűbbnek tartanánk a négyes szabály helyett a tárgyalt elvet **többszörös reprezentáció** szabályának nevezni, melyet a most bevezetett fogalmaink segítségével az alábbi formában fogalmazhatunk meg. **A matematikai problémákat, fogalmakat, jelenségeket ajánlatos több, a vizsgálat tárgyára nézve realisztikus reprezentációval szemlélhetni**. A többszörös reprezentáció elvének javasolt formája tehát

"every topic should be presented by multiple realistic representations".

Ezek a reprezentációk a konkretizált, a grafikus, a szimbolikus és a verbális reprezentáció típusokból kerülhetnek ki. A konkretizált reprezentáció típus bevezetésével a négyes szabály általánosításához jutunk, míg a realisztikus reprezentációkra való szorítkozás biztosítja, hogy a tárgyalás során felmerülő reprezentációknak a vizsgált matematikai fogalom, probléma, vagy jelenség szempontjából didaktikai ésszerűsége legyen, és ily módon hozzájáruljanak a jobb megértéshez és a tárgyalás nagyobb kognitív hatékonyságához.

4.2 Fogalmak tanítása

A didaktikai és elveket, amelyek a CAS felhasználásának didaktikai kutatásai eredményeiként jöttek létre, a matematika számos területén alkalmazzák. Első sorban a kalkulusbeli, a lináris algebrai, és geometriai alkalmazások érdemelnek említést. Ugyanakkor az automata elmélet fogalmai is kiválóan alkalmasak arra, hogy segítségükkel illusztráljuk a különböző didaktikai elvek gyakorlati alkalmazásait.

Ebben a fejezetben kidolgozzuk a véges állapotú gépek matematikai koncepciójának didaktikai megközelítésen alapuló tárgyalását. A való világ problémáiból indulunk ki (**Archimédeszi elv**). A véges állapotú gépek intuitív fogalmát használjuk arra, hogy bevezessük és implementáljuk a **véges automaták két reprezentációját**: az átmeneti táblát és az átmenet mátrixot. Ebben a fázisban a Maple automata elméleti csomagjának számos eljárását használjuk, ám ezek mindegyikét **fekete doboznak** tekintjük. Nem törődünk ugyanis azzal, hogy az egyes eljárások hogyan dolgoznak. Elegendő tisztában lenni felhasználásukkal, mivel ezeket az eljárásokat jelen tárgyalási szakaszban kísérleti eszközökként használjuk.

A didaktikai cél az **asztrakció folyamatának megvilágítása**. A matematikai modell megalkotása során olyan tisztán matematikai eszközöket kell találnunk, amelyek alkalmasak arra, hogy leírják a szekvenciális gépek összetevőit és működését. A munkát tapasztalatgyűjtéssel kezdjük. Az automata elméleti csomagot használjuk arra, hogy tapasztalatokat gyűjtünk az általánosított

átmenetfüggvény működéséről. Kísérleteink rávezetnek bennünket arra, hogy az összetett függvény képzés alkalmas eszköz olyan valóságos dolgok modellezésére, mint a bementi szalag és az olvasó fej. Földerítjük az általánosított átmenetfüggvény legfontosabb tulajdonságait, mely alapján képesek leszünk formálisan definiálni a determinisztikus automata és az általánosított átmenetfüggvény fogalmát, valamint a szavak és nyelvek felismerését. A **tanulási folyamat végére** teljes egészében megértjük az implementáció működését, mely révén **az eddig fekete dobozként működő eljárások fehér dobozzá vállnak**.

Ezek után folytatjuk az absztrakció folyamatát. Az absztrakció első lépésével szekvenciális gépek intuitív fogalmára alapozva megalkottuk a determinisztikus automata fogalmát. Az absztrakció következő lépésében erre a determinisztikus modellre alapozva felépítjük a nemdeterminisztikus automata fogalmát . Ugyanúgy, mint korábban, most is az induktív eljárást használjuk: **kísérletezés**, az eredmények és **megfigyelések formalizálása**, **általánosítás** és az **új koncepció bevezetése**.

A kiinduló pontunk a determinisztikus automata átmenet gráfja, amely a determinisztikus viselkedésnek köszönhetően, bizonyos speciális tulajdonságokkal rendelkezik. Azt kérdezzük, hogy mi történne, ha elvetnénk ezeket a tulajdonságokat, és megpróbálnánk minden egyes irányított címkézett gráfot valamilyen képzeletbeli automata átmenetfüggvényének tekinteni? Teljes egészében leírjuk ennek a feltevésnek a következményeit. Először azt kell realizálnunk, hogy megváltozik az átmenetfüggvény koncepciója. Ez a változás ugyanakkor azzal jár, hogy olyan új fogalmakat kell bevezetnünk: lehetséges futások halmaza futás helyett, lehetséges állapotok halmaza az aktuális állapot helyett. Fölfedezünk egy a bemenő szó feldolgozását leíró újabb reprezentációt, amelyet a lehetséges futások fájának nevezünk. Hasonlóan a determinisztikus esethez a kísérleteink végére mindent tudunk ahhoz, hogy formalizáljuk a nem determinisztikus automata definícióját. Didaktikai szempontból nézve tehát a **nem determinisztikus általánosított átmenetfüggvény fekete dobozából fehér dobozt készítünk**.

A tanulás egy időben zajló folyamat megállásokkal, visszacsatolásokkal, visszalépésekkel, kísérleti fázisokkal tarkítva. A **kísérletek** alapvető szerepet játszanak a megfigyelés során a tapasztalatok összegyűjtésében. Az itt felmerülő legnehezebb kérdés az, hogy **mennyi időt kell töltsünk kísérletezéssel**? Más szavakkal azt is kérdezhetnénk, hogy hányszor kell egy kísérletet megismételni annak érdekében, hogy az osztályban lévő minden egyes tanuló képes legyen az eredmények felvázolására, a tapasztalt jelenségek mögöttes meghúzódó általános szabályok megsejtésére és azok formalizálására? A nehézség itt abban áll, hogy az osztály tanulóinak felfogó képessége jelentősen eltéréseket mutathat.

A válaszunk erre a kihívásra a CAS használata. A Maple, mint bármelyik más interaktív CAS, lehetőséget ad a felhasználónak utasítások adott sorozatának többszöri, ismételt végrehajtására, miközben a felhasználó a számításban

résztvevő adatok módosítását is könnyedén elvégezheti. Mindössze a Maple munkalapok szerkezeti strukturálása mellett be kell vezetnünk a **munkalapok didaktika strukturálását**. A tanulók figyelmét fel kell hívni arra, hogy melyik a kezdőpontja és hol a végpontja azoknak egymást követő és logikailag összefüggő Maple utasításoknak, amelyek alkalmasak egy jelenség valamilyen szempontú megfigyelésére, vagyis amelyek kísérletet alkotnak. Ha a Maple eljárásainkat és utasításainkat úgy szervezzük, hogy a kísérlet első utasítása azokat az adatokat tartalmazza, amelyektől az adott kísérlet függ, akkor ez az első utasítás egy **PoP** (Point of Practice) **helyet definiál** a tanulási eljárás folyamatában. A POP pontot tehát arra használjuk, hogy megindítsunk vele egy kísérletet és segítsünk a tanulóknak abban, hogy megtalálja azokat az adatokat, amelyeket megváltoztathat a kísérletezés során. A kísérletet alkotó utasítások sorozatának utolsó parancsát **EoP**-vel (End of Practice) jelöljük.

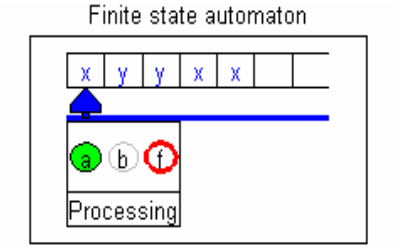
Ily módon tehát a tanítási anyagunk számos PoP-EoP párt tartalmazhat, amelynek mindegyike egy-egy kísérlet első illetve utolsó utasítását jelöli ki. A kísérlet természetesen nem csak Maple utasításokat és azok outputjait, hanem szöveges magyarázatokat, értelmezéseket is tartalmazhat. Nem tűnik indokoltnak a PoP-EoP párok egymásba skatulyázása, az átfedések pedig egyenesen zavaróak. Eszerint azt is mondhatjuk, hogy **Maple munkalapunk didaktikai struktúráját idegen PoP-EoP párok alkotják**. A kísérlet input adatait a POP utasítás tartalmazza. A kísérlethez tartozó utasítások sorozatát a tanulók annyiszor hajtják végre, ahányszor szükségük van a kísérlet eredményeinek megértéséhez és az eredmények helyes interpretálásához.

Determinisztikus eset

Motiváció

Intuitíven úgy gondolunk egy determinisztikus automatára, mint egy szekvenciális véges állapotú gépre, melynek van egy input szalagja és egy olvasó feje. Az input szalag tartalmazza a bemenő szót, amelyet az olvasó fej balról-jobbra olvas el karakterenként. Az automata működése mindig a kezdő állapotból indul. Az olvasófej leolvassa a legelső input jelent, melynek hatására az automata megváltoztatja állapotát. Ezt a változást az automata átmenet függvénye írja le. Ezek után az automata olvassa a következő bemenő jelet, ismét állapotot vált és mindezt addig végzi, amíg teljes egészében el nem olvasta az input szalagon található összes jelet. Azt a három lépéses eseményt, amely az aktuális input jel olvasásából, az állapot változásból és az input olvasó fej jobbra léptetéséből áll, átmenetnek nevezzük. Eszerint tehát az automata működése nem más, mint átmenetek egymás utánja. Ha az állapot, amelybe az automata jut, miközben végig olvasta a szalagon található összes jelet, végállapot, akkor azt mondjuk, hogy az automata a bemenő szót felismeri, vagy elfogadja.

