

SEJTPROCESSZOROK PROGRAMOZÁSÁNAK
MATEMATIKAI ALAPJAI

Doktori értekezés

Katona Endre

SZEGED, 1982.

T A R T A L O M

BEVEZETÉS	4
I. SEJTAUTOMATA ÉS SEJTPROCESSZOR	7
1. Klasszikus sejtautomaták	7
2. A sejtprocesszor fogalma és matematikai modellje	11
2.1. Változtatható átmenetfüggvény megvalósítása mikroprogramozással	12
2.2. Inhomogenitás létrehozása belső állapotok segítségével	16
2.3. A sejtprocesszor matematikai modellje	19
II. ALAPVETŐ SEJTALGORITMUSOK	23
1. Bevezetés	23
2. Bináris összeadás és alkalmazásai	27
2.1. Az összeadás sejtalgoritmusai	27
2.2. Összeadás ferde formátumú adatokkal	30
2.3. Bináris számláló	33
2.4. Polinom helyettesítési értékei sorozatának előállítása	35
3. Bináris szorzás és alkalmazásai	40
3.1. A szorzás sejtalgoritmusai	40
3.2. Vektorok belső szorzatának számítása	43



3.3. Mátrixszorzás	43
3.4. Többtényezős szorzás	49
3.5. Hatványozás négyzetreemeléssel	53
3.6. Iteratív osztás	56
4. Adatfeldolgozó jellegű sejtalgoritmusok	64
4.1. Asszociatív tár sejtterben	64
4.2. Statisztika készítés	67
4.3. Szimbólumszótár	69
4.4. Rendezés	72
III. INVERTÁLHATÓ SEJTAUTOMATÁK	79
1. A vizsgálatok célja, az eredmények összefoglalása	79
2. Definíciók	82
2.1. A véges, inhomogén sejtautomata fogalma	82
2.2. Az invertálhatóság fogalma	83
3. Az invertálhatóság szükséges és elegendő feltételei	87
3.1. A globális invertálhatóság szükséges feltétele	87
3.2. A lokális invertálhatóság szükséges-elegendő feltétele	89
4. Az inverzkonstruáló algoritmus	93
5. Lineáris sejtautomaták	95
5.1. Definíciók, alaptulajdonságok	95
5.2. Lineáris sejtautomaták invertálhatósága	98

6. A lineáris inverzkonstruáló algoritmus	103
6.1. Első fázis: invertálható függvénykombinációk generálása	103
6.2. Második fázis: a függvénykombinációk összerakása, illesztése	107
6.3. Az INVGÉN program	112
IV. FÜGGELÉK	117
1. Szimulációs listák a II. fejezethez	117
2. Futási eredmények a III. fejezethez	130
IRODALOMJEGYZÉK	130

BEVEZETÉS

A sejtautomata fogalmának kialakulása nagyjából egybeesik az első digitális számítógépek létrejöttével, mégis mind a mai napig nem készültek sorozatban gyártott sejtautomata-számítógépek. Ennek egyrészt hardware, másrészt software okai vannak.

Ami a hardware kérdését illeti, mindmáig kilátástalannak tűnt olcsón elegendően nagy méretű sejtteret építeni, amely már megfelelő számítási teljesítményt képes nyújtani /FÁY 1973/. Ezen nehézségek megoldására LEGENDI 1977/a mikroprogramozható sejtautomatát, un. sejtprocesszort javasolt, amelynél az egyedi sejtek áramköre jelentősen csökkenthető. Ha ehhez még tekintetbe vesszük a mikroelektro-nika rohamos fejlődését, ma már reálissá vált nagyméretű sejtteret tartalmazó sejtprocesszorok építése.

Software oldalról megoldatlan volt a sejtter hatékony programozása. Bár már NEUMANN 1966 megmutatta, hogy sejtterrel Turing-gép szimulálható, azaz tetszőleges számítási feladat elvégezhető, a sejtterrel végzett számítások nehézkesnek és gazdaságtalannak bizonyultak /CODD 1968/. A hatékony sejtterprogramozáshoz olyan homogén, párhuzamos algoritmusok, un. sejtalgoritmusok szükségesek, ame-

lyeknél a sejttér nagyfokú párhuzamos műveletvégző képessége kihasználható /KATONA 1981/a,b,c/. A sejttérprogramozáshoz segítséget nyújt a sejtautomaták működésének automataelméleti vizsgálata. Ide tartozik a sejtautomaták globális leképezéseivel, vagy a különböző típusú sejtautomatáknak egymással való szimulálásával foglalkozó számos cikk, tanulmány.

A jelen dolgozat a sejtautomaták, sejtprocesszorok fenti software kérdéseivel foglalkozik. Ennek megfelelően egyaránt tartalmaz algoritmusok létrehozására irányuló konstruktív matematikai részeket és mélyebb algebrai, automataelméleti megfontolásokat.

Az I. fejezet megadja a mikroprogramozott sejtprocesszor pontos matematikai modelljét, amely lehetővé teszi a sejtprocesszor automataelméleti vizsgálatát, sőt alapot adhat a hardware tervezésnél felmerülő komplexitási problémák vizsgálatához is.

A II. fejezet fokozott gyakorlati jelentőséggel bír: olyan hatékony sejtalgoritmusokat ismertet, amelyek alapját képezhetik egy sejtprocesszor-alapsoftwarenek, ezáltal - software részről - megnyitja az utat a sejtprocesszorok gyakorlati alkalmazása felé. Hasonló sejtalgoritmusok eddig alig tűntek fel a szakirodalomban /NISHIO 1975/, ill. más célra, pl. VLSI hardware tervezéshez készültek /KUNG 1980/.

A jelen dolgozat csak kiragadott példákat mutat be az eddig készült több mint 100 sejtalgoritmusból, ennél kissé bővebb válogatást KATONA 1981/d tartalmaz.

A III. fejezet egy speciális kérdéskörrel, a sejtautomaták invertálhatóságával foglalkozik. A sejtautomaták globális leképezéseinek tulajdonságait már számos dolgozatban vizsgálták, közöttük a globális invertálhatóság kérdését is. A jelen dolgozatban ezzel szemben - bizonyos gyakorlati alkalmazásoktól indítatván - az ún. lokális invertálhatóságra kerül a hangsúly. A fejezet feltárja a globális és lokális invertálhatóság, mint algebrai tulajdonság alapjait, szükséges-elegendő feltételt ad rá, amelynek alapján algoritmus adható lokálisan invertálható sejtautomaták konstruálására. A kapott eredmények jelentősen egyszerűsödnek és továbbfejleszthetők egy speciális sejtautomataosztály, a lineáris sejtautomaták esetében, ahol lehetővé vált invertálható sejtautomaták számítógépes konstruálása is.

A dolgozatot záró függelék szimulációs példákat tartalmaz a II. fejezet algoritmusaihoz, ill. részleteket mutat be a III. fejezetben tárgyalt inverzkonstruáló program futási eredményeiből.

I. SEJTAUTOMATA ÉS SEJTPROCESSZOR

E fejezetben vázoljuk a sejtautomata-fogalom keletkezését és fejlődésének néhány jellegzetes állomását, amelyek univerzális sejtautomata-számítógép létrehozására irányultak. Ezután a LEGENDI 1977-ben bevezetett sejtprocesszor architektúrát ismertetjük, kimutatva előnyeit a klasszikus sejtautomatákkal szemben. Végül megadjuk ezen sejtprocesszor architektúra teljes és egyszerűsített matematikai modelljét.

1. KLASSZIKUS SEJTAUTOMATÁK

Sejtautomatán a legáltalánosabb értelemben hálózatszerűen összekapcsolt Moore-automatákat értenek. Minden egyes Moore-automata /sejt/ bemenetére néhány másik sejt kimenete /azaz állapotának értéke/ van rákötve, ezeket az adott sejt szomszédainak szokás nevezni. A hálózat időszinkronban működik: minden sejt egyidejűleg vált állapotot az átmenetfüggvényének megfelelően, és minden $t=1,2,3,\dots$ időpillanatban történik egy ilyen globális, az egész sejttérre kiterjedő állapotátmenet.

Egy sejtautomata pontos definiálásához lényegében négy objektumot kell megadni:

(i) a sejtek /véges vagy végtelen/ halmazát;

(ii) a szomszédsági függvényt, amely minden sejthez a szomszédait /mint rendezett n -est/ rendel;

(iii) az egyes sejtek /véges/ állapothalmazát;

(iv) az egyes sejtek átmenetfüggvényeit.

Ha sejttérről beszélünk, akkor általában csak az első két /vagy három/ objektumra gondolunk, vagyis nem törődünk az egyes sejtek átmenetfüggvényeivel.

Neumann János, akitől a sejtautomata gondolata származik, a sejteket négyzetrács szerint kapcsolta össze, a sejttér az egész 2-dimenziós síkra kiterjesztette és minden sejtet azonos átmenetfüggvénnyel látott el. Ez a sejtautomata egy (C, N, A, f) négyessel definiálható, ahol

$C = \{ c_{i,j} \mid i, j \text{ egész szám} \}$ a sejtek halmaza,

$N: c_{i,j} \mapsto (c_{i,j}, c_{i,j-1}, c_{i,j+1}, c_{i+1,j}, c_{i-1,j})$

a szomszédsági függvény,

$A = \{ 0, 1, \dots, s-1 \}$ a sejtállapotok halmaza,

$f: A^5 \rightarrow A$ a közös lokális átmeneti függvény.

Neumann vizsgálatai az önreprodukció modellezésére irányultak, ehhez 29-állapotú sejteket használt egy speciális

f_N átmenetfüggvénnyel. Neumann megmutatta, hogy az általa definiált sejtautomatával Turing-gépet lehet szimulálni, így felvetődött a sejttér számítógépként való felhasználásának lehetősége is. Neumann sejtautomatája azonban erre még nem volt alkalmas.

A gyakorlati alkalmazás felé Codd tette meg a következő

lépést. Neumann 29 állapotú sejtjei helyett 8 állapotúakat használ és olyan f_C átmenetfüggvényt ad meg, amellyel szintén szimulálható az univerzális Turing-gép, ugyanakkor könnyebben alkalmazható egyfajta sejtterprogramozásra.

Codd irányvonalát Magyarországon Fáy Gyula folytatta: az f_C függvényt továbbfejlesztette, és végtelen sejtter helyett un. bábsejtekkel határolt véges sejtteret alkalmazott. Ez a sejtautomata egy $(C \cup C_d, N, A, f_{CI})$ négyessel írható le, ahol

$C = \{c_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m\}$ az aktiv sejtek /valódi sejtek/ halmaza,

$$C_d = \{c_{0,1}, \dots, c_{0,m}, c_{n+1,1}, \dots, c_{n+1,m}, c_{1,0}, \dots, c_{n,0}, c_{1,m+1}, \dots, c_{n,m+1}\}$$

a sejtteret körülvevő bábsejtek halmaza,

$N: c_{i,j} \mapsto (c_{i,j}, c_{i-1,j}, c_{i+1,j}, c_{i,j-1}, c_{i,j+1})$, minden $c_{i,j} \in C$ -re, a Neumann-féle szomszédsági függvény,

$A = \{0, 1, \dots, 7\}$ a sejtállapotok halmaza,

$f_{CI}: A^5 \rightarrow A$ az f_C függvény módosított változata.

A bábsejtek egyrészt az aktiv sejtek homogenitását biztosítják /igy a szélső aktiv sejteknek ugyanannyi szomszédjuk van, mint a belsőknek/, másrészt a sejtautomata input-output funkcióit látják el. A bábsejtek ugyanis nem végeznek állapot-átmenet számítás, hanem állapotukat minden billenési lépésben kívülről változtathatjuk. Amíg tehát a neumanni sejtautomata működését a kezdőállapota egyértelműen meghatározza, addig a bábsejtekkel határolt sejtter működését



menet közben befolyásolni, vezérelni tudjuk.

Konkrét számítási feladatok vizsgálatánál azonban kiderült, hogy aránylag egyszerű feladatok is nagyszámu sejtet és sok billenési lépést igényelnek (FÁY 1978), ugyanakkor csak egyetlen sejt fizikai megépítéséhez is terjedelmes áramkör szükséges, hiszen egy bonyolult szerkezetű, univerzális átmenetfüggvény realizálásáról van szó. Ugy tűnik tehát, hogy a klasszikus sejtautomata-konceptió alapján nem lehet gazdaságos univerzális számítógépet építeni, ehhez fel kell oldani a térben és időben homogén sejtautomata merevségét.

2. A SEJTPROCESSZOR FOGALMA ÉS MATEMATIKAI MODELLJE

A klasszikus sejtautomaták merevségének feloldására LEGENDI 1977a tett javaslatot, az ott bevezetett architektúrát a továbbiakban sejtprocesszornak nevezzük. A sejtprocesszor két lényeges tulajdonsággal rendelkezik, amely megkülönbözteti a klasszikus sejtautomatáktól:

/i/ A sejtek változtatható /cserélhető/ átmenetfüggvény-nyel rendelkeznek. Ennek előnyei kézenfekvők. Amíg egy univerzális átmenetfüggvénnyel pl. egy rendezési feladat csak felettébb körülményesen oldható meg, addig egy kimondott rendező átmenetfüggvénnyel ez a feladat igen hatékonyan elvégezhető. Ugyanigy konstruálhatók összeadó, szorzó, stb. átmenetfüggvények, amelyek mindig csak az adott feladat elvégzésére alkalmasak, azt viszont igen jó hatásfokkal hajtják végre. Ha a sejtter átmenetfüggvénye cserélhető, akkor megtehetjük, hogy a sejteket mindig az adott feladathoz konstruált célfüggvénynyel látjuk el.

/ii/ A sejtterben inhomogenitást lehet létrehozni, vagyis elérhető, hogy az egyes sejtek más-más átmenetfüggvény szerint billenjenek. Az inhomogenitás követelménye természetes módon adódik abból, hogy egy univerzális átmenetfüggvény helyett több feladat-orientált célfüggvényt használunk. Ha pl. egy aritmetikai kifejezést szeretnénk sejtterrel kiértékelni, akkor a sejtterben egyidejűleg kell hogy létezzenek összea-

dást, szorzást ill. osztást végző sejtcsoportok, azaz különböző átmenetfüggvénnyel rendelkező sejtek. Az inhomogenitás lehetősége ezen túlmenően az egyes alapműveletek sejttérbeli realizálását is megkönnyíti.

A következő két pontban kifejtjük, hogy a sejtprocesszor milyen módon teszi lehetővé az átmenetfüggvény változtathatóságát és a térbeli inhomogenitást, majd a 2.3. pontban megadjuk a sejtprocesszor matematikai modelljét.

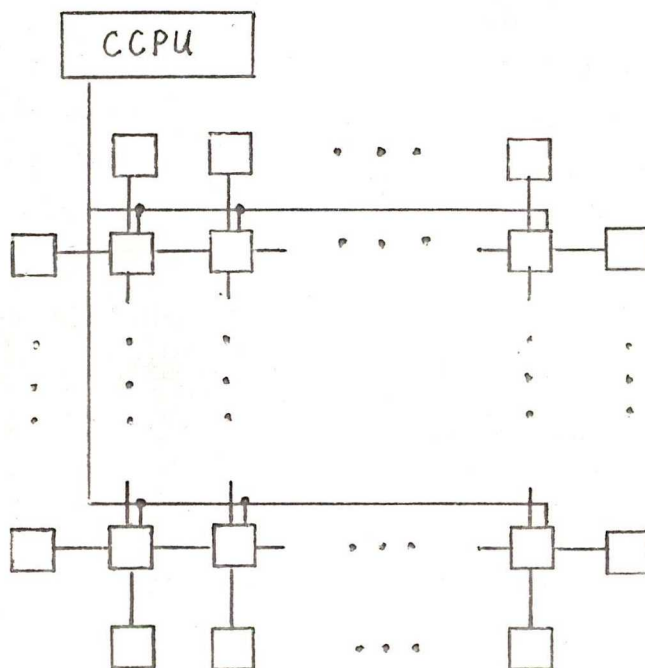
2.1. Változtatható átmenetfüggvény megvalósítása mikroprogramozással

Az átmenetfüggvény cserélhetőségét a sejtek mikroprogramozásával érjük el. A sejtek a korábbi fix átmenetfüggvényt realizáló áramkör helyett csak egy egyszerű parancs-végrehajtó hardware-t tartalmaznak, amely egy verzélő központ /CCPU/ által kiküldött parancsokat /mikroutasításokat/ hajt végre.

A sejtek a környezetük és a központi utasítás együttes hatására váltanak állapotot. A központ minden egyes sejthez ugyanazt az utasítást küldi /1.1. ábra/, de mivel az egyes sejtek környezetei különbözők, így a kiküldött utasítás hatása sejtenként más és más lehet.

A sejthardware-t és a CCPU utasításkészletét úgy kell meghatározni, hogy megfelelő mikroutasítás-sorozattal tetszőleges $f: A^5 \rightarrow A$ lokális állapot-átmenet végrehajtható legyen.

1.1. ábra: A sejtprocesszor
vázlata. /A szélső sejtek
az un. bábsejtek./



Ez nyilván úgy képzelhető el, hogy a sejtek új állapotát egy mikroprogram végrehajtása során fokozatosan számítjuk ki. Mivel ezen számítás alatt a sejtek folyamatosan lekérdezik a szomszédaik állapotait, így a sejtállapotok kettős tárolására van szükség: az A állapothalmaz mellé egy vele izomorf A másodlagos állapothalmazt kell felvenni, és amíg az új állapot kiszámítását a másodlagos sejtállapoton végezzük, addig az elsődleges sejtállapot változatlan marad. Az új állapot kiszámítását megkönnyítheti, ha a sejtben még egy W segédállapothalmaz is rendelkezésre áll.

A sejtek tehát egy $A \times \tilde{A} \times W$ bővített állapothalmazzal rendelkeznek. Az egyes állapotokat a hardware realizációnak megfelelően bináris szavakként definiáljuk, de természetes számnak is tekinthetjük:

$$A = \{ (a^{(1)}, \dots, a^{(s)}) \mid a^{(i)} \in \{0, 1\} \quad 1 \leq i \leq s \} \cong \{0, 1, \dots, 2^s - 1\}$$

az elsődleges sejtállapotok halmaza,

$$\tilde{A} = \{ (\tilde{a}^{(1)}, \dots, \tilde{a}^{(s)}) \mid \tilde{a}^{(i)} \in \{0, 1\} \quad 1 \leq i \leq s \} \cong \{0, \tilde{1}, \dots, \widetilde{2^s - 1}\}$$

a másodlagos sejtállapotok halmaza,

$$W = \{ (w^{(1)}, \dots, w^{(q)}) \mid w^{(i)} \in \{0, 1\} \quad 1 \leq i \leq q \} \cong \{0, 1, \dots, 2^q - 1\}$$

a segéd-sejtállapotok halmaza. Mindehhez a CCPU egy

$$P = \{ p_1, \dots, p_k, p^\# \}$$

parancskészlettel rendelkezik, ahol

$$p_i: A^5 \times \tilde{A} \times W \rightarrow \tilde{A} \times W \quad (1 \leq i \leq k)$$

az átmenetfüggvény-számító mikroutasítások, amelyek hatására megváltoznak a sejtek másodlagos és segéd-állapotai a szomszédok elsődleges állapotaitól és a teljes sajátállapottól függően;

$$p^\#: \tilde{A} \rightarrow A$$

pedig egy speciális átíró utasítás, amely A és $\tilde{A}$ között az izomorf leképezést valósítja meg: $p(\tilde{a}) = a$ minden $a \in A$ -ra. Erre

a lépésre akkor kerül sor, amikor a másodlagos sejtállapotokon már kialakult az $f: A^5 \rightarrow A$ átmenetfüggvénynek megfelelő új állapot. A mikroprogram tehát mindig a $p^\#$ paranccsal végződik.

Megjegyezzük, hogy maga a sejthardware egy $h: P \times A^5 \times A \times W \rightarrow A \times A \times W$ függvénnyel írható le, ahol a $p(\in P)$ parancsokat nem operátoroknak, hanem a CCPU-ból érkező bináris szavaknak tekintjük.

A P parancskészletet teljesnek nevezzük, ha tetszőleges $f: A^5 \rightarrow A$ lokális átmenetfüggvény előállítható valamely

$p_{i_1}, \dots, p_{i_n}, p^\#$ parancssorozattal.

Ha a sejtprocesszor parancskészlete eleget tesz a teljességi követelménynek, akkor vele tetszőleges /azonos méretű és állapot-számu/ neumann-i sejtautomata szimulálható. Emellett a sejtprocesszor természetes módon rendelkezik az átmenetfüggvény cserélhetőségének képességével, sőt az egymás utáni lépésekben végrehajtott átmenetfüggvények is különbözők lehetnek. A sejtprocesszor tehát időben változó átmenetfüggvénnyel is dolgozhat.

A parancskészlet meghatározásánál három egymás ellen ható szempontot kell figyelembe venni:

- a parancskészlet teljes legyen,
- a parancsokat végrehajtó sejthardware minél kisebb legyen,
- az átmenetfüggvényeket minél kisebb mikroprogrammal lehessen realizálni, különösen a sejtterprogramozásban gyakran használt függvények mikroprogramjai legyenek rövidek.

Az alábbiakban két konkrét példát mutatunk teljes parancskészletre, amelyeken a fenti szempontok jól tanulmányozhatók.

1. példa: Ebben a változatban segédállapotokra nincs szükség ($W=\emptyset$), így a parancsok $p: A^5 \times \tilde{A} \rightarrow \tilde{A}$ leképezésekre egyszerűsödnek. A parancskészlet az átmenetfüggvényeknek termekkel való leírására épül /CODD, 1968/. Termen egy $(a_1, \dots, a_5, a_6)$ állapotot értünk, amely azt fejezi ki, hogy a definiálni kívánt $f: A^5 \rightarrow A$ átmenetfüggvény esetében $f(a_1, \dots, a_5) = a_6$. Világos, hogy tetszőleges átmenetfüggvény leírható termék halmazával.

A parancskészlet esetében minden $(a_1, \dots, a_5, a_6)$ termnek egy $p_{a_1 \dots a_5 a_6}$ parancs felel meg, amelynek hatása:

$$p_{a_1 \dots a_5 a_6}(x_1, \dots, x_5, \tilde{x}) = \begin{cases} a_6 & \text{ha } x_1=a_1, \dots, x_5=a_5 \\ \tilde{x} & \text{egyébként.} \end{cases}$$

Az így definiált parancskészlet teljessége nyilvánvaló, és a parancsokat végrehajtó sejt hardware is egyszerű. Nagyobb állapotszám esetén azonban az átmenetfüggvényt realizáló mikroprogram igen hosszú, így ez a változat csak kis állapotszám esetén, illetve kevés termékkel leírható, parciálisan definiált átmenetfüggvények esetén gazdaságos.

2. példa: Olyan parancsokat engedünk meg, amelyek hatása egy

$$x := x \lambda y$$

értékadó utasítással egyenértékű, ahol λ tetszőleges kétváltozós logikai művelet, x a másodlagos sajátállapot vagy a se-

gédállapot egy bitje:

$$x \in \{\tilde{a}^{(1)}, \dots, \tilde{a}^{(s)}, w^{(1)}, \dots, w^{(q)}\},$$

y pedig a szomszédok elsődleges állapotai, vagy a sajátállapot tetszőleges bitje:

$$y \in \{a_1^{(1)}, \dots, a_1^{(s)}, \dots, a_4^{(1)}, \dots, a_4^{(s)}, a^{(1)}, \dots, a^{(s)}, \tilde{a}^{(1)}, \dots, \tilde{a}^{(s)}, w^{(1)}, \dots, w^{(q)}\}.$$

Ezen parancskészlet teljessége könnyen adódik, hisz az $f: A^5 \rightarrow A$ átmenetfüggvény s db különböző 5s változós Boole-függvényre bontható, amelyek - például normálformáik alapján - a fenti parancsokkal rendre kiértékelhetők. /Ehhez 1 segédbit használata elegendő./ Az átmenetfüggvényt realizáló mikroprogram hossza itt a függvény komplexitásától függ, így nagy állapotszám mellett is az egyszerű függvények rövid mikroprogrammal kiszámíthatók.

Megjegyezzük, hogy a két bemutatott példát elvi egyszerűségük miatt választottuk, tényleges hardware megvalósításhoz ezeknél lényegesen gazdaságosabb megoldások is adhatók.

2.2. Inhomogenitás létrehozása belső állapotok

segítségével

A sejtprocesszor CCPU-ja minden sejthez ugyanazt a parancsot küldi, így szükségképpen minden sejt ugyanaz az átmenetfüggvény szerint működik. A sejttérbeli inhomogenitás azzal érhető el, hogy a sejtek az eddig bevezetett A , $\tilde{A}$, W állapothalmazok mellett még egy

$$B = \{ (b^{(1)}, \dots, b^{(r)}) \mid b^{(i)} \in \{0, 1\} \quad 1 \leq i \leq r \} \cong \{0, 1, \dots, 2^r - 1\}$$

belső állapot halmazzal is rendelkeznek. A belső állapotok értékeit a sejtprocesszor indításakor állítjuk be, és azok a működés során nem változnak. A CCPU $p_1, \dots, p_k$ parancsait viszont úgy módosítjuk, hogy azok a belső állapotot is figyelembe vegyék:

$$p_i: A^5 \times \tilde{A} \times W \times B \rightarrow \tilde{A} \times W \quad (1 \leq i \leq k)$$

Igy elérhető, hogy a különböző belső állapotú sejtek más-képp reagáljanak ugyanarra a kiküldött parancsra, vagyis különböző átmenetfüggvényt hajtsanak végre.

Belső állapotos sejtek esetén a P parancskészletet akkor nevezzük teljesnek, ha tetszőleges $f: A^5 \times B \rightarrow A$ függvény realizálható valamely $p_{i_1}, \dots, p_{i_n}, p^*$ mikroprogrammal. Világos, hogy egy $f: A^5 \times B \rightarrow A$ függvény a $B = \{0, 1, \dots, 2^r - 1\}$ halmaznak megfelelően az $f_0, f_1, \dots, f_{2^r - 1}$ részfüggvényekre bontható, ahol

$$f_i(a_1, \dots, a_5) := f(a_1, \dots, a_5, i) \quad (0 \leq i \leq 2^r - 1).$$

Másszóval az i belső állapotú sejtek az $f_i: A^5 \rightarrow A$ átmenetfüggvény szerint működnek, vagyis a sejttérben 2^r különböző lokális átmenetfüggvény létezhet egyidejűleg. A parancskészlet teljessége azt jelenti, hogy ezek a függvények egymástól függetlenül tetszőlegesen választhatók.

2.3. A sejtprocesszor matematikai modellje

Az előző két pontban leírtak alapján sejtprocesszornak nevezünk egy $(C \cup C_d, N, \tilde{A}xWxB, P)$ négyest, ahol

$C = \{c_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m\}$ az aktív sejtek halmaza,

$$C_d = \{c_{0,1}, \dots, c_{0,m}, c_{n+1,1}, \dots, c_{n+1,m}, \\ c_{1,0}, \dots, c_{n,0}, c_{1,m+1}, \dots, c_{n,m+1}\}$$

a bábsejtek halmaza,

$$N: C \rightarrow (C \cup C_d)^5$$

$$c_{i,j} \mapsto (c_{i,j}, c_{i-1,j}, c_{i+1,j}, c_{i,j-1}, c_{i,j+1})$$

a szomszédsági függvény,

$A = \{0, 1, \dots, 2^s - 1\}$ az elsődleges sejtállapotok halmaza,

$\tilde{A} = \{\tilde{0}, \tilde{1}, \dots, \tilde{2^s - 1}\}$ a másodlagos sejtállapotok halmaza,

$W = \{0, 1, \dots, 2^q - 1\}$ a segéd-sejtállapotok halmaza,

$B = \{0, 1, \dots, 2^r - 1\}$ a belső sejtállapotok halmaza,

$P = \{p_1, \dots, p_k, p^{\#}\}$ a CCPU parancskészlete, ahol
 $p_i: A \times \tilde{A} \times W \times B \rightarrow \tilde{A} \times W \quad (1 \leq i \leq k)$ az átmenetszámitó parancsok,
 $p^{\#}: \tilde{A} \rightarrow A$
 $\tilde{a} \mapsto a$ az átíró parancs.

A fenti modell viszonylag adekvát képet ad a sejtprocesszor működési elvéről. A vizsgálatok jelentős részénél azonban nincs szükség ilyen részletes modellre. Sejtalgoritmusok készítésénél például megelégedhetünk azzal a feltételezéssel, hogy a P parancskészlet teljes, és így szabadon változhatunk az átmenetfüggvények között, nem kell mikroprogramozással törődnünk.

Ezen elv alapján a sejtprocesszorra az alábbi redukált modell adható. Maga a sejtprocesszor egy $(C \cup C_d, N, A \times B)$ sejttérre egyszerűsödik, amelynek komponenseit fent már definiáltuk. Ezen sejttér működését egy $P = (\alpha, \beta, D, F)$ sejttérprogram határozza meg, ennek komponensei a következők:

$\alpha: C \cup C_d \rightarrow A$ a kezdőkonfiguráció,

$\beta: C \rightarrow B$ a belső állapot mátrix,

$D = (d^{(1)}, \dots, d^{(z)})$ az I/O program, ahol $d^{(t)}: C_d \rightarrow A \cup \omega$

a t időpontban a bábsejtekre adott input ill. output. Ez pontosabban azt jelenti, hogy $d^{(t)}(c_{i,j}) \in A$ esetén a $c_{i,j}$ bábsejt a $d(c_{i,j})$ állapotot veszi fel, $d^{(t)}(c_{i,j}) = \omega$ esetén pedig "output állapot"-ba kerül, azaz a vele szomszédos aktív sejt $t-1$ időpontbeli állapotát veszi át.



$F = (f^{(1)}, \dots, f^{(z)})$ a számítási program, ahol $f^{(t)}: A^5 \times B \rightarrow A$ a t időpontban alkalmazott összevont átmenetfüggvény, amely az $f_i^{(t)}: A^5 \rightarrow A$ ($0 \leq i \leq 2^r - 1$) tetszőlegesen választható lokális átmenetfüggvényekre bomlik.

A számítási program megadása például az 1.2. ábra szerint történhet, ahol a táblázat minden rubrikájába tetszőleges átmenetfüggvény választható. Így egy z lépéses programban elvileg $z \cdot 2^r$ különböző függvény szerepélhet; a gyakorlatban persze a táblázat sok azonos, rendszerint időben ciklikusan ismétlődő függvényt tartalmaz. Az 1.2. ábra jól szemlélteti a sejtprocesszor által biztosított térbeli és időbeli inhomogenitást.

belső állapot idő	0	1	2	...	$2^r - 1$
1					
2					
3					
⋮					
z					

1.2. ábra: a számítási program egy megadási módja.

A sejtalgoritmusok tárgyalásánál mindig a fenti redukált modellt vesszük alapul. A sejtek külső és belső állapotszámát egyaránt 16-ban maximáljuk ($s \leq 4, r \leq 4$). Az algoritmusok időigényét a szükséges számítási program hosszával (z) becsüljük. Itt azonban meg kell jegyezni, hogy a tényleges időigény ettől többé-kevésbé eltérhet, mivel a különböző átmenetfüggvények mikroprogramjainak hosszában néha drasztikus különbségek lépnek fel.

III. ALAPVETŐ SEJTTALGORITMUSOK =====

1. BEVEZETÉS

Sejttalgoritmuson egy számítási feladat elvégzésére kidolgozott olyan párhuzamos eljárást értünk, amely különösen alkalmas sejtprocesszorral való végrehajtásra.

E fejezetben megmutatjuk, hogy a legalapvetőbb számítási feladatokra /összeadás, szorzás, rendezés, stb./ hatékony sejttalgoritmusok adhatók. Ezen elemi algoritmusok összerakásával fokozatosan egyre nagyobb, bonyolultabb feladatok oldhatók meg sejttérben nagyfokú párhuzamossággal, feltéve, hogy a feladatok jellege a párhuzamos végrehajtást egyáltalán lehetővé teszi. Ilyen összetett algoritmusnak tekinthető a 3.3. pontbeli mátrixszorzás is.