➢
```
showaut(genaut([a,x,b,a,y,f,b,y,a,b,x,f,f,y,a,f,x,b]),\
                        xyyxx);
```



Implementáció

Amit eddig elmondtunk a véges automatákról, elegendő ahhoz, hogy megalkossuk annak Maple-beli implementációját. Azzal kezdjük, hogy az automata állapotai összegyűjtjük egy véges halmazba, amelyet állapothalmaznak nevezünk. Mivel a Maple kezeli a halmaz adatstruktúrát, könnyedén tudunk létrehozni egy olyan S halmazt, amely az a, b és f állapotokból áll.

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

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

Az automata input szalagja egy input szót tartalmaz, amely nem más, mint bemenő jelek véges sorozata. Összegyűjtve az összes lehetséges bemenő jelet ismét csak egy véges halmazhoz jutunk, amelyet ábécének nevezünk. Jelöljük ezt az ábécét X-szel.

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

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

Vegyük észre, hogy az intuitív fogalmunk nem mond semmit arról, hogy hány különböző bemenő jel lehetséges, de biztosan érezzük, hogy ennek a halmaznak is végesnek kell lennie. Furcsa lenne ugyanis valamit végesnek nevezni, ha annak valamely komponense végtelen.

Ezek után implementáljuk az átmenetfüggvényt, amelyet delta-val jelölünk. Ez a függvény azt az állapotot határozza meg, amelybe az automata átmegy, miközben az aktuális állapotban olvassa az aktuális bemenő jelet. A delta függvény tehát olyan párokon dolgozik, amelynek első komponense egy állapot, második komponense egy input jel, értéke pedig a következő aktuális állapot.

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

A delta[a,x]:={b}: utasítások azt írja elő, hogy az *x* bemenő jel hatására az automata az *a* állapotból átmegy a *b* állapotba, míg a delta[a,y]:={a}: utasítás szerint az *y* bemenő jel az *a* állapotot változatlanul hagyja. A további parancsok hasonlóan interpretálhatók.

Vegyük észre továbbá, hogy a Maple tábla adatstruktúráját használtuk arra, hogy az átmenetfüggvényt implementáljuk. Ebben semmi meglepő nincs. Eléggé

természetes megoldás véges halmazokon értelmezett függvényeket táblázatos formában leírni. Az elkövetkezendőkben nem különböztetjük meg az átmenetfüggvényt annak implementációjától. függetlenül attól, hogy a Maple más szintakszist követel meg. Tehát nem teszünk különbséget a Maple szintaxis által előírt δ [a,x] és a matematikában szokásos δ (a,x) jelölés között.

Visszatérve az implementáció folyamatára, az automatának rendelkeznie kell egy kezdő állapottal és végállapot halmazzal. Hasonlóan ahhoz, ahogy ezt a δ specifikációjánál az a kezdőállapot helyett az {a} egyelemű halmazt használjuk. A végállapotok halmazát pedig F-rel jelöljük.

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

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

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

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

Azzal fejezzük be az implementációt, hogy összegyűjtjük az öt komponenst egy ötelemű listába...

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

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

... amelyet mint a következő utasítás is mutatja a Maple automaton típusú objektumnak ismer fel. .

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

$$true$$

Sajnos az $[\{f, a, b\}, \{x, y\}, \delta, \{a\}, \{f, b\}]$ lista nem sokat mond az átmenetfüggvény viselkedéséről, kivéve a nevét. Szükségünk van tehát egy beszédesebb reprezentációra. A printaut eljárást arra használjuk, hogy megjelenítsük az automata átmenet tábláját. Ennek első oszlopában láthatjuk az automata állapotait, míg első sorában az automata bemenő jeleit. Az *s* állapottal címkézett sor és az ξ bemenő jellel címkézett oszlop kereszteződésében pedig az átmenetfüggvény $\delta(s, \xi)$ értéke látható.

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

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

$$Initial\ state:, a$$

$$Set\ of\ final\ states:, \{f, b\}$$

Az átmenet tábla segítségével könnyedén leírhatjuk az automata különböző

átmeneteit. Például, ha az automata a *b* állapotban van, és mondjuk az *y* bemenő jelet olvassa, akkor a következő állapot $a = \delta(b, y)$. Ez az egyenlőség az átmenetet teljes egészében leírja.

Végül, mivel a δ minden állapotra és minden bemenő jelre definiált, ezért a következő állapotot az átment tábla segítségével az összes lehetséges állapotra és bemenőjelre meghatározhatjuk.

Futások

Vegyük észre, hogy elvesztettük a bemenő szalagot és az olvasó fejet az implementációs eljárás során. Valóban, az implementált automata mindegyik összetevője matematikai objektum: halmaz, tábla, lista. Az implementáció tehát absztrakció útján jött létre. Ugyanakkor, ha nincs input szalagunk, és olvasó fejünk, akkor valamilyen matematikai eszközt kell találnunk arra, hogy leírjuk az átmenetek sorozatát.

Képzeljük el, hogy a Ξ automata az *xxy* szót készül feldolgozni. Kezdetben Ξ az a kezdőállapotban van és a szó első betűjét olvassa, ami *x*. A következő állapot a delta átmenetfüggvény segítségével határozható meg.

$$1.\ \text{átmenet} : b = \delta\ [\ a, x\ ].$$

Tehát az automata a b állapotban van, amikor a második bemenő jelet olvassa, ami szintén x-szel egyenlő.

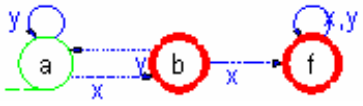
$$2.\ \text{átmenet} : f = \delta\ [\ b, x\ ].$$

Végül Xi az utolsó jelet olvassa, ami y, ami az f állapotot változatlanul hagyja .

$$3.\ \text{átmenet} : f = \delta\ [\ f, y\ ].$$

Ezek a lépések nagyon jól megfigyelhetők az automata átmenet gráfján, amelyet a `plotaut` eljárással állíthatunk elő.

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



Látható, hogy az átmenet gráf egy olyan irányított gráf, amelynek csúcsai reprezentálják az automata állapotait, míg a címkézett élek az automata átmeneteit írják le. Vegyük az átmenet gráfnak *a*-val jelölt csúcspontját, és kövessük azt a nyilat, amely *x*-szel van címkézve. A *b* állapottal jelzett szögpontba érünk, amely azt jelzi, hogy az automata új állapota a *b* állapot. Ebből az állapotból követjük az *x*-szel címkézett nyilat és elérünk az *f* állapotba, amely változatlanul marad, ha az automata olvassa *y*-t, mivel az *y*-nal címkézett él hurokél. Láthatjuk tehát, hogy az átmeneti gráf egy világos, nagyon jól használható **grafikus reprezentációját** adja az átmenetek sorozatának.

Helyettesítsük most a 2. átmenetet leíró egyenlőség jobb oldalát a 3. átmenetet leíró egyenlőség jobb oldalán található *f* helyébe! Az

$f = \delta(\delta(b, x), y)$ egyenlőséghez jutunk. Helyettesítsük ezek után ebbe az egyenlőségbe az 1. átmenet jobb oldalát. Azt kapjuk, hogy

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

Az egyenlőség jobb oldalán az átmenetfüggvény önmagával vett kompozícióját látjuk. A $\delta(\delta(\delta(a, x), x), y)$ pedig nem más mint az állapot, amelybe az automata elmozdul, miután végigolvasta az *xxy* bemenő szót. Hasonlóan, ha például arra lennénk kíváncsiak, hogy az automata mely állapotba jut az *yx* bemenő szó hatására, akkor a válasz $\delta(\delta(a, y), x)$. Ezen a ponton tehát megtaláltuk azt a matematikai eszközt, amely képes arra, hogy leírja átmenetek sorozatát, és ez az eszköz nem más, mint a **függvények kompozíciója**.

Az egyszerűség kedvéért bevezetünk egy új függvényt, Δ -t, amely bemenő szavakon dolgozik bemenő jelek helyett. Nevezetesen Δ meghatározza azt az utolsó állapotot, amelybe az automata elmozdul az aktuális állapotból, miközben végigolvassa a *w* bemenő szót. Például ha $w = xxy$ akkor

$$\Delta(a, xxy) = \delta(\delta(\delta(a, x), x), y) = \delta(\delta(b, x), y) = \delta(f, y) = f$$

Ez a koncepció a Delta eljárással került Maple-ben implementálásra. A Delta első paramétere maga az automata, a második egy állapot, míg a harmadik az input szó. Nézzük, hogyan dolgozik a Delta eljárás!

```
> w:=xxy;                          # PoP (Point of Practice)
                                   #   próbálkozzunk   más   bemenő
szavakkal is
                                   #   pl.   w:=xyyyx,   w:=x,   w:=``
(üres szó)
```

$$w := xxy$$

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

$$f$$

A Delta valóban kiszámolta az aktuális állapotot, de nem mondott semmit a számolás részleteiről. Ugyanakkor, ha használjuk a tr1 opciót, akkor Delta megjeleníti a köztes állapotokat is, mindazokat, amelyeket az automata működése során érintett.

```
> Delta(Xi,a,w,trace1):
-Delta begins with a
-Delta(a,x) = b
-Delta(b,x) = f
-Delta(f,y) = f
```

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

Világos, hogy az automata állapotok sorozatait érinti a bemenő szavak feldolgozása során. Nevezetesen pontosan annyi új állapotot, ahány jelből áll a bemenő szó. Azoknak az állapotoknak a listáját, beleértve az aktuális állapotot is, amelyeket az automata érint a *w* szó feldolgozása során, az automata ***w* bemenő szóhoz tartozó futásának nevezzük**. Például az $[a, b, f, f]$ lista a Ξ automata $w = xxy$ bemenő szóhoz tartozó futása. Vegyük észre, hogy az automata w bemenő szóhoz tarozó futásának első komponense az állapot, amelyben az automata megkezdi, utolsó komponense pedig az állapot, amelyben az automata befejezi a w bemenő szó feldolgozását.