A sejttalgoritmus mindig egy adott $(CUC_d, N, A \times B)$ sejtprocesszorra írt (α, β, D, F) sejttérprogram formájában realizálódik. A sejttérprogram megírásakor derül ki, hogy a sejttalgoritmus valóban hatékony-e, nem tartalmaz-e olyan buktatókat, amelyek a tényleges sejttérbeli végrehajtást már gazdaságtalanná teszik.

Éppen ezért az egyes sejttalgoritmusokat mindig konkrét sejttérprogram formájában mutatjuk be. Ehhez olyan

állapotszámu /legfeljebb 16 állapotu/ sejtteret alkalmazunk, amelyben az adott sejtalgorithmus a legtermészetesebb módon realizálható. Ezt megtehetjük, mert egy adott állapotszámu sejtterrel általában könnyen szimulálhatók más állapotszámu sejtterek. /Erre vonatkozó tételeket ad pl. VOLLMAR 1979./

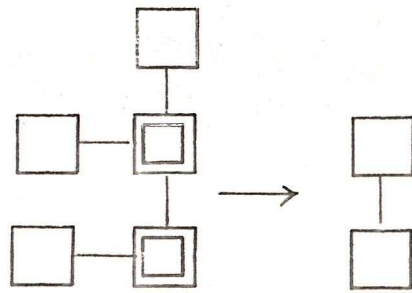
Az (α, β, D, F) sejtterprogram tényleges, formális megadásától általában eltekintünk, mivel az a legtöbb esetben nehezen érthető és kevésbé szemléletes lenne. A legkritikusabb részre, az átmenetfüggvények megadására azonban különös gondot fordítunk.

Átmenetfüggvényen mindig egy $f_i: A^5 \rightarrow A$ függvényt értünk, tehát a sejtprocesszor által végrehajtott $f: A^5 \times B \rightarrow A$ összevont átmenetfüggvényt a belső állapotok szerinti részfüggvényekre bontjuk.

Az átmenetfüggvényeket ún. mikrokonfigurációs módszerrel adjuk meg /LEGENDI 1979/. Mikrokonfiguráción egy kisebb sejtesoport állapotát értjük. Egy mikrokonfigurációs term két mikrokonfigurációból áll: a bal oldali a t időpontbeli állapotot, a jobb oldali a $t+1$ -re megkövetelt állapotot rögzíti.

A legegyszerűbb esetben a jobb oldalon csak egy sejt áll, a bal oldalon pedig ennek 5 szomszédja. A jobb olda-

lon azonban állhat több sejt is; ha ezek belső állapota különböző, akkor több különböző átmenetfüggvényt definiálunk egyidejűleg. A bal oldalon a közömbös szomszédokat egyszerűen elhagyjuk. Ekkor a jobb oldali sejt/ek/ t időpontbeli megfelelőjét a bal oldalon kettős keret jelzi, ha az azonosítás egyébként nem lenne egyértelmű:



A sejtalgorithmusok többségénél az A sejtállapot-halmazt nem egységes egésznek tekintjük, hanem $A = A_1 \times A_2 \times \dots \times A_k$ módon felbontjuk; ennek megfelelően minden $\underline{a}$ sejtállapot az $\underline{a} = (a_1, a_2, \dots, a_k)$ komponensekre bomlik. Az így felbontott sejteket szorzatsejteknek nevezzük. A sejtállapotokat általában bitjeikre bontjuk, vagyis 16-állapotú sejt állapotai 4-bites bináris számoknak tekintendők.

A sejtalgorithmusokban az egyes állapotkomponensek más-más funkciókat tölthetnek be: míg az egyik adatok tá-

rolását végzi, addig a másik vezérlőjeleket továbbit vagy összeadásnál átvitelt képez, stb. Általában mindegyik állapotkomponensre bizonyos adatáramlási irány jellemző /pl. az első állapotkomponenseken jobbra, a második komponenseken lefelé mozognak adatok, stb./. Ez indokolja, hogy a sejttérben a sejtek ugyanazon állapotkomponenseinek együttesét bitsatornának nevezzük /második bitsatornán pl. a második sejtkomponensek összességét értjük/.

A szorzatsejt-elv lehetővé teszi különböző irányu adatfolyamok keresztezését és folyamatos összehasonlítását ill. feldolgozását, hisz a különböző adatfolyamok különböző bitsatornákon "egymáson keresztül" vezethetők. A cross-over /adatfolyam-keresztezés/ problémája, amely a Codd-féle sejtautomatánál oly sok gondot okoz, /FÁY 1974/ a szorzatsejt-elv segítségével tehát megkerülhető.

Végül felhívjuk a figyelmet a Függelék-ben közölt szimulációs példákra, amelyek a sejtalgoritmusokat még érthetőbbé teszik.

2. BINÁRIS ÖSSZEADÁS ÉS ALKALMAZÁSAI

A bináris összeadás a legalapvetőbb aritmetikai művelet, hiszen erre épül a többi aritmetikai művelet is, és gyakorlatilag egyetlen számítási feladat sem képzelhető el nélküle. Ezért különösen fontos, hogy nagy sebességű/lépésenként egy összeadandót fogadó/ bináris összeadót lehet sejtterben konstruálni, hiszen ez lényegében megszabja az összetett szerkezetek működési sebességét is.

2.1. Az összeadás sejtalgorithmusa

Ha két /előjeltehtelen/ bináris számot összeadunk, akkor - szekvenciális gondolkodásmódunknak megfelelően - jobbról balra, helyértékenként végezzük a műveletet, közben ügyel-

1. lépés:	$\begin{array}{r} 0\ 1\ 0\ 1 \\ 0\ 1\ 1\ 1 \\ \hline 0 \\ \textcircled{1} \end{array}$	$\begin{array}{r} 0\ 1\ 0\ 1 \\ 0\ 1\ 1\ 1 \\ \hline 0\ 0\ 1\ 0 \\ \textcircled{0\ 1\ 0\ 1} \end{array}$
2. lépés:	$\begin{array}{r} 0\ 1\ 0\ 1 \\ 0\ 1\ 1\ 1 \\ \hline 0\ 0 \\ \textcircled{1} \end{array}$	$\begin{array}{r} \dots \\ 1\ 0\ 0\ 0 \\ \hline \textcircled{0\ 0\ 1\ 0} \end{array}$
3. lépés:	$\begin{array}{r} 0\ 1\ 0\ 1 \\ 0\ 1\ 1\ 1 \\ \hline 1\ 0\ 0 \\ \textcircled{1} \end{array}$	$\begin{array}{r} \dots \\ 1\ 1\ 0\ 0 \\ \hline \textcircled{0\ 0\ 0\ 0} \end{array}$
4. lépés:	$\begin{array}{r} 0\ 1\ 0\ 1 \\ 0\ 1\ 1\ 1 \\ \hline 1\ 1\ 0\ 0 \\ \textcircled{0} \end{array}$	

2.1. ábra: Szekvenciális összeadási algoritmus. Az átviteleket bekeretezve jelöltük.

2.2. ábra: Párhuzamos összeadási algoritmus. Az összeadandókat már az első lépésben feldolgozzuk, így azokra tovább nincs szükség.

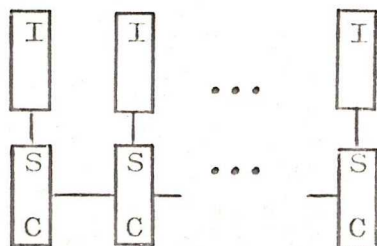
ve a keletkező átvitélekre /2.1. ábra/. Könnyen belátható azonban, hogy az összeadás egyszerre minden helyértéken is végezhető /2.2. ábra/! Ekkor egyszerre több helyen is keletkezik átvitel, melyeket továbbra is párhuzamosan tudunk feldolgozni. Ez a párhuzamos összeadási algoritmus nyilván gyorsabb a szekvenciálisnál, hisz k-bites számok szekvenciális összeadása mindenképp k lépést igényel, míg a parallel algoritmus esetén szerencsés esetben egyetlen lépés is elegendő lehet.

Ezzel a párhuzamos összeadással azonban mégsem nyerünk igazán időt, hiszen a legrosszabb esetben az összeadás most is k lépést igényel. Vegyük azonban észre, hogy az első lépésben a két összeadandó már összeolvad egy félkész összeggé és egy átvitel-vektorra. Semmi akadálya tehát, hogy a második lépésben az átvitélek feldolgozása mellett egy újabb összeadandó hozzáadását is elvégezzük, vagyis minden helyértéken a "félkész összeg jegy" + "új összeadandó jegye" + "az előző lépésben jobbról keletkezett átvitel" összeadást hajtsuk végre /2.3. ábra/.

1. lépés:	$\begin{array}{r} 0\ 1\ 0\ 1 \\ 0\ 1\ 1\ 1 \\ \hline 0\ 0\ 1\ 0 \\ \hline 0\ 1\ 0\ 1 \end{array}$	
2. lépés:	$\begin{array}{r} 0\ 0\ 1\ 0 \\ 1\ 0\ 1\ 0 \\ \hline 0\ 0\ 1\ 0 \end{array}$	← új összeadandó
3. lépés:	$\begin{array}{r} 0\ 0\ 0\ 1 \\ 1\ 1\ 1\ 1 \\ \hline 0\ 0\ 0\ 0 \end{array}$	← új összeadandó

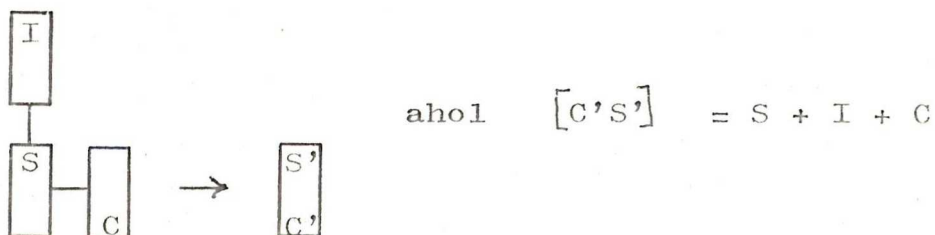
2.3. ábra: A bináris összeadás sejtalgoritmus.

A leírt sejtalgoritmust a legegyszerűbben egy 2-bites /4 állapotú/ sejtekből álló sejtsorral hajthatjuk végre, melyet felülről bábsejtsor határol /2.4. ábra/.



2.4. ábra: Összeadó sejtszerkezet

Az S /sum/ biteken számítjuk az összeget, a C /carry/ bitek az átvitelek képzésére szolgálnak. A kezdőállapotban mindegyik S és C bit 0, a bábsejtek I bitjeire pedig az első összeadandót írjuk. Minden lépésben az I bitekre egy újabb összeadandó kerül, ezáltal "táplálva" az összeadót. Az összeadósejtek a következő átmenetfüggvényt hajtják végre:



/A szögletes zárójellel azt fejezzük ki, hogy C'-t és S'-t együtt, mint kétjegyű bináris számot tekintjük./

A szerkezet működése világos: az első lépésben a bábsejteken lévő első összeadandó tulajdonképpen átíródik az S

bitekre, ehhez a második lépésben hozzáadódik a második összeadandó, stb. Az utolsó összeadandó beérkezése után a bábsejteket nullázzuk, és megvárjuk az eredmény kialakulását /az átvitelek végigfutását/.

A leírt sejtszerkezet esetében n db k -bites szám összeadásának helyigénye k db 2-bites sejt, időigénye pedig $n+k$ lépés.

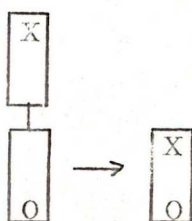
Megjegyzés: Ha a bal szélső összeadó sejtben átvitel keletkezik, ez azt jelenti, hogy az összeadó tulcsordult, a végeredmény tehát hamis lesz. Ezért célszerű a bal szélső összeadósejt átmenetfüggvényét úgy módosítani /inhomogén szerkezet!/, hogy ha a C bit egyszer 1-re vált, akkor mindvégig 1-ben maradjon. Így a végeredmény leolvasásakor észleljük az esetleges menetközben fellépett tulcsordulást is.

2.2. Összeadás ferde formátumu adatokkal

Az összeadás alkalmazásainál gyakran lesz szükségünk olyan összeadóra, amely az $a, b, \dots, z$ összeadandók $a+b+\dots+z$ összegén kívül az a , $a+b$, $a+b+c, \dots$ részletösszegeket is szolgáltatni képes. Az természetesen gazdaságtalan lenne, ha minden egyes összeadás után megvárnánk az átvitelek végigfutását / k lépés/. A következőkben megmutatjuk, hogy az összeadandók formátumának megváltoztatásával a 2.1. pontbeli összeadó sejtszerkezet az egyes részletösszegeket

is szolgáltatni képes, változatlan működési sebesség mellett.

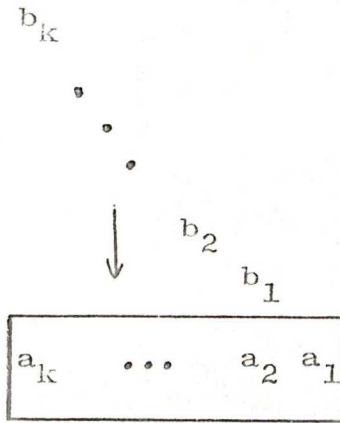
A szemléletesség kedvéért helyezzünk az összeadó sejt-sor alá és fölé "a felső bitsatornán lefelé shiftelő" sejteket, vagyis amelyek a következő átmenetfüggvényt hajtják végre:



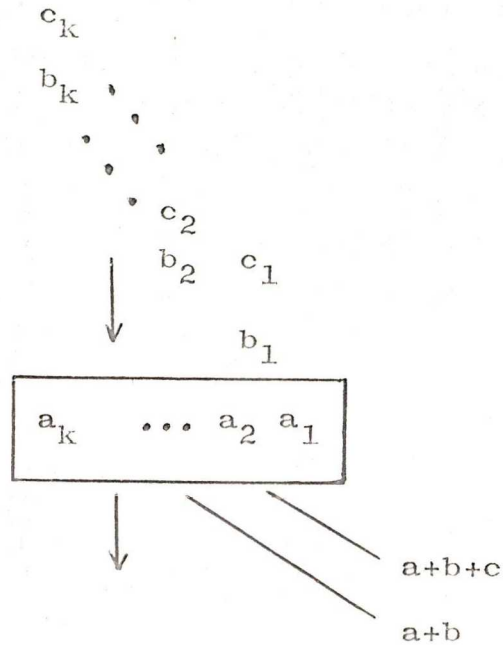
Az $a = [a_k \dots a_1]$ összeadandót az összeadó S bitjein, a $b = [b_k \dots b_1]$ összeadandót pedig a toló sejtek felső bitjein helyezzük el, a 2.5. ábra szerinti ferde formátumban /a többi toló sejtet nullázzuk/.

Ha a sejtteret elindítjuk, akkor a b_i bit az i -edik lépésben kerül az összeadóba. Maga az összeadási folyamat a szekvenciális algoritmus /2.1. ábra/ szerint zajlik le, így az i -edik lépés után az összeadó i -edik S bitje már az $a+b$ összeg kialakult i -edik bitjét fogja tartalmazni.

Ez a helyzet akkor sem változik, ha a 2.6. ábra szerint a b összeadandó után, ugyanolyan ferde formátumban egy $c = [c_k \dots c_1]$ újabb összeadandót küldünk. Az i -edik lépés után az i -edik S -bit most is az $a+b$ összeg i -edik bitjét tartalmazza, a c összeadandó hatása ezt az értéket csak a



2.5. ábra



2.6. ábra

következő lépésben fogja elrontani. Az $a+b$ részletösszeg tehát jobbról balra bitenként, aszinkron módon jelenik meg az összeadóban, és teljes egészében csak az összeadó alá helyezett toló sejtek felső bitjein lesz látható, az inputnak megfelelő ferde formátumban /2.6. ábra/. Könnyű belátni, hogy hasonló módon kapjuk meg az $a+b+c$ részletösszeget is, és több összeadandó esetén minden további részletösszeget.