Ha használjuk a runs opcióját, akkor Delta az utolsó állapot helyett a harmadik paramétereként megadott bemenő szóhoz tartozó futást jeleníti meg.

```
> Delta(Xi,a,w,runs);          # EoP (End of Practice)
```

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

Vegyünk két bemenő szót, *u*-t és *v*-t. A $\Delta(s, u)$ az állapot, amelybe az automata az s állapotból kiindulva kerül az *u* szó feldolgozása során. Jelöljük ezt az állapotot *t*-vel, és tekintsük a $\Delta(t, v)$ állapotot. Az *t* helyébe $\Delta(s, u)$ -t helyettesítve $\Delta(\Delta(s, u), v)$ -t kapjuk. Hogy érte el az automata ezt az állapotot? Elindult az *s*-ből, feldolgozta az *u* szót, majd a kapott állapotból indulva feldolgozta a *v* szót. Eszerint automatánk valójában a $w = uv$ szót dolgozta fel. Ezt az észrevételt az alábbi egyenlőséggel formalizálhatjuk. Bármely *s* állapotra, valamint *u* és *v* bemenő szavakra

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

Ez az egyenlőség azt a tényt fejezi ki, hogy ha az input szót két résszóra bontjuk, akkor a feldolgozást két lépésben elvégezhetjük. A feldolgozást kezdjük az első részszóval, majd abból az állapotból, amelybe az automata elmozdult az első résszó után, elkezdhetjük a második részszó a feldolgozását. (Mi a kapcsolat az *u*-hoz és *v*-hez tartotó futások és a $w = uv$ bemenő szóhoz tartozó futás között?)

```
> u:=yx:v:=xx:          # PoP
                # más szópárokat is próbáljunk ki!
                # u:=x: v:=yyy:
                # u:=xxy: v:=``:
                # u:=``: v:=yxx:
> 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
```

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

Vegyük észre, hogy Δ a δ átmenetfüggvény általánosításának tekinthető, mivel egyszerű input jelekre a két függvény megegyezik. Továbbá praktikussági szempontból Δ-t célszerű az üres szóra is definiálni azon a természetes módon, hogy az üres szó bármely állapotot változatlanul hagy. Mindezt a következő egyenlőségekkel írhatjuk le

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

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

ahol ε jelöli az üres szót.

Fogalomalkotás

Ezen a ponton mindent előkészítettünk ahhoz, hogy bevezessük az automata matematikai fogalmát, valamint az általánosított átmenetfüggvényt, amely megfelelő és adekvát eszköznek bizonyult az automata működésének leírására.

Definíció 1

Véges automata alatt egy Ξ =[A,X, δ ,S,F], rendezett ötöst értünk, ahol

1. A véges nem üres halmaz. Az A elemeit a Ξ állapotainak nevezzük.
2. X véges nem üres halmaz, melyet ábécének nevezünk. Ennek elemei a Ξ bemenő jelei.
3. δ : A×X->A leképezés, a Ξ átmenet függvénye.
4. S egyelemű részhalmaza A-nak. S egyetlen elemét az automata kezdőállapotának nevezzük.
5. F (üres vagy nem üres) részhalmaza A-nak, a végállapotok halmaza.

Definíció 2

A Δ : AxX *-> A általánosított átmenet függvényt a következő rekurzív definícióval adjuk meg. Bármely s állapotra és x bemenő jelre

1. $\Delta(s, \varepsilon) = s$
2. $\Delta(s, wx) = \delta(\Delta(s, w), x)$

Definíció 3

Azt mondjuk, hogy a Ξ automata elfogadja (vagy felismeri) a w bemenő szót, ha $\Delta(s, w) \in F$, ahol s az automata kezdőállapota ($S = \{s\}$).

Nemdeterminisztikus eset

Az előző pontban kifejlesztettünk egy matematikai modellt a determinisztikus szekvenciális gépekre. Most folytatjuk az eljárást. Tovább absztrahálunk és bevezetjük a nemdeterminisztikus automata fogalmát.

Motiváció

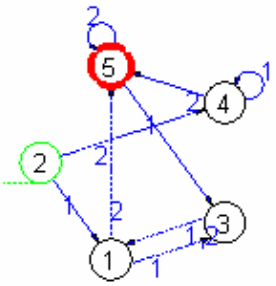
Ahogy már korábban láttuk, a determinisztikus automatákat reprezentálhatjuk címkézett irányított gráfokkal. A következő eljárás egy kételemű ábébén dolgozó véletlen ötelemű automata átmenet gráfját mutatja meg. Figyelem! A ploaut parancsot többször végrehajtva más-más eredményhez jutunk.

```
> numberofstates:=5;                    #PoP
  numberofsignals:=2;
```

$$numberofstates := 5$$

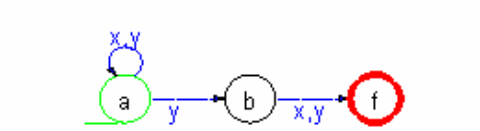
$$numberofsignals := 2$$

```
➢ plotaut(randaut(numberofstates,numberofsignals,\
        deterministic));                                  #EoP
```



Figyeljük meg, hogy ez a gráf rendelkezik egy speciális tulajdonsággal. Nevezetesen, bármely *s* állapotra és *x* bemenő jelre pontosan egy olyan él létezik, amely az *s*-sel címkézett csúcsból indul és az *x* bemenő jellel van megcímkézve. Mi történik, ha elhagyjuk ezeket a megszorításokat és egy tetszőleges címkézett irányított gráfot, valamilyen automata átmenet gráfjának szeretnénk tekinteni? Tekintsük például a következő ábrát.

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



Ez a gráf, az átmenet gráfok minden alapvető tulajdonságával rendelkezik. Valóban, a szögpontok állapotokkal vannak megcímkézve, az irányított élek pedig bemenő jelekkel. Nem rendelkezik viszont a gráf az imént feltárt speciális tulajdonsággal. Az *a* állapotból két olyan él is indul, ami az *y* bemenő jellel van cimkézve, továbbá az *f* állapotból egyetlen él sem indul.

Átmenetileg vonatkoztassunk el attól a ténytől, hogy ezt a gráfot nem tekinthetjük egyetlen (determinisztikus) automata átmenet gráfjának sem, és játszunk el a gondolattal, mintha ez a gráf mégis egy képzeletbeli automata átmenet gráfja lenne! A következő kérdésekre kell megtalálnunk a választ. Mi lesz a hatása ennek a feltételezésnek az automata definíciójára? Ha megtaláljuk a megfelelő új definíciót, akkor hogyan kell leírnunk az új automata működését? Végül hogyan kell definiálnunk a szavak, illetve nyelvek felismerését ebben az új automata fogalomban?

Nem determinisztikus átmenetfüggvény

Két dolog minden bizonnyal változatlan marad. Az állapotok *S* halmaza és az input jelek *X* halmaza. Hogy képzelhetjük azonban el az automata működését?

Lehetséges futások halmaza

Tekintsük az *yx* bemenő szót, és tegyük fel, hogy az aktuális állapot *a*. A determinisztikus esetben az átmeneti gráfon úgy kerestük ki a következő állapotot, hogy vettük az aktuális állapottal jelölt szögpontot és követtük a bemenő szóban található jelekkel címkézett éleket. Mivel minden állapotból egyetlen olyan él indult, ami adott bemenő jellel volt címkézve, a feldolgozás minden lépése meghatározott volt.

Most azonban két olyan él is indul az *a* állapotból, amely *y*-al van megcímkézve, az új állapot tehát lehet *a* is, és lehet *b* is. Úgy tűnik, hogy képzeletbeli automatánknak két lehetséges választása van. Ha az *a* állapotot választja, akkor az *x* jel azt a *b* állapotba viszi. Ebben az esetben az $[a, a, b]$ lehetséges futáshoz jutunk. Ugyanakkor, ha az automata a *b* állapotot választja az első átmenetben, akkor a második átmenet az *f* állapotot eredményezi. Ezáltal egy másik lehetséges futáshoz, $[a, b, f]$ -hez jutunk.

Első észrevételünk tehát az, hogy a nem egyértelműen címkézett élek következtében képzeletbeli automatánk működése nemdeterminisztikussá válik. Az automatának választania kell a lehetséges átmenetek között, ám semmiféle

eszköz nincs arra, amely meghatározná, hogy melyiket kell választania. Ily módon az egyetlen dolog, amit tehetünk, hogy összegyűjtjük az összes tehetséges futást, más szóval meghatározzuk az adott bemenő szóhoz tartozó lehetséges futások halmazát. A *w* bemenő szóhoz tartozó lehetséges futások halmazát $\mathrm{LFH}(w)$ -vel jelöljük.

```
> w:=yx;            # PoP
                    # Próbálkozzunk más bemenő szavakkal is!
                    # Mi lesz az üres szóhoz tartotó LFH?
```

$$w := yx$$

A következő utasítás előállítja $\mathrm{LHF}(w)$ -t.

```
> Delta(Phi,a,yx,runs);      # EoP
```

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

Mi lesz a hatása a definiálatlan átmeneteknek? Tegyük fel, hogy az aktuális állapot *b* és legyen az input szó *xy*. Az első átmenet determinisztikus, az új aktuális állapot *f*. Ezek után az automatánk olvassa az *y* jelet, de nincs ilyen átmenet definiálva. Ekkor - nem mondhatunk mást - a működés megszakad.

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

$$\{\ \}$$

Az *xy* szóhoz tartozó lehetséges futások halmaza - $\mathrm{LHF}(xy)$ - üres.

Szavak felismerése

A determinisztikus modellben az átmeneti gráf segítségével úgy döntöttük el egy szó elfogadását, hogy végigjártuk a gráfban a bemenő szó betűivel címkézett éleket, és megnéztük, hogy a végállapotba jutottunk-e. Eszerint nem csináltunk mást, mint kerestünk olyan utat az átmeneti gráfban, ami kezdőállapotból indul, a bemenő jeleivel van megcímkézve, és végállapotba jut.
Vegyük észre, hogy ez a megfogalmazás változatlan formában átvihető az új átmeneti gráfra! Keresnünk kell az átmeneti gráfban olyan utat, ami kezdőállapotból indul a bemenő szó állapotaival van megcímkézve, és végállapotba vezet. A szavak felismerése tehát természetes módon átvihető képzeletbeli automatánkra.

```
> w:=yx;                    # PoP
          # adjunk w-nek más-más bemenő szót értékként, és
          # döntsük el, hogy felismerhető-e a most bevezetett
          # felismerési koncepció szerint.
```

Az *yx* szót az automata felismeri, mert $\mathrm{LFH}(yx)$ -ben van olyan futás, amelynek utolsó állapota végállapot.

```
> Delta(Phi,a,w,runs);      # EoP
```

Lehetséges állapotok aktuális halmaza

Tekintsünk most a $w = yy$ bemenő szót, és állítsuk elő a *w*-hez tartozó lehetséges futások halmazát, $\mathrm{LFH}(w)$ -t.

```
> w:=yy;                    # PoP
```

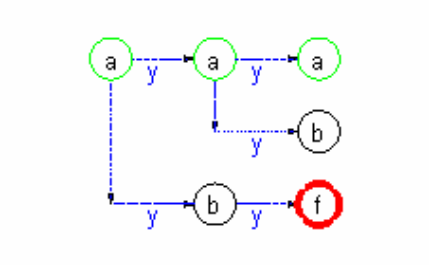
$$w := yy$$

```
> Runs:=Delta(Phi,a,w,runs);
```

$$Runs := \{[a, a, a], [a, a, b], [a, b, f]\}$$

Ezt a halmazt reprezentálhatjuk egy címkézett, ciklusmentes irányított gráffal.

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



Figyeljük meg, hogy ez a reprezentáció nagyon szépen mutatja az automata lehetséges működéseit a $w = yy$ bemenő szó feldolgozása során. Mindig elágazást találunk ebben a fában, hogy ha valamelyik átmenet nem egyértelmű. A fa különböző útjai a különböző futásokat reprezentálnak. Ezt a gráfot a ***w* bemenő szóhoz tartozó futások fájának** (w-hez tartozó FF) nevezzük és $\mathrm{FF}(w)$ -vel jelöljük. Gyűjtsük most össze az összes olyan állapotot, amely a fa gyökerétől ugyanolyan távolságra van!

```
> 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;
>
```

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

Figyeljük meg, hogy a lista második halmaza azokból az állapotokból áll, amelyek az *a* állapotból elérhetők az *y* bemenő jel hatására. Hasonlóan a harmadik halmaz azokból az állapotokból áll, amelyeket a második állapothalmazba tartozó állapotokból lehet elérni, miközben a második input jelet olvassuk. Mindez formálisan a következő egyenlőségekkel írható le

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

Azt mondhatjuk tehát, hogy az {a,b} halmaz nem más, mint azon lehetséges állapotok halmaza - LAH({a},y) -, amely azokból az állapotokból áll, amelyeket az automata választhat, miközben az *a* állapotban van és az *y* bemenő jelet olvassa. Hasonlóan mondhatjuk, hogy az {a,b,f} szintén lehetséges állapotok egy halmaza, melynek állapotait az automata választhatja, miközben az {a,b}halmazba tartozó valamelyik állapotból indul ki és a második jelet olvassa. Ez esetben is jogos az LAH({a,b},y) jelölés.

Vegyük észre, hogy az automata működésének egy új reprezentációját találtunk meg, mellyel a nemdeterminisztikus automata működése az alábbi módon írható le. Az automata működését a kezdőállapothalmaz valamelyik elemében kezdi, tehát első LAH nem más, mint a kezdő állapotok halmaza: $L_0 = S$. Ezek után az automata olvassa az első bemenő jelet. A lehetséges állapotok következő halmazát L_1 -et úgy kapjuk, hogy vesszük mindazon állapotokat, amelyekbe a L_0 állapotaiból a bemenő szó első jelével címkézett él vezet, vagyis $L_1 := \mathrm{LAH}(L_0, w_1)$. Ezek után az automata olvassa a következő bemenő jelet és megismételhetjük ezt az eljárást: $L_2 := \mathrm{LAH}(L_2, w_2)$. Mindez jól megfigyelhető a következő utasítás outputján.

```
> Delta(Phi,{a},w,trace2);          # 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}
```

$$\Delta(\{a\}, yy) = \{f, a, b\}$$

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

Vegyük észre, hogy determinisztikus esetben az LAH mindig egyelemű halmaz.

Fogalomalkotás

Eddig az új (=nemdeterminisztikus) automata fogalom grafikus reprezentációjával dolgoztunk. Ahhoz, hogy vizsgálataink teljesek legyenek be kell vezetni az új fogalom szimbolikus reprezentációját is.

A kérdés tehát az, hogyan kell az új automata fogalom átmenetfüggvényét definiálni, hogy megmaradjon a determinisztikus modellben jól bevált kapcsolat a szimbolikus reprezentáció s az átmeneti gráf között? Az alábbi kísérlet segít a válaszadásban.

```
> állapot:=a;                  # PoP
        # próbálkozzunk más állapotokkal is (pl. a,b vagy f)
> w:=x;
        # w-nek most csak bemenő jelet adjunk értékül
```

$$w := x$$

```
> Delta(Phi,a,x);
```

A nemdeterminisztikus átmenetek következtében az új átmenetfüggvény egy állapothoz és egy bemenő jelhez nem új állapotot, hanem állapotok egy halmazát rendeli. Ha ez a részhalmaz üres, akkor úgy tekintjük, hogy nincs átmenet definiálva.

Definíció 1.

Automata alatt egy $\Xi = [\ S, X, \delta, Q, F\ ]$ rendezett ötöst értünk, ahol

S véges nemüres halmaz. Az A elemeit **a** Ξ . állapotainak nevezzük.

X véges nemüres halmaz**, melyet ábécének nevezünk. Az X elemei az automata bemenő jelei**.

A δ : SxX-> 2^S függvény, az automata átmenet függvénye.

Q (üres vagy nem üres) részhalmaza A -nak, a kezdőállapotok halmaza.

F (üres vagy nem üres) részhalmaza A -nak, a végállapotok halmaza.

Hangsúlyozzuk, hogy ez az automata fogalom nemdeterminisztikus. Ugyanakkor, ha $|Q| = 1$ minden $s \in S$ állapotra és $x \in X$ bemenő jelre $|\delta(s, x)| = 1$ teljesül, akkor a determinisztikus automata fogalmához jutunk.

Definíció 2.

A Δ : 2^S x X*-> 2^S általánosított átmenetfüggvényt a következő rekurzív definícióval adjuk meg. Az állapothalmaz bármely P részhalmazára és bármely w bemenő szóra valamint xi bemenő jelre

1. $\Delta(P, \varepsilon) = P$
2. $\Delta(P, \xi)$ egyenlő $\delta(s, \xi)$ -vel, ahol $s \in P$.
3. $\Delta(P, \xi\, w) = \delta(\Delta(P, \xi), w)$

Egyelemű P halmaz esetén ez a fogalom egybeesik a determinisztikus esetben bevezetett fogalommal.

Definíció 3.

Azt mondjuk, hogy az automata elfogadja (felismeri) a w benemő szót, ha $\Delta(Q, w) \cap F \neq \{\ \}$. A Ξ automata által felismert szavak halmazát $L(\Xi)$ -vel jelöljük.

4.3 Problémamegoldás tanítása

Ebben a fejezetben rátérünk az automata elméleti problémamegoldásra módszertanának vizsgálatára a komputer algebrai rendszerek (CAS)

felhasználásával. Az **aktivitási elvre** alapozva, amely azt állítja, hogy a tanulónak aktívan részt kell venni a tanulási folyamat minden egyes fázisában, a **vezetett felfedezéses tanulás** módszerét választjuk. Bár a didaktikai irodalom ezt a módszert 4 fázisra osztja, annak érdekében, hogy hangsúlyozzuk az előkészítő fázis fontosságát, szívesebben beszélünk az alábbi 5 fázisról.