Végezetül megjegyezzük, hogy a toló sejtek csak az összeadó inputjának és outputjának szemléltetését szolgálták; ferde inputot adni ill. fogadni képes sejtszerkezetet /pl. másik összeadót, lást a 2.4. pontot/ közvetlenül illeszthetünk az összeadóhoz. Az összeadó helyigénye tehát most is k db 2-bites sejt, n szám összeadása pedig ferde formátum

mellett is $n+k$ lépést igényel.

2.3. Bináris számláló

A sejttérbeli folyamatok időzítéséhez, összehangolásához fontos, hogy számláló regisztert tudjunk szimulálni sejttérben. A számlálás, mint művelet az összeadás speciális esetének tekinthető /1 hozzáadás/, így a számláló sejtszerkezet a bináris összeadás sejtalgoritmusára épül.

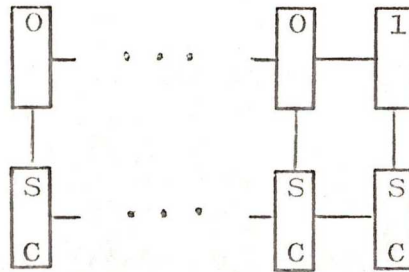
Olyan sejtszerkezetet szeretnénk tehát konstruálni, amely a sejttérben minden időpontban épp a végzett billenési lépések számát mutatja. Ha a bináris számok szokásos megjelenítéséhez ragaszkodunk, akkor nem könnyű feladat előtt állunk, hiszen pl. a $011111 \rightarrow 100000$ átmenetet a sejttér lokális szomszédsági viszonyai mellett nem könnyű megoldani.[‡] Sejtterben azonban jól használhatók a ferde formátumban mozgó adatsorozatok, ilyen outputot szolgáltató számláló pedig az előző pontbeli összeadási elv alapján könnyen készíthető.

Tekintsünk egy bináris összeadó sejtsort, a fölötte lévő sejtsorhoz rendeljünk identikus átmenetfüggvényt, és

[‡]Ilyen k -bites "egyenes számlálót" $c \cdot k$ állapotbit felhasználásával eddig nem sikerült konstruálni. A legjobb /gyakorlatban jól használható/ megoldás esetén $c_1 \cdot k$ külső és $c_2 \cdot k \cdot \log_2(\log_2 k)$ belső állapotbit szükséges.

írjuk ebbe a sorba a 0...01 konstanst /2.7. ábra/. Az összeadó alá "felső bitcsatornán lefelé shiftelő" sejteket helyezhetünk az output megjelenítésére.

2.7. ábra



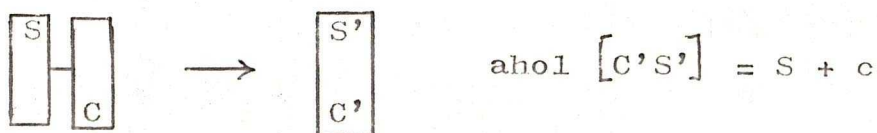
Az összeadó 0...0 állapotból indul, a felette elhelyezett konstans pedig egyenértékű azzal, mintha ferde formátumban érkező 0...01 konstansok sorozatát kapná inputként. Így az összeadó outputján a részletösszegek fognak megjelenni ferde formátumban, a szerkezet tehát a 0, 1, 2, 3,... számok bináris megfelelőit fogja generálni, vagyis bináris számlálóként működik.

Vegyük észre, hogy az összeadó feletti sejtsor elhagyható, ha az összeadósejtek átmenetfüggvényét a kapott inputnak megfelelően módosítjuk.

A jobb szélső számlálósejt átmenetfüggvénye:

$$\begin{array}{|c|} \hline S \\ \hline \end{array} \rightarrow \begin{array}{|c|} \hline S' \\ \hline C' \\ \hline \end{array} \quad \text{ahol} \quad [C'S'] = S + 1$$

A többi számlálósejt átmenetfüggvénye:



Ezek szerint a k -bites bináris számláló helyigénye k db 2-bites sejt, és a szerkezet lépésenként egy újabb bináris számot állít elő ferde formátumban.

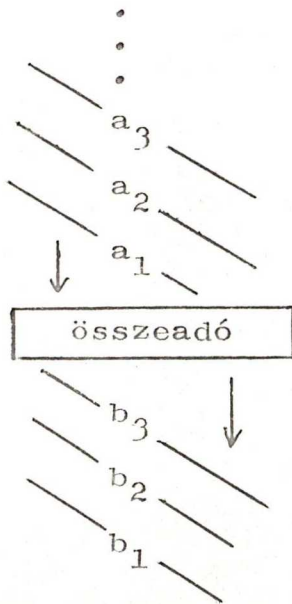
2.4. Polinom helyettesítési értékei sorozatának előállítás

A 2.2. pontban láttuk, hogy az összeadó sejtszerkezet ferde formátumu input adatok esetén részletösszegeket is számol, vagyis egy $a_1, a_2, a_3, \dots$ input sorozat esetén egy $b_1, b_2, b_3, \dots$ output sorozatot állít elő, ahol $b_t = \sum_{i=1}^t a_i$ /2.8. ábra/.

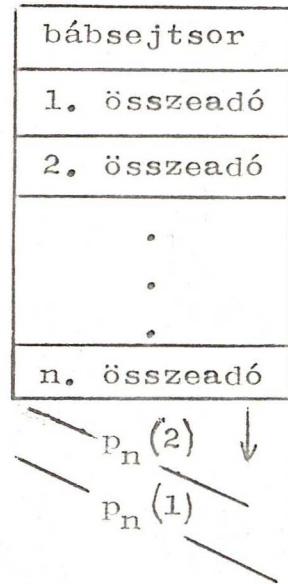
Ezen tulajdonság egyszerű alkalmazása a négyzetszámok sorozatát előállító sejtszerkezet /Katona 1982/1/. Jól ismert ugyanis, hogy a négyzetszámok sorozata előáll a páratlan számok sorozatának összegzésével: $t^2 = \sum_{i=1}^t (2i-1)$, a páratlan számok sorozata pedig könnyen generálható egy összeadóval, amely 1-ről indul és értékét lépésenként 2-vel növeli.

Nem nehéz feladat hasonló módon /3 összeadóval/ a köb-





2.8. ábra



2.9. ábra

számok sorozatának előállítása sem. Felvetődik azonban a kérdés, hogy egy tetszőleges $p_n(t)$ n -edfoku polinom helyettesítési értékeinek $p_n(1)$, $p_n(2)$, $p_n(3)$, ... sorozatát elő lehet-e állítani n db összeadó egymás alá helyezésével és megfelelő kezdőértékek beállításával /2.9. ábra/. A kérdésre igenlő választ ad az alább bizonyított tétel, amelyhez az összeadósor egy egyszerűsített modelljét vezetjük be.

Az összeadókban az átvitelek végigfutásától eltekintünk, az összeadásokat egy lépés alatt elvégzettnek tekintjük. Az összeadások átlapolt vagy nem átlapolt elvégzési módja ugyanis az algebrai vizsgálatok szempontjából közömbös. Így $s_i(t)$ -vel jelölhetjük az i -edik összeadó t időpontbeli tartalmát ($i=1, \dots, n$), $s_0(t)$ -vel a felső bábsejtsor tartalmát,

és az összeadók működését az

$$S(0) = (s_0(0), s_1(0), \dots, s_n(0))$$

kezdővektorral és az

$$s_0(t+1) = s_0(t)$$

$$s_i(t+1) = s_i(t) + s_{i-1}(t) \quad (t=1, \dots, n)$$

rekurzív formulákkal definiálhatjuk.

Tétel: Tetszőleges $p_n(t) = a_n t^n + \dots + a_1 t + a_0$ polinomhoz megadható az összeadóknak olyan $S(0)$ kezdőállapotvektora, hogy minden $t=0, 1, 2, \dots$ értékre $p_n(t) = s_n(t)$ teljesüljön.

Bizonyítás: n szerinti teljes indukciót alkalmazunk.

$n=1$ esetén világos, hogy egy $p_1(t) = a_1 t + a_0$ polinom helyettesítési értékei az $S(0) = (a_1, a_0)$ kezdővektorral generálhatók.

Ha feltesszük, hogy tetszőleges $(n-1)$ -edfoku polinomra az állítás igaz, akkor egy $p_n(t) = a_n t^n + \dots + a_1 t + a_0$ polinomhoz a következőképp adható meg a megfelelő $S(0)$ kezdővektor. Tekintsük a $p_{n-1}(t) = p_n(t+1) - p_n(t)$ $(n-1)$ -edfoku polinomot, amelyhez létezik olyan $(s_0(0), \dots, s_{n-1}(0))$ kezdővektor, hogy $p_{n-1}(t) = s_{n-1}(t)$ minden t -re teljesül. Ekkor az $S(0) = (s_0(0), \dots, s_{n-1}(0), a_0)$ kezdővektor épp a $p_n(t)$ polinomot fogja generálni, mivel $t=0$ esetén $p_n(0) = a_0 = s_n(0)$, és ha t -re $p_n(t) = s_n(t)$ akkor $(t+1)$ -re:

$$p_n(t+1) = p_n(t) + p_{n-1}(t) = s_n(t) + s_{n-1}(t) = s_n(t+1) \quad . \text{ Q.E.D. }$$

A bizonyítás alapján tehát egy $p_n(t)$ polinomhoz az $S(0)$ kezdővektor meghatározása a

$$p_{n-1}(t) = p_n(t+1) - p_n(t)$$

$$p_{n-2}(t) = p_{n-1}(t+1) - p_{n-1}(t)$$

⋮

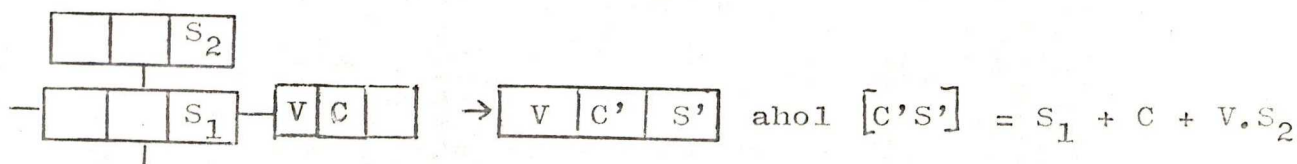
$$p_0(t) = p_1(t+1) - p_1(t)$$

polinomok kiszámításával történik, és $S(0)$ a polinomok konstans tagjaiból áll elő:

$$S(0) = (p_0(0), p_1(0), \dots, p_n(0))$$

Végül még ki kell térnünk a 2. ábra szerinti, n összeadóból álló sejtszerkezet indításának kérdésére. Ha ugyanis a kezdőértékek beállítása után egyszerűen elindítjuk a sejtteret, akkor minden helyértéken minden összeadósejt egyszerre működni kezd, és ezáltal a kezdőértékek összezavarodnak. Ezért az összeadókat fokozatosan, jobbról balra sejtenként kell elindítani, amikor is azok az 1. ábra szerinti ferde adatforgalomnak megfelelően kezdenek dolgozni, és működésük megfelel az algebrai bizonyításnál alkalmazott modellnek.

A fokozatos indításhoz a sejteket egy balra shiftelő vezérlőcsatornával kell bővíteni, és az összeadó átmenetfüggvényt úgy módosítani, hogy csak 1-es vezérlőbit esetén történjen összeadás:



Induláskor minden összeadósejt 0 vezérlőbitet tartalmaz, a jobbszélső bábsejteket viszont 1-es vezérlőbitekkel látjuk el, így a vezérlőjelek balra terjedve elvégzik a fokozatos indítást.

Összefoglalva, egy n -edfoku polinom helyettesítési értékeinek generálásához $n \cdot k$ 3-bites sejt szükséges / k az operandushossz/, és lépésenként generálódik egy újabb helyettesítési érték.

Megjegyzések:

(i) A leírt sejtalgoritmus komplementes kódolású előjeles, számokra és fixpontos bináris törtszámokra egyaránt alkalmazható, így tetszőleges valós együtthatós polinom helyettesítési értékeinek sorozata generálható, az operandushossztól függő pontossággal.

(ii) A sejtalgoritmus érdekes speciális esete az $S(0) = (1, 0, \dots, 0)$ kezdővektor által generált sorozat. Könnyen belátható, hogy ebben az esetben az összeadók a Pascal-háromszög elemeit generálják, pontosabban a k -adik összeadóra $s_k(t) = \binom{t}{k}$ minden $t \geq k$ -ra. ($t < k$ esetén $s_k(t) = 0$.)

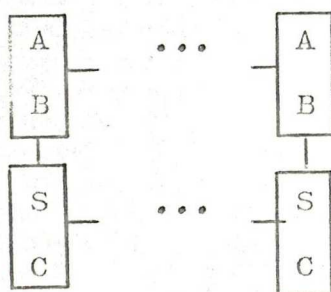
3. BINÁRIS SZORZÁS ÉS ALKALMAZÁSAI

Szorzásra több különböző sejtalgoritmus adható /SMITH 1969, PÁLVÖLGYI 1978, DIÓSLAKI 1979, SZABÓ 1979/, ezek mindegyike a bináris összeadásra épül, csak a részletszorzatok generálásának módjában térnek el egymástól. Itt a legegyszerűbb szorzó sejtszerkezetet ismertetjük, majd ennek alkalmazását mutatjuk be vektor- és mátrixszorzásra. Ezután két módosított szorzó sejtalgoritmust ismertetünk, amelyek az iteratív osztóban kerülnek felhasználásra.

3.1. A szorzás sejtalgoritmus

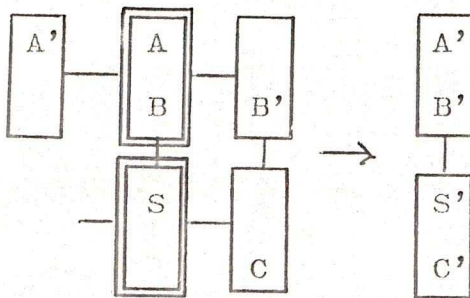
A szorzás műveletét - amint az szokásos - az összeadásra vezetjük vissza, így a szorzó sejtszerkezet is az összeadóra épül. Feladatunk, hogy az összeadó feletti sejt-sorban rendre előállítsuk az egyes részletszorzatokat, amelyek így összeadódnak, és az összeadó S bitjein megkapjuk a szorzatot.

A szorzó két sor kétbites sejtéből áll /2.10. ábra/.



2.10. ábra: Szorzó sejtszerkezet

Az A biteken az $a = [a_k \dots a_1]$ szorzandó áramlik jobbra, a B biteken a $b = [b_k \dots b_1]$ szorzó balra, és az összeadósejtek pedig éppen felettük álló két bit szorzatát tekintik inputnak. Az első és második sejtsor tehát különböző átmenetfüggvényt hajt végre, a két függvény együtt definiálva a következő:



ahol $[C'S'] = S + C + A \cdot B$.

Ahhoz, hogy az egymással szemben haladó szorzandók épp a részletszorzatokat állítsák elő, jegyek közé egy-egy üres helyet /nullát/ kell iktatni. Az ekkor kialakuló működést a 2.11. ábra szemlélteti. Ezen látható, hogy a b_k bit sorra találkozik az $a_1, a_2, \dots, a_k$ bitekkel, ezáltal képezi az $[a_k \dots a_1] \cdot b_k$ részletszorzatot, amely aszinkron módon kerül az összeadóba. Ugyanez történik a b_{k-1} bit esetében is, de már egy helyértékkal jobbra tolódva, az összeadóba tehát helyérték-helyesen kerülnek az egyes részletszorzatok.

Két k jegyű szám szorzata nyilván $2k$ jegyű, így az összeadót is $2k$ bitre kell méreteznünk. A szorzás induló

más mellett/, ennek végére az átvitelek az összeadó közepéig már végigfutottak. A végeredmény kialakulásáig tehát még k lépés kell, így $3k-1 \approx \underline{3k}$ a teljes időigény.