1. Motiváció
2. Tapasztalatgyűjtés
3. Absztrakció, problémamegoldás
4. Az új tartalom beillesztése a létező tudás rendszerébe
5. Kiértékelés, összefoglalás

A **motivációs fázis** alapvető célja a tanulók érdeklődésének a felkeltése. Hogy lehet ezt elérni? Egy jól strukturált tanmenetet feltételezve a tanulók pillanatnyi tudása mindig megfelel azoknak a feladatoknak, amelyeket a hallgatóknak aktuálisan meg kell oldani. Ez azzal az érzéssel járhat a hallgatókban, hogy az ő tudásuk adekvát és abszolút "nincsen szükség soha semmi új tudásra, mivel mi mindig meg tudunk oldani minden olyan problémát, amit meg kell oldanunk".

Ezen az alapon a motivációnak egyik lehetséges eszköze az éppen az, hogy rámutassunk hogy a **tanuló pillanatnyi tudása, problémák csak egy specifikus körére vonatkoztatva lehet adekvát és soha sem abszolút**. A tanárnak fel kell hívni a figyelmet, és példákkal kell illusztrálnia, hogy mindig találunk olyan problémát, amelynek megoldása fontos számunkra, ám amit a jelenlegi tudásunkkal, a jelenlegi eszközeinkkel nem tudunk megoldani. Mindkét feltételt szükségesnek érezzük ahhoz, hogy a diákok kellően motiválttá váljanak.

Mindezek a lépések az időleges belső bizonytalanságot válthatnak ki a tanulókban, amelyek belső feszültséget kelthetnek, de pont ez az, amire szükségünk van. Azok a tanulók, amelyek nem rendelkeznek ezzel az egészséges belső feszültséggel, nagy valószínűséggel érdektelenek a probléma megoldás irányában. Másrészt ez a belső feszültség, amelyet feloldunk a tanulási folyamat végére, adja meg az örömöt és a siker élményt.

Arra a kérdésre, hogy hogyan gyakoroljunk, hogyan gyakorlatozzuk, és hogyan gyűjtsünk tapasztalatokat, válaszunk a számítógép algebrai rendszer (CAS) használata. A számítógép algebrai rendszerek a Maple-hez hasonlóan adekvát eszköznek bizonyultak a modern matematika tanítás tekintetében. Mivel interaktív rendszerek ezért a **számítógép algebrai rendszerek hatékony eszközét adják a kísérletezésnek**.

A didaktikai kihívás valójában az, hogy hogyan szervezzük a Maple munkalapjainkat. Egy lehetséges megoldás a **PoP-EoP blokkok** használata, amelyek összetartozó utasítások egy sorozatát jelölik ki. A PoP pontban kijelölt utasítás általában az adatokat, paramétereket tartalmazza, amelyektől az

elkövetkezendő számítási lépések függnek. Mindez segít a hallgatónak abban, hogy könnyen megváltoztassa az input paramétereket és leellenőrizhesse azok eredményét a megváltozott számítási eljárás EoP-val jelölt outputján.

Ebben a fejezetben az automata elméleti csomagot, mint a matematika tanítás egy problémamegoldó eszközét tekintjük. Mivel a vizsgálatunk tárgya a determinizmus és nem determinizmus közötti különbség figyelmünket a téma ezen aspektusára irányítjuk.

Az egyes alfejezetben két egyszerű feladatot oldunk meg és kimutatjuk, hogy az a mód, ahogy automata elméleti problémákat megoldunk, gyakran sőt, gyakorlatilag **majdnem mindig nem determinisztikus automatához vezet**. A következő alfejezetben bebizonyítjuk, hogy mind a determinisztikus mind a nem determinisztikus automata, amelyet az előző fejezet feladataira hoztunk létre, megoldását adja a kettes számú gyakorlatnak. A két bizonyítás összehasonlítása rávilágít arra a lényegi differenciára és technikai használatra, valamint a kognitív hatékonyság közti különbségére a két automata típusnak.

Feladatok

Ennek a résznek kettős a feladata. Egyrészt meg akarjuk mutatni, hogyan használjuk az automata elméleti csomagot problémák, feladatok megoldására. Ugyanakkor kíváncsiak vagyunk arra is, hogy milyen típusú automaták jönnek létre a megoldási folyamat során. E célból megoldunk két egyszerű, bár reményeink szerint nem triviális feladatot. A feladatokra nem csak egy megoldást adunk annak érdekében, hogy vizsgálhassuk és összehasonlíthassuk a keletkező automaták tulajdonságait.

1. Feladat

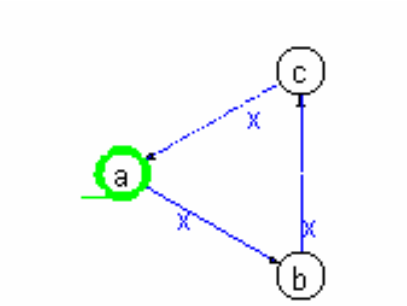
Hozzunk létre olyan automatát, amely felismeri a $\{ x^{(3n)} y \mid 0 \le n \}$ *nyelvet.*

Pólya Györgyöt követve kezdjünk egy sokkal egyszerűbb feladat megoldásával. Hagyjuk el az *y* betűt a szavak végéről és tekintsük a kiindulási nyelv helyett az $\{ x^{(3n)} \mid 0 \le n \}$. Ez a nyelv olyan jellegű szavakból áll, mint pl. az üres szó (ε) vagy az *xxx*, az (*xxx*)*xxx* vagy az (*xxx*)(*xxx*)*xxx*, amelyeket könnyen felismerhetünk egy háromállapotú automatával.

```
> Xi:=genaut([a,x,b,b,x,c,c,x,a],a,a);
```

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

```
> plotaut(Xi);
```

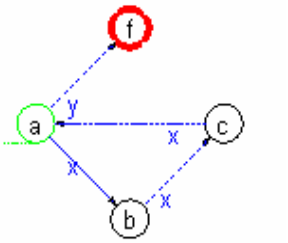


Ezek után megváltoztathatjuk az automatát oly módon, hogy felismerje a y, xxxy, xxxxxxy, xxxxxxxxxy alakú szavakat is. Semmi mást nem kell tennünk ugyanis, mint létrehozni egy új átmenetet az *a* állapotra, és az *y* bemenő jelre. Az új átmenet az eredményének végállapotnak kell lennie, amely ugyanakkor nem lehet egyik létező állapot sem (miért?). Eszerint létre kell hoznunk egy eddig nem létező új *f* állapotot. Mivel az *a* állapot most már nem végállapot, először töröljük őt a végállapotok halmazából, majd ezek után hozzá vesszük az új tranzakciót és az új végállapotot.

```
>
Phi:=addaut(delaut(Xi,final=a),transition=[a,y,f],final=f);
```

$$\Phi := [\{a, f, c, b\}, \{x, y\}, xT, \{a\}, \{f\}]$$

```
> plotaut(Phi);
```

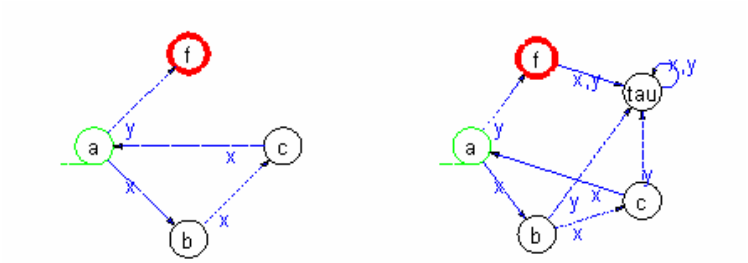


Ez az automata megoldását adja feladatunknak. Tekintsük most a megoldást szolgáltató automata karakterisztikus tulajdonságait. Determinisztikus de nem teljes. Valóban, számos definiálatlan átmenetet találunk. Ugyanakkor ezek az átmenetek irrelevánsak a szavak felismerése szempontjából. Ha ragaszkodnánk ahhoz, hogy a feladat eredményeként teljes automatát kapjunk, akkor a keletkező automatának lényegesen több állapota és számos felesleges átmenete lenne. Mindez természetesen zavarja a láthatóságot, az áttekinthetőséget és a tisztaságot.

```
> Omega:=ConCom(Phi);
```

$$\Omega := [\{\tau, a, f, c, b\}, \{x, y\}, \delta, \{a\}, \{f\}]$$

```
> plots[display]({plotaut(Phi,[-
2,0]),plotaut(Omega,[2,0])});
```



A másik automata több állapottal és számos redundás tranzakcióval rendelkezik, amely nehezebbé teszi működésének megértését. Miközben egy szempillantás alatt meg tudjuk állapítani, hogy az előző automata felismeri az összes $x^{(3n)}y$ alakú szót, addig ugyanez a feladat, bizony percekig is eltarthat a másik automata esetén. Még szembeötlőbbé válik a két automata közötti didaktikai különbség, ha azt kívánjuk hangsúlyozni és igazolni, hogy az automaták nem ismernek fel más szavakat

2. Feladat

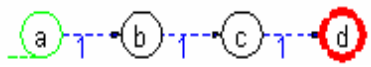
Hozzunk létre olyan automatát a {0,1} ábécé felett, amely akkor és csak akkor ismer fel egy szót, ha azok tartalmaznak három egymás melletti 1-et.

Hasonlóan az előző feladat megoldásához induljunk most is egy egyszerűbb részproblémák megoldásával. A három egymás utáni egyes felismerése könnyű feladat.

```
> Xi:=genaut([a,1,b,b,1,c,c,1,d],a,d);
```

$$\Xi := [\{a, c, d, b\}, \{1\}, \delta, \{a\}, \{d\}]$$

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



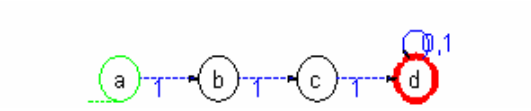
Hogy tudjuk felhasználni ezt az ideiglenes automatát arra, hogy megoldjuk az eredeti feladatot? Azok a szavak, amelyeket fel akarunk ismerni, tartalmaznak 111 alakú részszót. Ezek a szavak tehát nem fejeződnek be a három egyessé, hanem valamilyen v szóval folytatódnak. Más szavakkal tehát az ilyen szavaknak van 111v alakú szuffixe.

Nagyon könnyű megváltoztatni a Ξ automatát úgy, hogy felismerjen $111\,v$ alakú szavakat. Mivel a Ξ automata egyáltalán nem tartalmaz definiált átmenetet a d állapotra, szabadon megválaszthatjuk azt, hogy az új átmenetekre $\delta(d, 0) = \delta(d, 1) = d$ teljesüljön.

```
> Phi:=addaut(Xi,transition={[d,0,d],[d,1,d]});
```

$$\Phi := [\{a, c, d, b\}, \{0, 1\}, xT, \{a\}, \{d\}]$$

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

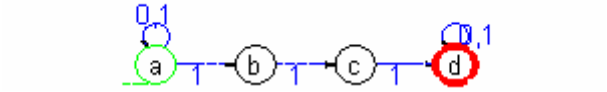


Ám azok a szavak, amelyben vagy 111 alakú résszó, nem csak tetszőleges szuffixet, hanem tetszőleges prefixet is tartalmaznak. Szuffixre van már megoldásunk, és minden okunk meg van, hogy ugyan így járjunk el a perfix esetére is. Vegyünk fel két új átmenetet, ami az a állapotot fixen hagyja.

```
> Phi:=addaut(Phi,transition={[a,0,a],[a,1,a]});
```

$$\Phi := [\{a, c, d, b\}, \{0, 1\}, xT, \{a\}, \{d\}]$$

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

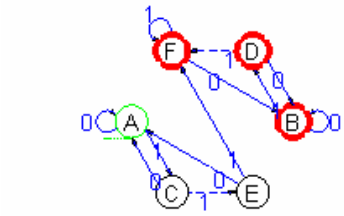


Ez az automata megoldja a kettesszámú problémákat, ugyanakkor se nem teljes, se nem determinisztikus. Ha ragaszkodnánk ahhoz, hogy determinisztikus automatát kapjunk, a következő parancsot kellene kiadnunk.

```
> Omega:=renaut(ConDet(Phi),sta=A);
```

$$\Omega := [\{D, E, F, A, B, C\}, \{0, 1\}, \delta, \{A\}, \{D, F, B\}]$$

```
> plotaut(Omega);
```



A keletkező automata, mint azt korábban is láttuk, több állapottal és több átmenettel rendelkezik. Mivel az összes állapot részt vesz a szavak felismerésében, nem tudunk elhagyni egyet sem belőlük ahhoz hasonlóan, ahogy ezt az előző fejezetben tettük. Ugyanakkor felhívjuk a figyelmet arra, hogy ez nem jelenti, hogy nincs ennél kisebb elemszámú automata, ami e keresett nyelvet felismeri.

Összefoglalás

Két feladatot oldottunk meg ugyanolyan karakterisztikával. Mindkét esetben automatát kellett találnunk, amely felismer egy előre adott nyelvet. Oly módon találtunk meg a megoldást, hogy elindultunk egy automatáról, amely felismert egy az adott nyelvvel szoros kapcsolatban álló nyelvet, és addig változtattuk a keletkező automatát, amíg eljutottunk egy olyan automatához, ami már a

szükséges nyelvet ismeri fel. Ez az eljárás determinisztikus, ám de **nem teljes automatához vezetett** az első esetben, **és nem determinisztikus** és nem teljes automatához a második esetben.

Általában azt állíthatjuk, hogy a legtöbb esetben a természetes emberi gondolkodás nem teljes és nem determinisztikus automatához vezet az automata elméleti feladat megoldások során. Ami a nem teljességet illeti az oka ennek az alapvető emberi lustaság. Igen, valóban nem szeretünk felesleges munkát végezni, és sokkal jobban szeretünk elérni eredményt a lehető legkevesebb erőfeszítéssel. Ugyanakkor az irreleváns állapotok és az irreleváns átmenetek nem segítenek a keletkező automata működésének megértésében, amely azt jelenti, hogy a kognitív hatékonysága a teljes és determinisztikus automatáknak alacsonyabb. Ami a nem determinizmust illeti a kognitív struktúrája a hallgatóknak, és a tanulóknak sokkal közelebb esőnek tűnik a nem determinisztikus modellhez, mind a véges állapotú gépek determinisztikus modelljéhez. Sokkal jobban szeretünk elképzelni egy automatát irányított gráf grafikus reprezentációjában, mint a szimbolikus reprezentációban, és a gráfon egyszerűen irányított utakat keresünk a szavak felismerése során.

Bizonyítások

Ebben a fejezetben megmutatjuk, hogy mind a két automata, amelyet a kettes számú feladat megoldásaként előállítottunk valóban megoldását adja a feladatnak. Célunk, két bizonyítás elvégzése, amely lehetőséget ad azok összehasonlítására és általános didaktikai észrevételek megtételére.

Nemdeterminisztikus eset: a Φ automata

Legyen $w = u111v$, ahol u és v tetszőleges szavak a {0,1} ábécé felett. Az automata egy lehetséges működése, hogy végig az *a* állapotban marad, miközben feldolgozza az *u* bemenő szót, ezek után olvassa a 111-et és átmegy az *f* állapotba, míg ebbe az állapotba maradva feldolgozza a *v* suffixet. Ez azt mutatja, hogy az automata felismeri az ilyen alapú szavakat. Másrészt, ha az input szó nem tartalmaz három egymás utáni egyest, akkor az automata nem képes elérni, az *f* állapotot a szó feldolgozása során. Ez azt jelenti, hogy a Φ az ilyen alakú szavakat nem ismeri fel.

Determinisztikus eset: a Ω automata

Annak megfelelően, hogy a determinisztikus automata bonyolultabb belső struktúrával rendelkezik, a most következő bizonyítás koránt sem olyan egyszerű és könnyen áttekinthető, mint ahogy az a nemdeterminisztikus esetben volt.
Legyen *w* egy input szó, amely $u111v$ alakú. Az általánosság megszorítása nélkül feltételezhetjük, hogy a három egymás utáni 1-es legelső előfordulását jelöltük ki, amiből az következik, hogy u-ban már nincs három egymás utáni 1-es.

Észrevétel

Bármely olyan *u* bemenő szóra, amelyben nincs három egymás utáni 1-es $\Delta(\Omega, A, u) \in \{A, C, E\}$.

Bizonyítás

Az olyan szavakat, amelyekben nincs három egymás utáni 1-s, úgy képzelhetjük el, hogy a bennük lévő nullák sorozatát 1 vagy 11 alakú "1-es blokkok" szakítják meg. Például a 000101100110 szóban három 1-es blokkot találunk, míg a 00 szóban pedig egyet sem.

Észrevételünket az *u* szóban előforduló 1-es blokkok száma szerinti teljes indukcióval bizonyítjuk. Ha u nem tartalmaz 1-es blokkot, akkor valamely $0 \le n$ egész számra $u = 0^n$, és ekkor $\Delta(\Omega, A, u) = A$, mert a 0 bemenő jel az A állapotot fixen hagyja.

Ezután tegyük fel, hogy az *u* szóban van 1-es blokk, továbbá, hogy állításunk már bizonyítottuk minden olyan szóra, melyben nincs három egymás utáni1-es és az 1-es blokkok száma a *u* szóban előforduló 1-es blokkok számánál kisebb. Válasszuk ki az *u*-ban előforduló legelső 1-es blokkot. Ekkor $u = 0^n\,1^k\,0\,0^m\,v$ alakú, ahol $0 \le m, n$ és $k \in \{1, 2\}$. Hogy működik az automata az ilyen alakú szavakra?

$$
\begin{aligned}
\Delta(\Omega, A, u) &= && u = 0^n\,1^k\,0\,0^m\,v \\
&= \Delta(\Omega, A, 0^n\,1^k\,0\,0^m\,v) = && \# \ \Delta \text{ definíciója} \\
&= \Delta(\Omega, \Delta(\Omega, A, 0^n), 1^k\,0\,0^m\,v) = && \# \ \Delta(\Omega, A, 0^n) = A \\
&= \Delta(\Omega, A, 1^k\,0\,0^m\,v) = && \# \ \Delta \text{ definíciója} \\
&= \Delta(\Omega, \Delta(\Omega, A, 1^k\,0), 0^m\,v) = && \# \ \Delta(\Omega, A, 1^k\,0) = A \\
&= \Delta(\Omega, A, 0^m\,v) = && \# \ \Delta \text{ definíciója} \\
&= \Delta(\Omega, \Delta(\Omega, A, 0^m), v) && \# \ \Delta(\Omega, A, 0^m) = A \\
&= \Delta(\Omega, A, v)\,. &&
\end{aligned}
$$

Vegyük észre, hogy ezen a pontos alkalmazhatjuk az indukciós feltevést. Ugyanis a *v* szóban nem lehet három egymás utáni 1-es, ugyanis ha lenne, akkor u-ban is lett volna. Ugyanakkor *v*-ben eggyel kevesebb az 1-es blokkok száma, mint *u*-ban. Eszerint az indukciós feltevés biztosítja, hogy $\Delta(\Omega, A, v) \in \{A, C, E\}$.