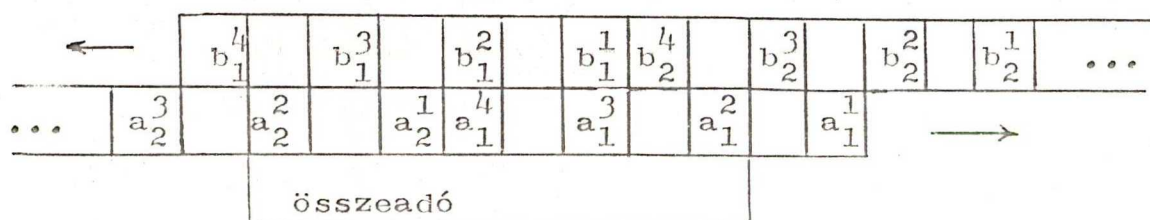
3.2. Vektorok belső szorzatának számítása

Ha a szorzó sejtszerkezet adatcsatornáira nem csak egy-egy operandust, hanem operandusok sorozatát adjuk, akkor a páronként keletkező szorzatok az összeadóban összegződnek. A szorzó sejtszerkezetnek ez a tulajdonsága lehetővé teszi, hogy a szerkezetet lényegében változtatás nélkül vektorszorzásra használjuk. Eljárásunkat az alábbiakban részletezzük.

A feladat tehát az $(a_1, \dots, a_n)$ és a $(b_1, \dots, b_n)$ k -bites számokból álló vektorok $a_1 b_1 + \dots + a_n b_n$ belső szorzatának számítása. Ehhez elegendő az $a_1, \dots, a_n$ számokat rendre a szorzó "A" adatcsatornájára, a $b_1, \dots, b_n$ számokat pedig a "B" adatcsatornájára adni, melynek eredményeképpen $a_i b_i$ -vel szorzódik, és a szorzatok az összeadóban összegződnek. Ügyelni kell azonban arra, hogy az a_i szám a szorzón való áthaladása során a b_{i-1} és b_{i+1} számok egyetlen bitjével se szorzódjon. Ezt a legcélszerűbben úgy biztosíthatjuk, hogy az egymás után küldött számok között nem hagyunk hézagot /2.13. ábra/, így a_i csak b_i -vel találkozik "szinkronban", b_{i-1} -nek ill. b_{i+1} -nek csak a felritkító nulláival kerül fedésbe.

Azonban még ekkor is keletkezhet hibás részletszorzat: amikor az a_i és b_i számok épp egymás fölött vannak, a_{i+1} és b_{i-1} legkisebb helyértékű bitje az összeadó balszélső sejtje felett fedésbe kerül! Emmiatt a balszélső összeadósejt átmenetfüggvényét úgy kell módosítani, hogy az csak átvitelt képezzen, az adatcsatornát hagyja figyelmen kívül:

$$[C'S'] = S+C \quad \text{/passziv összeadó sejt/}.$$



2.13. ábra: 4-bites számokból álló vektorok szorzása.

a_j^i az a_j szám i -edik bitjét jelenti./

Megjegyezzük, hogy amíg két k -bites szám szorzata biztosan elfért a $2k$ bites összeadóban, addig k -bites számokból álló vektorok szorzása esetén már keletkezhet tulcsordulás. Ez elkerülhető, ha a szorzó sejtszerkezetet n elemű vektorok esetén $\log_2 n$ passziv összeadósejttel balról megtoldjuk.

A vektorszorzás időigénye $2kn+k$ lépés, tehát kevesebb, mint n egymás utáni szorzásé, mivel az átvitelek végigfutását csak az utolsó szorzás után kell megvárni.

3.3. Mátrixszorzás

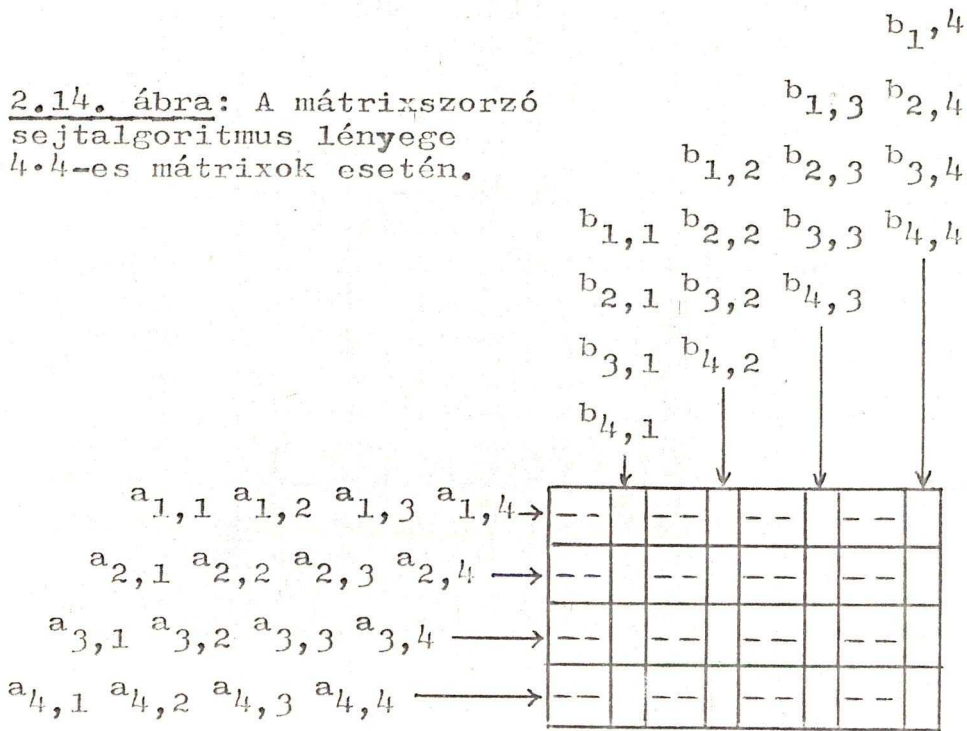
A mátrixszorzást, mint műveletet kézenfekvő vektorszor-
zásra visszavezetni, hisz - A-val ill. B-vel jelölve a szor-
zandó mátrixokat - a $C = A \cdot B$ szorzatmátrix c_{ij} eleme az A
mátrix i-edik sorának és a B mátrix j-edik oszlopának belső
szorzataként áll elő.

Vektorszorzó sejtszerkezet ismeretében tehát a mátrix-
szorzás sejttérbeli megoldása már csak adatforgalmi problé-
mát jelent: $n \cdot n$ -es mátrixok^{*} szorzása esetén elegendő n^2
vektorszorzót mátrixalakban elrendezni, és olyan adatáram-
lást biztosítani, hogy az (i,j) indexű vektorszorzón az A
mátrix i-edik sora és a B mátrix j-edik oszlopa haladjon át;
ekkor az (i,j) indexű vektorszorzó szükségképpen a c_{ij} ele-
met fogja kiszámítani. Erre adunk frappáns megoldást az aláb-
biakban.

A megoldás lényegét a 2.14. ábra mutatja. Az A mátrixot
soronként késleltetve a vektorszorzók "A" adatcsatornáira
adjuk, majd a mátrix egyik szorzóból a másikba áramolva bal-
ról jobbra áthalad a sejttéren. Ezzel egyidejűleg a B mátrixot
oszloponként késleltetve a szorzók között elhelyezett függő-
leges adatcsatornákra küldjük, amelyeken az lefelé áramlik,

^{*}Az ismertetésre kerülő szorzási eljárás téglalap alaku má-
trixok esetében is alkalmazható, pusztán a könnyebb tárgyalás
kedvéért szorítkozunk négyzetes mátrixokra.

2.14. ábra: A mátrixszorzó sejtalgoritmus lényege 4•4-es mátrixok esetén.



2.15. ábra: Egy vektor-szorzó adatforgalma.

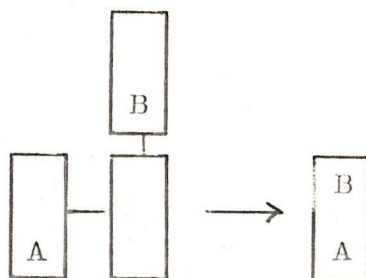


2.16. ábra: A mátrixszorzó sejtter belsőállapot-mátrixa.

1	1	...	1	4	1	1	...	1	4	...
2	3	...	3	5	2	3	...	3	5	
1	1	...	1	4	1	1	...	1	4	...
2	3	...	3	5	2	3	...	3	5	
⋮					⋮					

de minden egyes szorzóhoz érve bekanyarodik /pontosabban elágazik/ annak "B" adatacsatornájára. Így minden egyes vektorszorzóban a 2.15. ábra szerinti adatáramlás megy végbe. A vektorszorzók bal végén a "B" adatacsatorna megszakad, az operandusok nem lépnek át a szomszéd szorzóba.

A sejtszerkezethez szükséges inhomogenitást a 2.16. ábra mutatja. A 2-es belső állapotú sejtek passzív összeadó sejtek /a szorzók bal végén/, a függőleges adatacsatornáknak a 4-es és 5-ös sejtek felelnek meg. A 4-es sejteknél kereszteződik a jobbra shiftelő vízszintes, és a lefelé shiftelő függőleges adatacsatorna, ezért ezek a sejtek "cross-over" /keresztezõ/ átmenetfüggvényt hajtanak végre:



Az 5-ös sejtek a felső bitcsatornán egyszerűen lefelé shiftelnek, míg az alsó biten állandó 0 értéket tartanak /ezt ugyanis a velük balról szomszédos összeadósejt átvitelnek érzékeli/.

A leírt szervezéssel elértük, hogy az (i, j) indexű vektorszorzón az A mátrix i -edik sora és a B mátrix j -edik oszlopa haladjon át. Most már csak azt kell biztosítani, hogy az

elsőnek beérkezett vektorelemek $(a_{in}$ és $b_{nj})$ egyszerre ériék el a szorzót, hisz ekkor már minden további a_{ik} szám szükségképpen a neki megfelelő b_{kj} -vel fog szorozódni.

Ez a sejttérbe érkező mátrixsorok ill. oszlopok megfelelő késleltetésével érhető el /2.14. ábra/. Az A mátrix első sora és B első oszlopa egyidőben kell hogy érkezzon az $(1,1)$ indexű szorzó bemeneteire. Az $a_{1,n}$ elem $2k+1$ lépés után éri el az $(1,2)$ indexű szorzót, a B mátrix második oszlopát tehát $2k+1$ lépéssel kell késleltetni. Ugyanakkor a $b_{n,1}$ elem a függőleges adatcsatornán már 2 lépés alatt eléri a $(2,1)$ indexű szorzó bemenetét, így A második soránál csak 2 lépés késleltetés szükséges. Általában igaz, hogy ha az A mátrix i -edik sorát $(i-1) \cdot 2$ lépéssel, a B mátrix j -edik oszlopát $(j-1) \cdot (2k+1)$ lépéssel késleltetjük, akkor azok az (i,j) indexű vektorszorzóban szinkronban fognak találkozni. Ezzel a mátrixszorzás szinkronizálási problémáját megoldottuk.

Az A mátrix sorai ill. B oszlopai nyilván a vektorszorzásnál alkalmazott formátumban kell hogy érkezzenek a sejttérbe /az egyes számok nulákkal felritkítva, de hózag nélkül egymás után/. A C szorzatmátrix az összeadókban alakul ki a szorzandó mátrixok teljes áthaladása után.

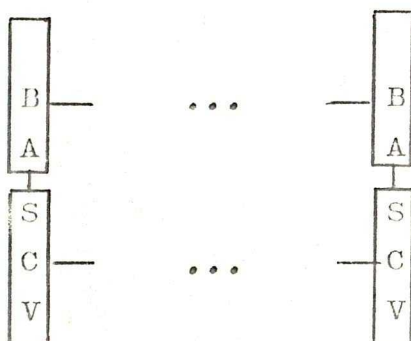
Időigény: Az A és B mátrixok teljes áthaladásának ideje $4kn + 2n - 2$ lépés, a mátrixszorzás időigénye tehát durván $4kn$ lépésnek tekinthető.

Helyigény: $2(2k+1)n^2 \approx 4kn^2$ 2-bites sejt.

3.4. Többszorzós szorzás

A szorzó sejtszerkezetet e pontban úgy módosítjuk, hogy azzal az $a_1, a_2, \dots, a_n$ k-bites fixpontos számok $a_1 \cdot a_2 \cdot \dots \cdot a_n$ szorzatát lehessen számítani. A szorzandók itt "elővesszős" számok, azaz a bal szélső bit helyértéke 2^{-1} . Ez a feltételezés azzal az előnnyel jár, hogy a szorzás során az értéktelen jegyek fognak lecsordulni, túlesordulás pedig egyáltalán nem következhet be.*

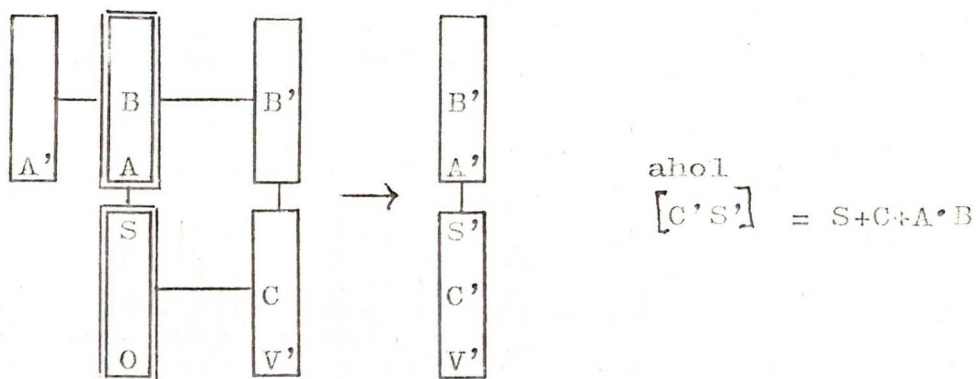
A többszorzós szorzás sejtalgorithmusának lényege, hogy minden egyes szorzás után az összeadóban kialakult szorzatot - egy jobbról balra futó vezérlőjel segítségével - a jobbra shiftelő adatsatornára adjuk, így az összeszorzódik a jobbról érkező következő tényezővel. A szorzó sejtszerkezetet ezért egy vezérlőcsatornával kell kiegészíteni, emiatt 3-bites sejtekre van szükség /2.17. ábra/.



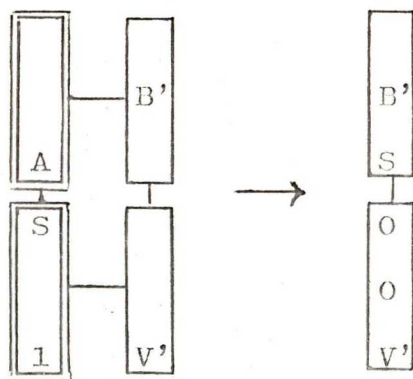
2.17. ábra: Többszorzós szorzó sejtszerkezet.

*Megjegyezzük, hogy "hátralvesszős" /vagyis egész/ számok esetére is adható sejtalgorithmus, ehhez azonban az előző pontbeli től eltérő szorzótípusból kell kiindulnunk /lásd PÁLVÖLGYT 1978/.

A két sor átmenetfüggvényét együtt definiáljuk. 0 vezérlőjel esetén egyszerűen szorzást kell végezni, ez a következő mikrokonfigurációs termmel írható le:



1-es vezérlőjel esetén a /már kialakult/ szorzatot át kell írni a jobbra shiftelő adatsatornára, és egyúttal az összeadót törölni:



A kezdőállapotban az a_1 szorzandót az összeadóban helyezzük el, az $a_2, a_3, \dots, a_n$ szorzandókat pedig a "B" adatsatornára adjuk ritkított formában, közöttük 3 helykitöltő nullát hagyva.