Tekintsük ezek után ismét a kiindulási $w = u\,111\,v$ bemenő szót. Emlékeztetünk rá, hogy *w*-ben a három egymás utáni 1-es legelső előfordulását jelöltük ki. Ebből az u szó további tulajdonságai következtethetünk. Például ha u

1-esre végződne, akkor az általunk megjelölt három 1-es nem a legelső előfordulás lenne. AZ u szó tehát nem végződhet 1-re, és így vagy az üres szó vagy pedig 0-re végződik.

Ha $u = \varepsilon$, akkor nyilvánvaló, hogy $\Delta(\Omega, A, u) = A$. Ám ez az egyenlőség a második esetben is érvényben marad. Ugyanis ekkor u=z0 alakú, ahol t-ben nincs három egymás utáni 1-es, így az 1. Észrevétel szerint $\Delta(\Omega, A, z) \in \{A, C, E\}$. Ám a 0 bemenő jel mindhárom állapot az *A* állapotba viszi. Összefoglalva tehát azt tudtuk megmutatni, hogy

$$\Delta(\Omega, A, u) = A .$$

Ezt felhasználva

$$\begin{aligned}
&\Delta(\Omega, A, w) && \# \ w = u111v \\
&= \Delta(\Omega, A, u111v) && \# \ \Delta \text{ definíciója} \\
&= \Delta(\Omega, \Delta(\Omega, A, u), 111\,v) && \# \ \Delta(\Omega, A, u) = A \\
&= \Delta(\Omega, A, 111\,v) && \# \ \Delta \text{ definíciója} \\
&= \Delta(\Omega, \Delta(\Omega, A, 111), v) && \# \ \Delta(\Omega, A, 111) = F \\
&= \Delta(\Omega, F, v)
\end{aligned}$$

Ez alapján annak megmutatásához, hogy $\Delta(\Omega, A, w)$ végállapot, csak annyit kell megjegyezni, hogy a végállapotok {F,B,D} halmaza zárt az átmenetekre, tehát akármilyen is a *v* szó $\Delta(\Omega, F, v)$ (= $\Delta(\Omega, A, w)$) mindig végállapot lesz. Ezzel megmutattuk, hogy Ω a w szót felismeri.

Fordítva, tegyük fel, hogy Ω felismeri a w bemenő szót. Azt kell megmutatnunk, hogy w tartalmaz három egymás melletti 1-t. Mivel $\Delta(\Omega, A, w) \in \{B, D, F\}$, ezért w szükségképpen $w = u1v$, ahol $\Delta(\Omega, A, u) = E$. Ekkor viszont u-nak 11-re kell végződnie. Tegyük fel például, hogy az *u* szó 00-ra végződik. Ekkor alkalmas *z* szóra $u = z00$, és így

```
 Delta(Omega,A,u)=                  # u = z00
=Delta(Omega,A,z00)=                # Delta definíciója
=Delta(Omega,Delta(Omega,A,z),00)=  # az észrevétel szerint
                                    # Δ(Ω,A,z) ∈ {A,C,E},
                                    # ezeket az állaptokat
                                    # 0 az A-ba viszi
=A
```

teljesül, ami lehetetlen. Ugyanígy győződhetünk meg arról, hogy *u* sem 01-re, sem 11 nem végződhet. Az *u* szó tehát $u = z11$ alakú, és innen

$$w = u1v = (z11)\,1\,v = z111v ,$$

ami teljessé teszi annak bizonyítását, hogy *w*-ben találunk három egymás melleti 1 jelet.

Összefoglalás

Ez a két bizonyítás tisztán mutatja a különbséget a determinisztikus és a nem determinisztikus automatákkal való munka között. Nemdeterminisztikus automata esetén bizonyításunk egyszerű volt, világos, rövid és elegáns. **A nem determinisztikus automaták egyszerű belső struktúrája egyszerű bizonyításokhoz vezet.** Mindez egy további érvet szolgáltat a korábbi megjegyzésünkhöz, hogy a nemdeterminisztikus automaták közelebb állnak az emberi gondolkodás kognitív struktúrájához.

Ugyanakkor az informális érvelések sokszor rizikósak, mivel könnyen hibázhatunk, elnézhetünk bizonyos eseteket. Ezért igenis szükségünk van formális, algebrai bizonyításra, amelyeket első sorban a determinisztikus automatákkal való munka során használhatunk. A determinisztikus automaták ugyanis speciális unér algebráknak tekinthetők, és ez megadja a lehetőségét annak, hogy hatékony algebrai eszközökkel vizsgáljuk azokat. Bár ezek az eszközök rendkívül erősen formalizáltak, messze nem szükségtelenek. **Éppen ellenkezőleg a formális bizonyítás segít abban, hogy kialakítsuk az egzakt matematikai gondolkodást, továbbá, hogy tágítsuk a gyakorlatunkat a szimbolikus számítások terén.**

4.4 Konklúzió

Történetileg a természetes útja a véges automata bevezetésének a determinisztikus automata matematikai modelljének megalkotásával kezdődik. A definíció és annak taglalása erősen algebrai beállítottságú az automata fogalom szimbolikus reprezentációján alapul. A tanuló tehát ezek alapján megtanulja, hogy hogyan kezelje a determinisztikus automatát algebrai eszközökkel, hogyan tudja leírni annak működését, hogyan használja az automatákat szavak felismerésére és nyelvek elfogadására. Ezek után a nem determinisztikus automata bevezetése a determinisztikus automata fogalom általánosításaként történik. **Ez a kezelés a szimbolikus reprezentáción épül, amely rendkívül nehézzé teszi a nem determinisztikus automata működésének megértését.** A nem determinisztikus automaták tanítása valóban a komoly didaktikai kihívást jelent minden előadó számára.

Ugyanakkor azt láttuk, **hogyha a grafikus reprezentációt választjuk a szimbolikus helyett, akkor a nem determinisztikus automata könnyűvé és egyszerűen kezelhetővé válik**. Valóban a nem determinisztikus automata az intuitív emberi gondolkodáshoz közelebb állónak bizonyult. Másrészt a nem determinisztikus automata exponenciálisan tömörebb, mint a determinisztikus, amely azt jelenti, hogy exponenciálisan kevesebb állapotot és átmenetet tartalmaz. Ez az egyszerű belső struktúra megengedi, hogy egyszerűbb, és világosabb kezelését adjuk a különböző állítások matematikai bizonyításának.

5 Referenciák

[1] A.V. Aho, J.E. Hopcroft & J.D. Ullman: Data structures and algorithms. - Addison-Wesley, Reading, Mass. 1983
[2] A. Ambrus, Bevezetés a Matematikadidaktikába, Eötvös kiadó,1995
[3] A. Ambrus, Nemzetközi Tendenciák a Matematika Oktatásában, http://xml.inf.elte.hu/~mathdid/tendenc.pdf
[4] D.S.Anachiev, M.V.Volkov, Collapsing words vs. syncronizing words
[5] J. Cerný, A. Piricka & B. Rosenauerová, On directable automata, Kybernetika (Praha) 7 (1971), 289-297
[6] E. Connaly, D. Hughes-Hallett, A.,Gleason, Functions Modeling Change: A Preparation for Calculus
[7] J. Cottrill, Ed. Dubinsky, K. Schwingendorf, K. Thomas, D. Vidakovic, Understanding the Limit Consept: Beginning with a coordinated Process Schema
[8] Z. Fülöp: Formális Nyelvek és szintaktikus elemzésük, Polygon, Szeged 1999.
[9] K.J. Fuchs, Computer Algebra Systems in Mathematic Education, International Symposium - Anniversary of Pollack Mihály College of Engineering, 2002
[10] F. Gécseg & I. Peák: Algebraic theory of automata. - Akadémiai Kiadó, Budapest 1972.
[11] F. Gécseg, B. Imreh, On Completeness of Nondeterministic Automata, Acta Mathematica Hungarica 68 (1995) 151-159
[12] J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison- Wesley, Reading, MA, 1979.
[13] A, Heck, Introduction to Maple, Springer Verlag, New York, Inc.,1993
[14] B. Imreh, M. Steinby, Some Remarks on Directable Automata, Acta Cybernetica 12 (1995), 23-25.
[15] B. Imreh, M. Ito, M. Steinby On commutative directable nondeterministic automata
[16] M. Klincsik, Gy. Maróti, "Maple 8 tételben", Novadat, 1996
[17] D.C. Kozen, Automata and Computability. - Springer-Verlag, New York, 1997.
[18] 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
[19] W. Lindler, Konstructivischer CAS-intensive Mathematickunterricht laengs der APOS-Theorie - die Falltudie Faerben von Graphen
[20] D. Mackie, Using Computer Algebra to Encourage a Deep Learning Approach to Calculus
[21] Gy. Maróti, Didactic Approach for Teaching Nondeterminism in Automata Theory, ZDM, Volume 35 (April), Number 2, 2003
[22] Gy. Maróti, CAS based approach for discussing subset construction, ZDM, Volume 35 (April), Number 2, 2003
[23] Gy. Maróti, Determinism versus Nondeterminism, ZDM, 2003