A sejtter elindításakor az összeadó jobb végéről egy 1-es vezérlőjel indul, ennek hatására az a_1 operandusz fokozatosan átiródik az "A" adatcsatornára, és - mintegy mellékhatásként - mullákkal felritkítódik, a szorzó sejtalgoritmus követelményeinek megfelelően. Belátható, hogy az a_1 és a_2 szorzandók az előlvesszős számábrázolásnak megfelelően helyérték-helyesen találkoznak az összeadó felett. /A bal szélső bitek az összeadó balról második sejtje felett találkoznak, mivel $0.1 \times 0.1 = 0.01$./

Egy lépéssel a_1 utolsó jegyének beérkezése után az $a_1 \cdot a_2$ szorzat jobb szélső jegye már kialakul. Már ekkor elindítunk egy újabb vezérlőjelet, amely a szorzatot kialakulásának ütemében az "A" adatcsatornára adja, felülírva az a_1 szorzandó már feleslegessé vált bitjeit. Ezzel idősinkronban érkezik az a_3 szorzandó a "B" adatcsatornára, és máris megkezdődik az $a_1 \cdot a_2 \cdot a_3$ szorzat számítása. A szorzat visszacsatolását tehát menet közben el tudjuk végezni! Ennek köszönhető, hogy az $a_2, \dots, a_k$ szorzandók folyamatosan egymás után érkezhetnek a szorzóba, és a szorzó gyakorlatilag teljes kihasználtsággal működik.

Helyigény: $2k$ helyértékes szorzó esetén $4k$ db 3 -bites sejt, de egy bitcsatorna kihasználatlan, így a szigorú értelemben vett helyigény $10k$ bit.

Időigény: n db k -bites szám összeszorzása $(2k+2)(n-1) + 2k \approx$
 $2kn$ lépés.

Hibaszámitás: Az alábbiakban megvizsgáljuk, hogy $2k$ helyértékes szorzó esetén a fenti sejtalgoritmus milyen pontosságú eredményt ad.

Az $a_1 \cdot a_2$ szorzásnál még nem követünk el hibát, hisz az eredményt $2k$ bitre kapjuk. A következő szorzásnál azonban $2k$ és k bites számokat szorzunk, és akkor a teljes részletszorzat-paralelogramma utolsó k helyértéke nem generálódik. Ennek az elhagyott résznek a maximális értéke $k=4$ esetén:

$$\begin{array}{r} 1 \\ 1 \ 1 \\ 1 \ 1 \ 1 \\ 1 \ 1 \ 1 \ 1 \\ \hline 1 \ 1 \ 0 \ 0 \ 0 \ 1 \end{array}$$

Általában, k bit esetén: $2^k - 2^{k-1} + 2^k - 2^{k-2} + \dots + 2^k - 1 = k \cdot 2^{k-1} - (2^k - 1) = (k-1) 2^{k-1}$ lesz a fenti összeg, amely a szorzat számára $\varepsilon = (k-1) \cdot 2^{-2k}$ hibát jelent.

Mivel a szorzandók 1-nél kisebbek, így a többszörös szorzat hibája a következőképpen becsülhető:

$$(\dots ((a_1 a_2 a_3 - \varepsilon) a_4 - \varepsilon) a_5 - \dots) a_n - \varepsilon > a_1 \cdot \dots \cdot a_n - (n-2) \varepsilon$$

Az n db k bites előlvevesszős szám szorzata tehát maximum $(n-2)(k-1) \cdot 2^{-2k}$ hibával rendelkezik. Ez azt jelenti, hogy ha a szorzat utolsó $h = \log_2 ((n-2)(k-1)) + 2$ bitjét elhagyjuk, akkor $2k-h$ bitre helyesen kerekített eredményt kapunk.

Az eredmény tehát akkor lesz k jegyre pontos, ha $2^{k-2} > (n-2)(k-1)$ fennáll, ez a feltétel pedig a gyakorlatban

fellépő esetekben bőven teljesül.

3.5. Hatványozás négyzetreemeléssel

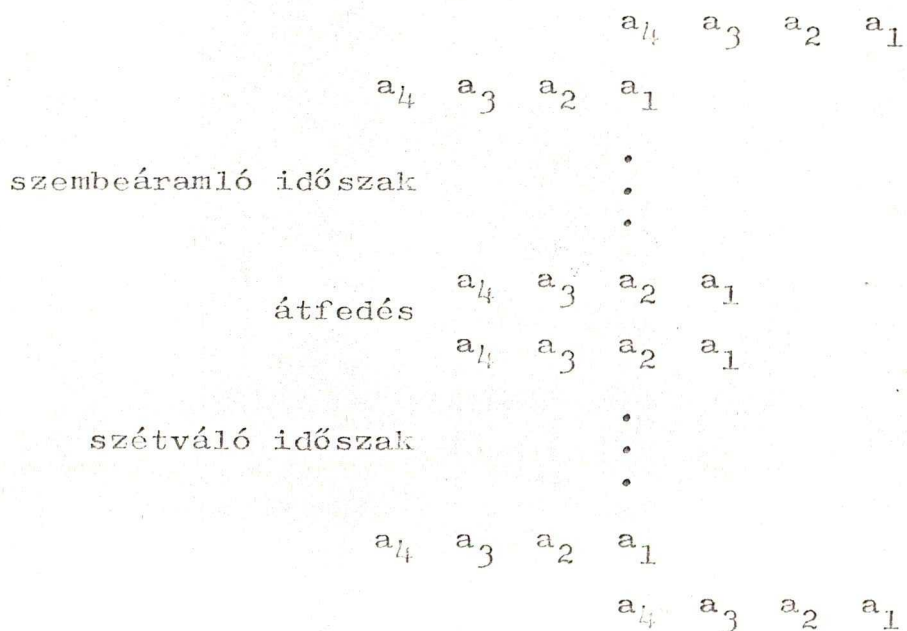
E pontban sejtalgoritmust adunk valamely a fixpontos, előlívesszős szám 2^n -edik hatványának számítására. A hatványozást a leggyorsabb módon, n egymás utáni négyzetreemeléssel végezzük: $a \quad a^2 \quad a^4 \quad \dots \quad a^{2^n}$. A sejt-szerkezetet elsősorban az iteratív osztásnál /3.6. pont/ történő felhasználáshoz készítettük, de természetesen önállóan is alkalmazható.

A feladat megoldásához először négyzetreemelő sejtalgoritmust célszerű készítenünk, ehhez nyilván a szorzó sejtalgoritmust kell alapul venni.

Két szám szorzásának sejtalgoritmus, mint folyamat, két szakaszra bontható: egy szembeáramló és egy szétváló időszakra. Ha a két szorzandó megegyezik /2.18. ábra/, akkor a szembeáramló és szétváló időszakban ugyanazok a részletszorzat-bitek kerülnek az összeadóba. Ezeket a részletszorzatokat tehát elegendő lenne csak egyszer generálni, és 2-vel szorozva /egy helyértékkel balra tolva/ adni az összeadóba.

Erre az egyik lehetséges megoldás, hogy a szorzást a 2.18. ábra "átfedés" állapotából indítjuk, és az "átfedés"-nek megfelelő részletszorzat generálása után mindkét szor-

zandót eggyel balra léptetjük.



2.18. ábra: A szorzó sejtalgoritmus azonos szorzandók esetén

Az alább ismeretett négyzetreemelő sejtszerkezet a fenti elvet a többtényezős szorzás "visszacsatolási" módszerével kombinálja. A sejtszerkezet felépítése megfelel a 2.17. ábrának, és a 2.19. ábra szerinti inhomogenitást /belső-állapot mátrixot/ igényli.

1	1	1	1	1	1	1	1
2	2	2	3	2	2	2	2

2.19. ábra: 8-bites hatványozó belső-állapot mátrixa

A kezdőállapotban a k bites a hatványozandó szám a $2k$ helyértékes összeadóban van. Az összeadó közepéről /a 3-as belső állapotú sejtéből/ indított vezérlőjellel a -t átírjuk az "A" adatsatornára, azzal egyben felritkítva a számot. Miután a teljes szám az "A" adatsatornára kerül, speciális átmenet következik: az a számot az "A" adatsatornán jobbra léptetjük, és egyúttal átírjuk a "B" adatsatornára is. Az összeadót ugyanakkor nullázzuk.

A továbbiakban a szorzás átmenetfüggvényét alkalmazzuk, azzal az eltéréssel, hogy az "A" adatsatornát az első két lépésben megállítjuk. Ezzel elérjük, hogy a 2.18. ábra "átfedés" állapotához tartozó részletszorzat helyértékhelyesen kerüljön az összeadóba, a további részletszorzatok azonban egy helyértékkel balra tolódva fognak keletkezni.

Amint az a^2 szám a közepéig már kialakult az összeadóban, egy újabb vezérlőjellel megkezdhetjük átírását az "A" adatsatornára. A középről indított vezérlőjellel az a^2 szám hátsó felét tulajdonképpen levágjuk, az ebből adódó kerekítési hibát e pont végén vizsgáljuk. A vezérlőjel indításával lényegében előlről kezdődik a négyzetreemelési eljárás, és annyiszor ismételhető, ahány egymás utáni négyzetreemelést akarunk végezni.

Időigény: Egy k bites szám 2^n hatványának számítása $2k$ bites szorzóval $n(2k+2)+k \approx 2kn$ lépés.

Helyigény: $4k$ db 3 bites sejt /ill. $10 k$ bit, a többiténye-



zós szorzóhoz hasonlóan/.

Hibaszámitás: Minden négyzetreemelés előtt a $2k$ bites szám utolsó k bitjét elhagyjuk, az ebből adódó kerekítési hiba $|\varepsilon| \leq 2^{-k-1}$ [‡]. Ennek alapján az a^{2^n} végeredmény hibája a következőképp becsülhető /feltételeztük, hogy mindig felfelé kerekítünk/:

$$\begin{aligned} & (\dots ((a^2 + \varepsilon)^2 + \varepsilon)^2 + \dots)^2 + \varepsilon < \\ & > a^{2^n} + (\dots ((2\varepsilon + \varepsilon) \cdot 2 + \varepsilon) \cdot 2 + \dots) \cdot 2 + \varepsilon = a^{2^n} + (2^n - 1) \cdot \varepsilon < \\ & > a^{2^n} + 2^{n-k-1}. \end{aligned}$$

Ez azt jelenti, hogy $k-n$ jegyre pontosan kapjuk a^{2^n} értékét. Ha tehát k bitre pontos eredményt akarunk, akkor a szorzót $2(k+n)$ bitre kell méretezni.

3.6. Iteratív osztás

Az osztás sejttérbeli megoldásánál a fő nehézséget az időigény jelenti. Ennek oka a hagyományos osztási algoritmus rossz párhuzamosíthatóságában rejlik, amit az alábbiakban mutatunk meg.

A hagyományos /szekvenciális/ szorzási és osztási algoritmusok időigénye között nincs lényeges különbség: k -bites operandusok esetén mindkét algoritmus $O(k^2)$ elemi bit-

[‡]Itt figyelembe vettük, hogy az elhagyáskor a 2^{-k} helyértéken kerekítés végezhető.

szintű műveletet igényel. Míg azonban a szorzás kitűnően párhuzamosítható azzal, hogy egyidejűleg több részletszorzat-bitet generálunk ill. adunk össze $O(k)$ lépés/, addig az osztás szekvencialitása nem oldható fel a következők miatt:

A hagyományos osztási algoritmus két fő művelet ciklikus ismétléséből áll:

/i/ megállapítjuk, hogy az osztandó kijelölt részénél kisebb-e az osztó, azaz "megvan-e benne".

/ii/ ha igen, akkor az osztót levonjuk az osztandó kijelölt részéből és a hányadoshoz 1-es számjegyet írunk; ha nem, akkor a levonás elmarad és a hányadost 0-val töldjük meg.

Világos, hogy a /ii/-beli levonást csak akkor kezdhethetjük el, ha az /i/-beli összehasonlítás eredményét már ismerjük. Mivel az összehasonlítás és a kivonás egyaránt k lépést igényel, így az osztás csak $O(k^2)$ lépésben oldható meg, még ha az egyes résztevékenységeket tudjuk is párhuzamosítani.

Ezekután világos, hogy a hagyományos osztási algoritmust nem célszerű sejtterbe leképezni, ehelyett az osztást visszavezetjük szorzási és összeadási műveletekre. A hányados pontos értékét itt nem számjegyenként, hanem iteratív uton, fokozatos közelítéssel kapjuk meg. Ennél az iteratív osztásnál $O(k \cdot \log_2 k)$ billenési lépés szükséges a k bitre pontos hányados eléréséhez.

Az eljárás alapja az alábbi egyenlőség, amely tetszőleges
 $0 < b < 1$ esetén fennáll:

$$/*/ \quad \frac{a}{b} = a \cdot (1 + (1-b)) \cdot (1 + (1-b)^2) \cdot (1 + (1-b)^4) \cdot \dots$$

illetve pontosabban: $\frac{a}{b} = a \cdot \lim_{m \rightarrow \infty} \prod_{i=0}^m (1 + (1-b)^{2^i})$.

Ennek igazolásához az

$$\frac{1-x^{2^{m+1}}}{1-x} = (1+x)(1+x^2)(1+x^4) \cdot \dots \cdot (1+x^{2^m})$$

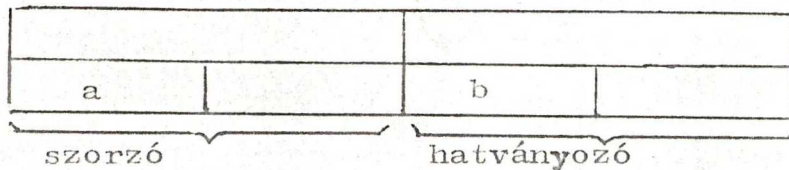
triviális egyenlőségből indulunk ki. Ha $0 < x < 1$, akkor x^n
0-hoz tart, így

$$\lim_{m \rightarrow \infty} \prod_{i=0}^m (1+x^{2^i}) = \frac{1}{1-x},$$

ebből pedig $b = 1-x$ helyettesítéssel könnyen adódik a /*/
összefüggés.

Az a/b hányados előállításához tehát a /*/ egyenlet jobb
oldalán álló kifejezést kell megfelelő pontossággal kiszámi-
tani, vagyis addig kell számolnunk, amíg $(1-b)^{2^i}$ értéke már
elég kicsi lesz.

A sejttérbeli megoldáshoz egy többszorzós szorzó /3.4.
pont/ és egy négyszetreemeléssel hatványozó /3.5. pont/ sejt-
szerkezetet kapcsolunk össze a 2.20. ábra szerint. Az ennek
megfelelő inhomogenitás a 2.21. ábrán látható.



2.20. ábra: Az iteratív osztó szerkezeti vázlata.

1 1 1 1 1 1 1 1	5 3 3 3 3 3 3 3
2 2 2 2 2 2 2 2	4 4 4 6 4 4 4 4

2.2. ábra: 8-bites iteratív osztó belső-állapot mátrixa.

A kezdőállapotban az a osztandót a szorzó összeadójában, a b osztót a hatványozó összeadójában helyezzük el. Mindkét operandusznak k-bites előlvesszős számot feltételezünk, speciális normalizálással: $2^{-2} \leq a < 2^{-1}$ és $2^{-1} \leq b < 1$, vagyis $a = 0.01\dots$, $b = 0.1\dots$ /ezek a feltételezések a tulajdonságok elkerülését, ill. a gyors konvergenciát szolgálják/.

A $/\ast/$ formula számítása a sejtszerkezetben a következőképp történik:

Először a hatványozóban tárolt b osztóból $(1-b)$ -t kell képezni. Ez a művelet tulajdonképpen komplement-képzésnek felel meg, az első lépésben alkalmazott speciális átmenettel érjük el: a hatványozó összeadójának minden sum-bitjét negáljuk, és a 6-os belső állapotú sejtnél még 1-et hozzá is adunk a negált értékhez. Ugyanebből a sejtéből indítjuk

a vezérlőjelet, amely - az esetleges átvitelek végigfutásának ütemében - $1-b$ értékét az "A" adatsatornára adja. A hatványozó ezután már a 3.5. pontban leírt módon működik, melynek eredményeképpen a "B" adatsatornán sorra előállnak az $1-b$, $(1-b)^2$, $(1-b)^4$, ... hatványok, és balra haladva közvetlenül a szorzó szerkezetbe jutnak.

A szorzót a hatványozóból érkező vezérlőjel indítja, amely ekkor elkezdi az $a \cdot (1-b)$ szorzat képzését. Nekiünk azonban az $a \cdot (1+(1-b)) = a + a \cdot (1-b)$ érték kellene; ez egyszerűen úgy számítható, hogy a szorzó szerkezet összeadóját nem töröljük a végigfutó vezérlőjellel. Így minden egyes $(1-b)^{2^i}$, amely a "B" adatsatornán érkezik, $(1+(1-b)^{2^i})$ -vel való szorozást eredményez.

A $/\ast/$ formulából és a $2^{-1} < b < 1$ feltételből könnyen belátható, hogy k bitre pontos eredmény eléréséhez $\log_2 k$ iterációs lépés szükséges. A hányadost a szorzóban, az a szám helyén kapjuk meg; a kiindulási a , b operandusok az osztási eljárás során elvesznek.

Hibaszámitás: A hányados hibája esetünkben két részből áll: iterációs és kerekítési hibából: $H=I+K$. A k bites pontosság tehát például úgy érhető el, ha $I < 2^{-k-2}$ és $K < 2^{-k-2}$. Ezen feltételek teljesülését vizsgáljuk az alábbiakban.

Iterációs hiba: Az $\frac{1-x^{2^{m+1}}}{1-x} = (1+x) \cdot \dots \cdot (1+x^{2^m})$ egyenlőség alapján m iterációs lépés után a hányados hibája $I = \frac{a \cdot (1-b)^{2^{m+1}}}{b}$; ez - a és b értékhatárait figyelembe véve -

$2^{-2^{m+1}}$ -nél kisebb. Így $K < 2^{-k-2}$ fennáll, ha $m+1 > \log(k+2)$, azaz k jegyes pontossághoz gyakorlatilag $m = \log_2 k$ iterációs lépést kell végeznünk.

Kerekítési hiba: Amint a korábbi példáknál láttuk, a kerekítési hiba nagysága a sejtstruktúrák méretétől függ. Tételezzünk most fel $2h$ sum-bites hatványozót és s sum-bites szorzót. Az alább következő hibaszámításból fogjuk megkapni a $K < 2^{-k-2}$ feltételhez szükséges minimális h és s értéket.

A hatványozó kerekítési hibája $\xi = 2^{-h-1}$, így a szorzóba érkező hatványok értékei:

$$x, x^2 \pm \xi, x^4 \pm 3\xi, \dots, x^{2^m} \pm (2^m - 1)\xi.$$

A szorzó kerekítési hibája $\delta = (h-1) \cdot 2^{-s}$, így a hánynados hibákkal terhelt értéke a következő:

$$\left[\dots \left(\left((a(1+x) - \delta)(1+x^2 \pm \xi) - \delta \right) (1+x^4 \pm 3\xi) - \delta \right) \dots \right] \cdot (1+x^{2^m} \pm (2^m - 1)\xi) - \delta$$

Innen kiemelhetjük és összegezzük a teljes kerekítési hibát: /abszolút értékben számítva/:

$$\begin{aligned} K &= \left(\dots \left((\delta \cdot 2 + \xi + \delta) \cdot 2 + 3\xi + \delta \right) \dots \right) \cdot 2 + (2^m - 1)\xi + \delta = \\ &= 2^m \delta + 2^{m-1} \xi + 2^{m-1} \delta + 2^{m-2} 3\xi + 2^{m-2} \delta + \dots + (2^m - 1)\xi + \delta = \\ &= (2^{m+1} - 1)\delta + \sum_{i=1}^m 2^{m-i} (2^i - 1) \xi = \\ &= (2^{m+1} - 1)\delta + ((m-1)2^m + 1)\xi \approx \\ &\approx 2^{m+1} \delta + (m-1)2^m \xi. \end{aligned}$$

A kívánt $K < 2^{-k-2}$ feltétel teljesül, ha $2^{m+1}\delta < 2^{-k-3}$ és $(m-1)2^m\varepsilon < 2^{-k-3}$. Ez utóbbiból, m és ε értékét helyettesítve, kapjuk:

$$(\log_2 k - 1) \cdot 2^{\log_2 k} \cdot 2^{-h-1} < 2^{-k-3}$$

Logaritmizálás után:

$$k + \log_2 k + \log_2 (\log_2 k - 1) + 2 < h$$

Mivel $k > 4$ esetén $\log_2 k > \log_2 (\log_2 k - 1) + 2$, így h minimális értékének $h := k + 2 \cdot \log_2 k$ választható, a továbbiakban ezzel a h -val számolunk.

A $2^{m+1}\delta < 2^{-k-3}$ feltételből, m és δ helyettesítésével, kapjuk:

$$2^{\log_2 k + 1} \cdot (h-1) \cdot 2^{-s} < 2^{-k-3}.$$

Logaritmizálás után:

$$\log_2 k + 1 + \log_2 (h-1) + k + 3 < s$$

Mivel $h < 2k$, így s értékének $s := k + 2 \cdot \log_2 k + 5$ választható.

Összefoglalva: k bitre pontos hányados eléréséhez $k + 2 \cdot \log_2 k + 5$ sum-bites szorzó ill. $2k + 4 \cdot \log_2 k$ sum-bites hatványozó szükséges, és $\log_2 k$ iterációs lépést kell végezni.

Helyigény: k bites operanduszokból k bitre pontos hányados eléréséhez $2 \cdot (3k + 6 \cdot \log_2 k + 5) \approx$ $6(k + 2 \cdot \log_2 k)$ 3-bites

sejt szükséges.

Időigény: A pontos lépésszám $2(k+2 \cdot \log_2 k)(\log_2 k+1) +$
 $+2k+4 \cdot \log_2 k+5$, ez nagyságrendben $O(2k \cdot \log_2 k)$ lépésnek felel
meg.

4. ADATFELDOLGOZÓ JELLEGŰ SEJTAGORITMUSOK

Az aritmetikai alapl műveletek mellett az e pontban tárgyalt adatmanipulációs algoritmusok is fontos részét képezik a sejtprocesszor alapsoftware-ének. A fejezetcím arra utal, hogy ezek az algoritmusok a tényleges vállalati adatfeldolgozásban is alkalmazhatók, amint azt KATONA 1981/a részletesen kifejti.

4.1. Asszociatív tár sejtterben

Összetett sejtszerkezeteknél /pl. a MOLNÁR 1979-ben leírt pontbeli assemblernél/ fontos lehet, hogy valamilyen típusu tárolót tudjunk szimulálni sejtterrel. A hagyományos tártípusok közül az asszociatív tár bizonyult e célra a legalkalmasabbnak, mivel sokoldaluan felhasználható, és a hardware asszociatív táruk alapvetően párhuzamos működésűek.

Tárolók sejtterrel való szimulálásánál a legfőbb gondot a gyors elérés biztosítása jelenti, ugyanis a sejtter lokális szomszédsági viszonyai mellett egy n soros tárból való visszakeresés legalább n lépést igényel. A gyors elérés szó szerinti értelemben tehát megoldhatatlan. Pipe-line szervezéssel azonban elérhető, hogy a tárhoz való fordulások gyors egymásutánban követhessék egymást, bár a kiolvasások /ill. beírások/ csak hosszabb idő elteltével mennek végbe.

Ezt az elvet valósítja meg az asszociatív tár alább ismer-

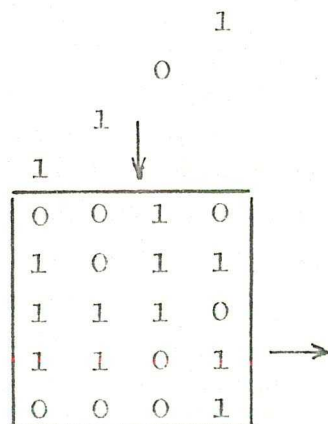
tetett sejtalgoritmus. Az output késése ugyan bizonyos mértékig korlátozza a tár felhasználhatóságát, de a 4.2. és 4.3. pontban bemutatott alkalmazási példák igazolják, hogy az asszociatív tár még így is az egyik legfontosabb sejtszerkezetnek tekinthető.

Egy asszociatív tárnak lényegében a következő feladatokat kell elvégezni:

- n db k -bites szót tárolni;
- ha a bemenetre egy k -bites szó érkezik, ezt mindegyik tárolt szóval össze kell hasonlítani, és megjelölni azt a tárolt szót /szavakat/, amellyel az input szó megegyezik.

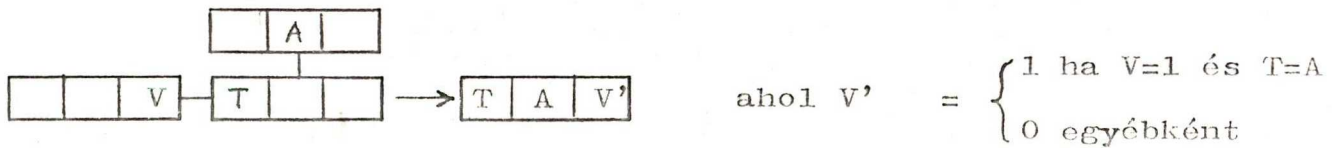
A sejttérbeli megoldáshoz $n \cdot k$ méretű, 3-bites sejtekből álló sejttérrel használunk; az állapotbiteket rendre T, A, V-vel jelöljük. A sejtszerkezet működési vázlata a 2.22. ábrán látható. Az asszociatív tárból lévő n db k -bites szót a T biteken tároljuk /minden sorban egy szó/. Az input szó ferde

2.22. ábra: Az asszociatív tár működési vázlata



formában a felső bábsejtsor A bitjeire érkezik, és az A biteken balról jobbra vezérlőjelek mozognak, melyek az input szónak a tárolt szóval való összehasonlítását végzik. Ennek az összehasonlításnak a mechanizmusa a következő:

A bal bábsejtszlopról 1-es vezérlőjelek indulnak. Az i-edik lépésben az i-edik sor bal széléről induló vezérlőjel jobbra haladva sorra találkozik a sorában tárolt szó és az input szó egymásnak megfelelő bitjeivel. Az 1-es vezérlőjel különböző bitek esetén 0-ra vált, így a sejttér jobb szélén csak akkor jelentkezik 1-es vezérlőjel, ha a két szó megegyezett. Az átmenetfüggvény:

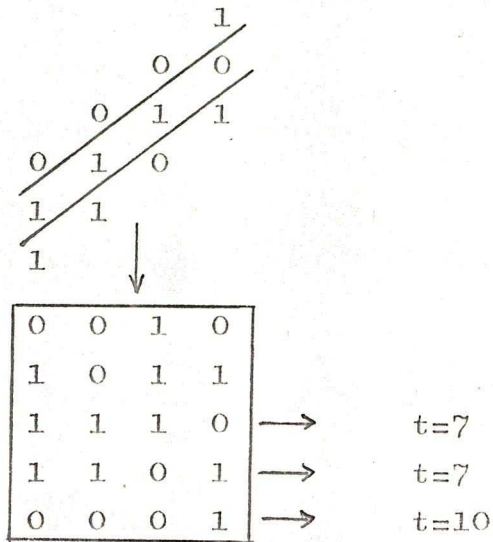


A sejtszerkezet helyes működése csak akkor biztosított, ha az input szó előtt és után 0-kat küldünk a felső bábsejtsorra, és ha a tárban nem szerepel a 0...0 szó.

Belátható, hogy a sejtszerkezet akkor is helyesen működik, ha több input szót küldünk közvetlen egymás után a sejttérbe /2.23. ábra/.

A leírtak alapján a megegyezést jelző vezérlőjelek értelmezése, és így az asszociatív tár használata meglehetősen nehézkesnek tűnhet. Az alábbiakban azonban látni fogjuk, hogy az asszociatív tár számos alkalmazásánál a vezérlőjelek értelmezé-

2.23. ábra: az asszociatív tár működése pipe-line üzemmód esetén. A megegyezést jelző vezérlőjeleknél a keletkezés időpontját is feltüntettük.



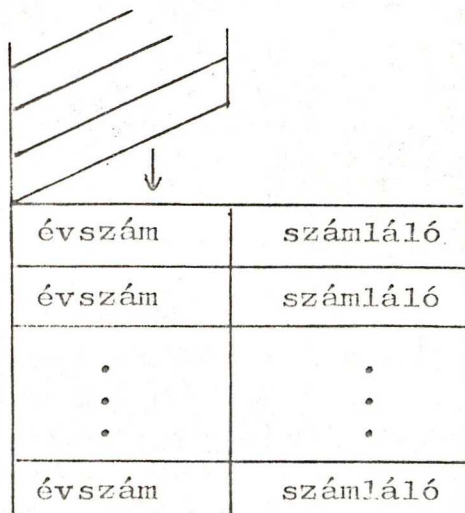
si problémája vagy fel sem lép /statisztika készítés, 4.2. pont/, vagy igen természetes módon oldható meg /szimbólum-szótár, 4.3. pont/.

4.2. Statisztika készítés

Feladat: Egy file minden egyes rekordja egy ember személyi adatait, köztük születési évszámát tartalmazza. Feladtunk a születési évek szerinti statisztika készítése, vagyis meghatározni, hogy mely évben hány ember született.

Megoldás: A szóba jöhető születési évszámokat egy asszociatív tárban helyezzük el, és a tár minden sorához jobbról egy bináris számlálót illesztünk /2.24. ábra/.

A számlálók feladata, hogy az asszociatív tárból jobbra kilépő vezérlőjeleket megszámlálják. Ezért a számlálók megfordítjuk /legkisebb helyérték baloldalt/, és a bal sejteket



2.24. ábra

speciális átmenetfüggvénnyel látjuk el /az asszociatív tár 3-bites sejteket igényel, így a számlálóknál egy bit kihasználatlanul marad/:

$$\boxed{} \boxed{} \boxed{V} - \boxed{} \boxed{} \boxed{S} \rightarrow \boxed{} \boxed{C'} \boxed{S'} \quad \text{ahol} \quad [C'S'] = S + V$$

Az évszám-statisztika elkészítéséhez csak annyit kell tenni, hogy a nyilvántartó file rekordjaiban szereplő évszámokat ferde formában át kell hajtani az asszociatív táron. Ha pl. az 1950 évszám 25-ször fordult elő a file rekordjaiban, akkor áthajtás után az asszociatív tár 1950-et tartalmazó sorához kapcsolódó számláló szükségképpen 25-öt kell hogy tartalmazzon.

4.3. Szimbólumszótár

Feladat: Adott egy

a_1	b_1
a_2	b_2
$\vdots$	$\vdots$
a_n	b_n

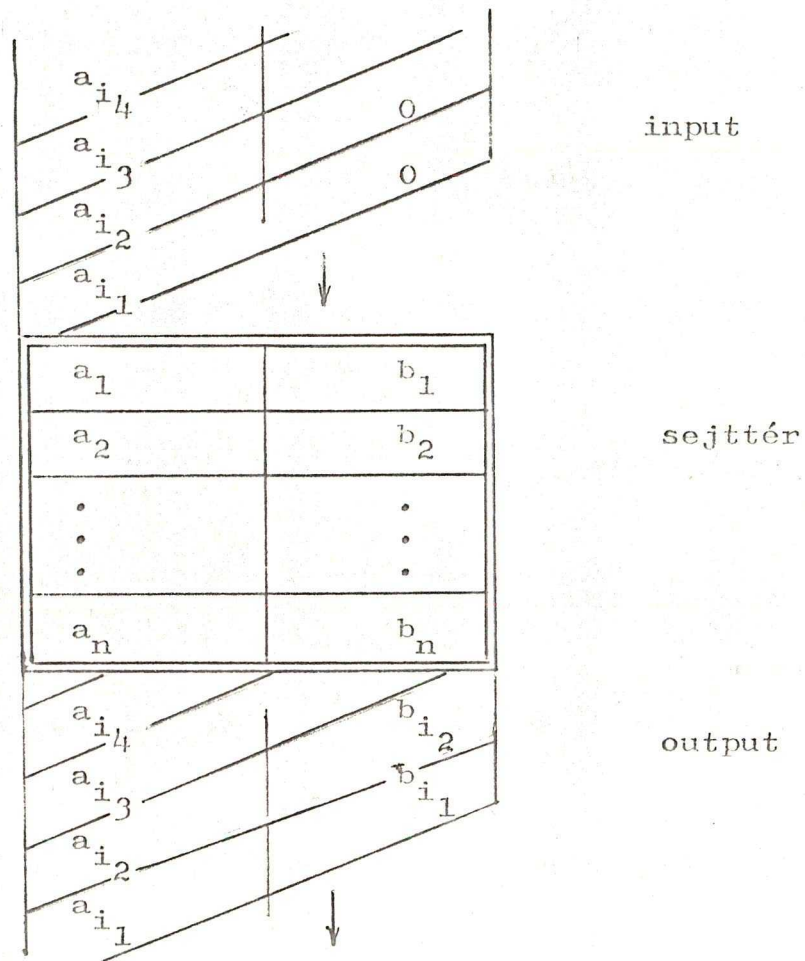
szimbólumszótár, ahol $a_1, \dots, a_n$ k_1 -bites szavak, $b_1, \dots, b_n$ pedig k_2 -bites szavak, és az a_i szónak "fordításkor" a b_i szó felel meg. Feladatunk egy tetszőleges $a_{i_1}, \dots, a_{i_m}$ input /szó-sorozat/ átkódolása a megfelelő $b_{i_1}, \dots, b_{i_m}$ szövegre.