[24] Gy. Maróti, Illustrated Analysis of Rule of Four using Maple, TMCS, 2004
[25] M.B. Monagan, K.O Geddes, K.M. Heal, G.Labahm, S.M.Vorkoetter, J. McCaron. P.DeMarco: Maple 8 Advanced Programming Guide -Waterloo Maple , 2002
[26] National Council of Teachers of Mathematics (NCTM), Principles and Standards for School Mathematics, 2000
[27] Cs. Sárvári, Network Based Math Teaching Using CAS, International Symposium - Anniversary of Pollack Mihály College of Engineering, 2002
[28] J. Ziegenbalg: Algorithmen Von Hammapuri bis Gödel, Specktrum Akademischer Verlag, 1996

6 Appendix

6.1 A módosított Imreh-Steinby algoritmus forrásszövege

```
> ImrehSteinby:=proc(B::automaton)
local
queue,i,j,xa,xb,xA,xdelta,invT,xC,xs,xn,xp,xd,xdw,xw,xtable,
xinv,xi,\

A,xdirT,xS,xR,xprint,xtrace,xtab,xword,ws1,ws2,xdirect,xshow
,s:
#***************************** sort
ws1:=proc(x,y)
 local xs:
 xs:=lhs(x)[1]<lhs(y)[1];
 xs:=xs or (lhs(x)[1]=lhs(y)[1] and lhs(x)[2]<lhs(y)[2]):
 RETURN(xs)
end;
ws2:=proc(x,y)
 local xs:
 xs:=lhs(x)[1]<lhs(y)[1];
 xs:=xs          or          (lhs(x)[1]=lhs(y)[1]          and
convert(lhs(x)[2],string)<convert(lhs(y)[2],string)):
 RETURN(xs)
end;
#***************************** body
s:=stack[new]();
xtrace:=0:xtab:=false:xshow:=false:xtable:=false:xinv:=false
:xword:=false:
if nargs>1 then
 if member(`trace1`,{args[2..-1]}) or \
  member(1,{args[2..-1]}) then xtrace:=1 fi:
 if member(`trace2`,{args[2..-1]})or \
  member(2,{args[2..-1]}) then xtrace:=2 fi:
 if member(`invtra`,{args[2..-1]}) then xinv:=true fi:
 if member(`dirtab`,{args[2..-1]}) then xtab:=true fi:
 if member(`word`,{args[2..-1]}) then xword:=true fi:
 if member(`show`,{args[2..-1]}) then xshow:=true fi:
 if member(`table`,{args[2..-1]}) then xtable:=true fi:
fi:
#************************************** 1. lépés: inverz
átmeneti tábla konstruálása
A:=B:
invT:=table([]):
for i in indices(A[3]) do
 for j in A[3][i[]] do
 if assigned(invT[j,i[2]]) then
  invT[j,i[2]]:=invT[j,i[2]] union {i[1]}
```

```
 else
  invT[j,i[2]]:={i[1]}
 fi:
 od:
od:
if xinv or xtrace>0 then
 printf("Inverse transition table:\n");
 print(sort(op(op((invT))),ws2));
 printf("\n");
fi:
#*************************************     2.     lépés:
inicializció
if xtrace>0 then
 printf("Initialization:\n");
fi:
xdirT:=table([]):
for xa in indices(invT) do
 xS:=invT[xa[]]:
 xS:=map(x->{x[]},Cartesian(xS,xS)):
 for xb in xS do
 #if xb[1]<xb[2] then
 if nops(xb)=2 then
  #stack[push](xb,s):
  if xtrace>0 then
  printf("..push(%a): %a\n",xb,map(x->x[],[entries(s)]));
  fi:
  if not assigned(xdirT[xb]) then
   stack[push](xb,s):
   xdirT[xb]:={xa[2]}
  else
   xdirT[xb]:=xdirT[xb] union {xa[2]}
  fi:
 fi:
 od:
od:
if xtrace>0 or xtab then
 print(sort(op(op(xdirT)),ws1));
fi:
#************************************  3.  lépés:  a  verem
kiürítése
while not stack[empty](s) do
 xa:=stack[pop](s):
 if xtrace>0 then
 printf("pop()->%a : %a\n",xa,map(x->x[],[entries(s)]));
 fi:
 for xs in A[2] do
 xprint:=false:
 if  assigned(invT[xa[1],xs])  and  assigned(invT[xa[2],xs])
then
```

```
  xS:=map(x-
>{x[]},Cartesian(invT[xa[1],xs],invT[xa[2],xs])):
  #xR:=select(x->not assigned(xdirT[x]),xS):
  if xtrace=2 or (xtrace=1 and xS<>{}) then
  printf(".(%a,%a)   : %a -> %a\n",xa,xs,xS,xS)
  fi:
  for xb in xS do
   if not assigned(xdirT[xb]) then
    stack[push](xb,s):
    if xtrace>0 then
    printf("..push(%a): %a\n",xb,map(x->x[],[entries(s)]));
    fi:
    xdirT[xb]:=map(x->cat(xs,x),xdirT[xa]):xprint:=true:
   else
    xdirT[xb]:=xdirT[xb]             union             map(x-
>cat(xs,x),xdirT[xa]):xprint:=true:
   fi:
  od:
 else
  if xtrace>1 then
  printf(".(%a,%a)   : %a\n",xa,xs,skip)
  fi:
 fi:
 if (xtab or xtrace>0) and xprint then
  print(sort(op(op(xdirT)),ws1));
 fi:
 od:
od:
#************************************  5.  lépés:  output
előállítása
if xshow then
 printf("Result of ImrehSteinby:")
fi:
xn:=nops(A[1]):
xdirect:=evalb(nops(op(op(xdirT)))=xn*(xn-1)/2):
xw:=Fail:
if xdirect then
 xw:=``:xS:=A[1]:
 while nops(xS)>1 do
 xi:=xdirT[{xS[1],xS[2]}][1]:
 xS:=Delta(A,xS,xi,set):
 xw:=cat(xw,xi)
 od:
fi:
if xtable then
 RETURN(eval(xdirT))
elif xword then
 RETURN(xw)
else
```

```
 RETURN(xdirect)
fi:
end:
```

6.2 Peroid eljárás forrásszövege

```
> period:=proc(p,x,k,a)
local p1,p2,p3,p4,p5,p6,p7:
p1:=plot(p,coords=k):
p2:=plots[display](p1,plot([[x,0]],style=point,symbol=circle
,\
          color=black,coords=k),\
          plots[textplot]([x-0.2,-0.2,"x"],coords=k)):
p3:=plots[display](p1,p2,plot([[x,t,t=0..sin(2*x)]],color=bl
ack,coords=k)):
p4:=plots[display](p1,p2,p3,plot([[x+Pi,0]],style=point,\
          symbol=circle,color=black,coords=k),\
          plots[textplot]([x+Pi-0.2,-0.2,"x+Pi"],coords=k)):
p5:=plots[display](p1,p2,p3,p4,plot([[x+Pi,t,t=0..sin(2*(x+P
i))]],\
          color=black,coords=k)):
p6:=plots[display](p1,p2,p3,p4,p5,plot([t,sin(2*x),t=x..x+Pi
],\
          color=blue,coords=k)):
plots[display]([p1,p2,p3,p4,p5,p6],insequence=true);
end:
```

6.3 Konvergencia eljárás forrásszövege

```
> konvergencia:=proc(sorozat,epsilon)
local k,N,a,pa,pp,pm,pt,ps,pr,pd:
N:=40;
a:=expand(limit(sorozat,n=infinity)):
pa:=plot(a,x=0..N,color=blue):
pp:=plot(a+epsilon,x=0..N,color=magenta):
pm:=plot(a-epsilon,x=0..N,color=magenta):
pt:=plots[textplot]([[N+1,a+epsilon,`a+epsilon`],
           [N+1,a,`a`],
           [N+1,a-epsilon,`a-epsilon`]],align=RIGHT):
ps:=NULL:
for k to N do
 ps:=ps,[k,subs(n=k,sorozat)]:

pr:=plot([ps],x=0..N+3,style=point,symbol=circle,color=black
):
 pd||k:=plots[display]({pa,pp,pm,pr,pt}):
od:
plots[display]([pd||(1..N)],insequence=true,
        title=cat(`Sorozat: {`,convert(sorozat,name),`}`));
end:
```

CAS automata elméleti felhasználásának didaktikai kérdései

Értekezés a doktori (Ph.D.) fokozat megszerzése érdekében
a **Matematika** tudományágban

Írta: **Dr. Maróti György** okleveles matematikus

Készült a Debreceni Egyetem **Matematika és számítástudomány** doktori iskolája **Matematika didaktika** programja keretében

Témavezető: **Dr. Csákány Béla**, egyetemi tanár

A doktori szigorlati bizottság:

elnök: **Dr. Lajkó Károly** egyetemi docens, _____

tagok: **Dr. Ambrus András** egyetemi docens, _____

Dr. Fazekas Attila egyetemi adjunktus, _____

A doktori szigorlat időpontja: **2004. január 26**.

Az értekezés bírálói:
Dr. Dömösi Pál, egyetemi tanár
Dr. Imreh Balázs, egyetemi docens

A bírálóbizottság:

elnök: Dr. _____ _____

tagok: Dr. _____ _____

Dr. _____ _____

Dr. _____ _____

Dr. _____ _____

Az értekezés védésének időpontja: **2004** _____