Megoldás: A szótár bal oldalát egy $n \cdot k_1$ méretű asszociatív tárba töltjük. Ehhez jobbról egy $n \cdot k_2$ -es sejttérrészt illesztünk, amelynek T bitjeire a szótár jobb oldalát írjuk /a sejtek állapotbitjeit itt is T , A , V -vel jelöljük/. A szótár tehát természetes módon jelenik meg a sejttér T bitjein /2.25. ábra/.

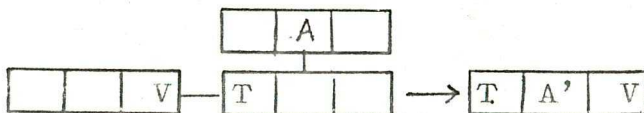
Az $a_{i_1}, \dots, a_{i_m}$ input szavakat jobbról k_2 db nullával kiegészítve ferde formában a sejttér felső bábsejtsorára adjuk. /A nullák tényleges ráadása természetesen felesleges, elegendő nulla állapotú bábsejteket alkalmazni./

A sejttér működése a következő: a bal oldalon /az asszociatív tárban/ az $a_{i_1}, \dots, a_{i_m}$ szavak felismerése folyik. A

2.25. ábra



meg egyezést jelző vezérlőjelek átlépnek a jobb oldalra, és ott változatlanul haladnak tovább, vezérelve annak működését. A jobb oldalon 0 vezérlőjel esetén az A biteken egyszerűen lefelé shiftelés folyik, 1 jelnél viszont a T biteken tárolt szó átiróódik az A adatcsatornára. A jobb oldal átmenetfüggvénye tehát:



$$\text{ahol } A' = \begin{cases} A & \text{ha } V=0 \\ T & \text{ha } V=1. \end{cases}$$

A működés megértéséhez kísérvük figyelemmel egy a_i input szó áthaladását a sejtterén. Az a_i szó a jobbról hozzátoldott k_2 db 0-val változatlanul halad lefelé az i -edik sorig. Itt - az input és a tárolt szó megegyezése folytán - 1-es vezérlő-jel fog végighaladni a sejtter jobb oldalán, melynek hatására az a_i szóhoz toldott 0-k felüliródnak a b_i szóval. A b_i szó tehát ferdén, az a_i szó meghosszabbításaként fog lefelé haladni az adatsatornán. Feltéve, hogy a szótárban szereplő $a_1, \dots, a_n$ szavak mind különbözők, a b_i szó nem fog újból felül-iródní, és így az a_i szóval együtt hagyja el alól a sejtteret.

A leirt folyamat akkor is ugyanígy játszódik le, ha közvetlen egymás után több input szó érkezik a sejtterbe. Az $a_{i_1}, \dots, a_{i_m}$ input szöveg áthajtásával tehát előáll a $b_{i_1}, \dots, b_{i_m}$ output szöveg, mégpedig minden egyes b_{i_s} a sejtterből kilépő a_{i_s} ferde meghosszabbításaként jelenik meg /2.25. ábra/.

A sejtszerkezet működési elvéből az alábbi járulékos tulajdonságok adódnak:

/i/ Ha egy input szó nem szerepel a szótár bal oldalában, akkor "fordításaként" a 0...0 szó kerül outputra.

/ii/ Ha egy input szó többször is szerepel a szótár bal oldalában, akkor a legalsó előfordulásához tartozó szó kerül fordításként outputra.

Helyigény: $n \cdot (k_1 + k_2)$ 3-bites sejt.

Időigény: a fordítandó $a_{i_1}, \dots, a_{i_m}$ szöveg teljes áthajtásának ideje, azaz $m+n+k_1+k_2$ lépés. Hosszu szöveg fordítása esetén tehát egy input szóra egy lépés jut.

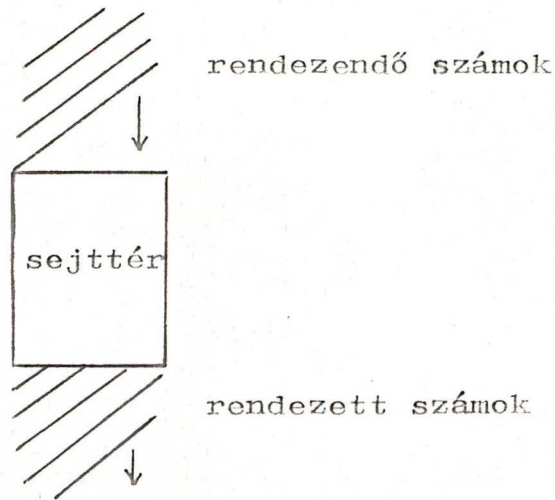
Megjegyezzük, hogy a szimbólumszótárat megvalósító sejt-szerkezet lényegében egy teljes asszociatív tárnak tekinthető, amelynél az i -edik sorban tárolt információt $/a_i$ -t és b_i -t/ az információ egy részének $/a_i$ -nek/ a megadásával olvashatjuk ki. Az alkalmazásoknál rendszerint az asszociatív tárnak ezt a formáját használjuk.

4.4. Rendezés

E pontban bináris számok /szavak/ nagyság szerinti rendezésre adunk sejtalgoritmust. Megjegyezzük, hogy más rendező sejtalgoritmusok is ismertek: NISHIO 1975 egydimenziós sejttérrel $O(n \cdot k)$ lépésben rendez n db k -bites számot, LEGENDI 1977 kétdimenziós sejttérrel $O(n+k)$ lépésben oldja meg a feladatot.

Az alább ismertetett algoritmus - csakugy, mint a fent idézett kettő - szomszédos számok felcserélgetésével jut el a rendezett állapothoz. Jellegzetessége, hogy a kezdőállapotban a sejttér üres, a rendezendő számok pedig ferde formátumban folyamatosan érkeznek a sejttér felső bábsejtsorára, és a sejttérben való áthaladásuk után rendezve hagyják el az alsó bábsejtsort /2.26. ábra/.

2.26. ábra:
az áthaladás közbeni
rendezés sémája.



A megoldás előnye az idézett másik kettővel szemben, hogy több számcsoporthoz egymás utáni rendezése esetén nem szükséges váltáskor a sejttérrel üríteni és tölteni, hanem a számcsoporthoz folyamatosan egymás után érkezhetnek a sejttérbe.

Egyelőre csak egy rendezendő számcsoporthoz tételezzünk fel. Ebben az esetben a rendezendő számok előtt 0-kat /minimális számokat/, utánuk pedig 1-eseket /maximális számokat/ kell a sejttérbe küldeni, mert - amint látni fogjuk - csak így biztosított a rendezés helyes lefolyása. Több egymás utáni számcsoporthoz kezelését e pont végén tárgyaljuk.

A sejtalgoritmus lényege, hogy a rendezendő számok feles sebességgel haladnak át a sejttéren. Így a kisebb számok előreugrálva /teljes lépést lépve/ megelőzhetik a nagyobbakat, vagyis lehetővé válik a szomszédos számok menet közbeni felcserélése.

A sejtalgoritmus legkönnyebben 4-bites sejtekkel realizálható; a sejtek ábrázolásánál a biteket négyzet alakban célszerű elrendezni:

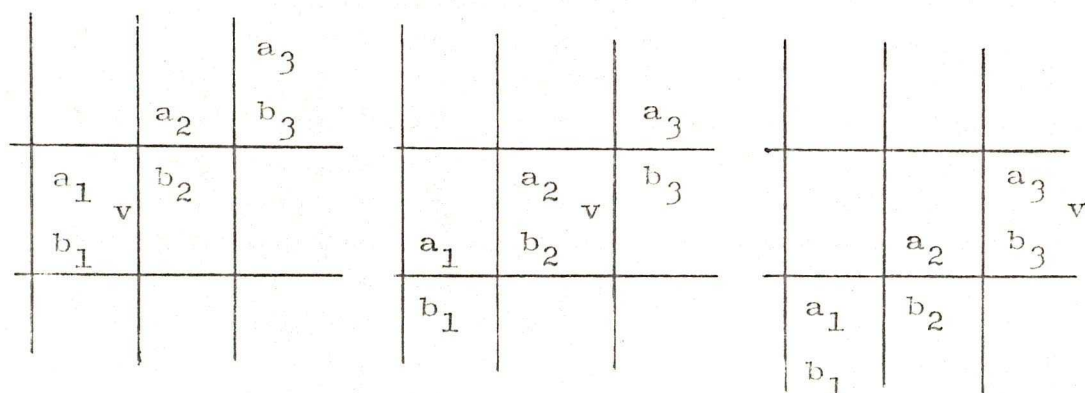
A_1	V_1
A_2	V_2

A_1 és A_2 adatcsatorna-bitek, ezeken haladnak a rendezendő számok feles sebességgel lefelé /az utvonat: felső szomszéd A_2 bitje $\rightarrow A_1 \rightarrow A_2 \rightarrow$ alsó szomszéd A_1 bitje/. A V_1 és V_2 bitek együtt egy kétbites vezérlőjelet tartalmaznak. A vezérlőjelek balról jobbra haladnak, és 3 lehetséges értékük van:

H : hasonlító jel

M : "marad" jel

C : "csere" jel



1. lépés

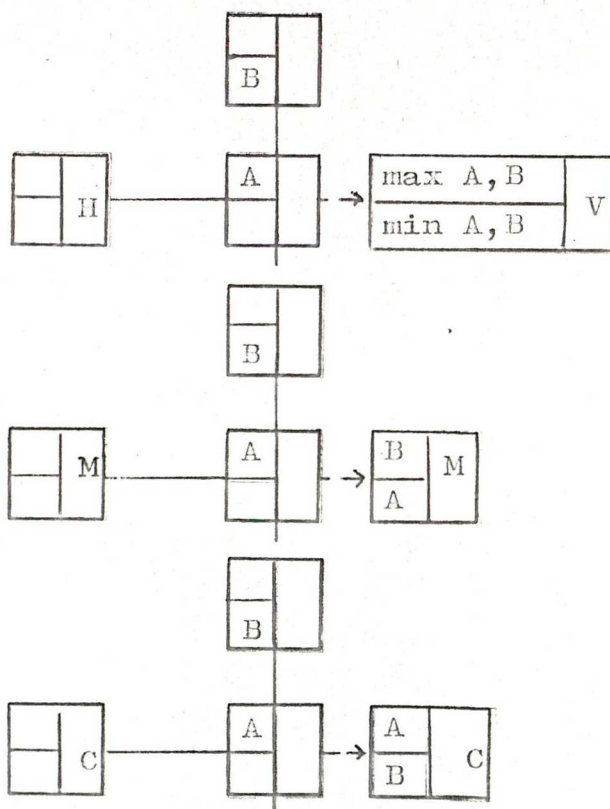
2. lépés

3. lépés

2.27. ábra: két szám összehasonlítása/v kétbites vezérlőjelet jelöl, $a_1 a_2 \dots a_k$ ill. $b_1 b_2 \dots b_k$ pedig az összehasonlitandó számok./

Mivel a rendezendő számokat ferde formátumban adjuk a felső bábsejtsorra, így egy balról jobbra haladó vezérlőjel

épp két egymás utáni számon fut végig /2.27. ábra/, és elvégezheti azok összehasonlítását illetve cseréjét. A bal bábsejtről H jel indul, és addig halad jobbra, amíg megegyező biteket talál. Az első különböző bitpárnál dől el, hogy a két összehasonlított szám közül melyik a nagyobb. Ekkor a H jelből C vagy M jel keletkezik aszerint, hogy kell-e cserélni vagy nem. Az M és C jelek már változatlanul futnak végig a sejtso-



$$\text{ahol } V = \begin{cases} H & \text{ha } A = B \\ M & \text{ha } A < B \\ C & \text{ha } A > B \end{cases}$$

Az eddigiekből világos, hogy két szám összehasonlítását és cseréjét hogyan végzi a sejtter. A következőkben megvizsgáljuk, hogy n szám áthajtása esetén mi történik.



Nevezzünk egy számcsoportot hátról m_1 elemre rendezett-nek, ha a csoport m_1 db legnagyobb eleme növekvő sorrendben a csoport végén helyezkedik el /a rendezés során ezek tehát már nem változtatják helyüket/.

Állítás: Ha egy egysoros rendező sejtterén egy hátról m_1 elemre rendezett számcsoportot hajtunk át, akkor a sejt-sort elhagyó számcsoport hátról m_1+1 elemre rendezett lesz.

Bizonyítás: Legyenek a számcsoport elemei $a_1, \dots, a_{m-m_1}, \dots, a_m$ és legyen $\bar{a}$ az $a_1, \dots, a_{m-m_1}$ számok legnagyobbika. Az $a_1, \dots, a_{m-m_1}$ rekordok áthaladása során $\bar{a}$ minden párcserénél hátra kell hogy kerüljön, így a rendező sejt-sor A_1 bitjein "bevárja" az $a_{m-m_1+1}, \dots, a_m$ számokat, és a_{m-m_1+1} előtt fog kilépni a sejtsorból. Ezzel a file m_1+1 elemre rendezett lesz.

n soros rendező sejtter esetén minden sejtsor a fölötte lévő sejtsorból kilépő output file-t kapja inputként, így evidens, hogy egy n soros rendező sejtter egy hátról m_1 elemre rendezett számcsoportból hátról m_1+n elemre rendezett készit.

Ezek szerint n szám teljes lerendezéséhez n soros sejt-ter szükséges^{*}. Ha azonban több rendezendő számunk van, mint

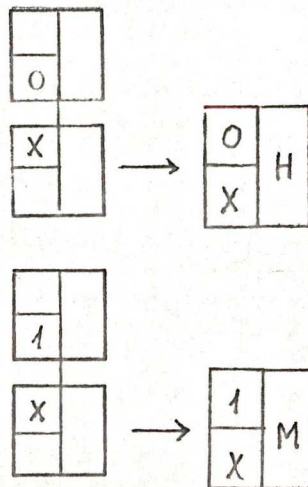
^{*}Valójában $n-1$ sor is elegendő lenne, hisz egy hátról $n-1$ elemre rendezett számcsoport már teljesen rendezett.

ahány soros a sejtter ($m > n$), akkor is megoldható a rendezés a számcsoport többszöri áthajtásával: minden áthajtás n -nel növeli a csoport "rendezettségét", így m/n áthajtás után a számcsoport rendezve hagyja el a sejtteret.

Már említettük, hogy a rendezendő számok után 1-eseket kell a sejtterbe küldeni mindaddig, míg a rendezett számcsoport alul el nem hagyja a sejtteret. Csak ekkor kezdődhet - a sejtter nullázása után - újabb számcsoport rendezése.

Több egymás utáni számcsoport rendezésére ez a módszer nyilvánvalóan nehézkes. A következőkben megmutatjuk, hogy a sejtszerkezet kis módosításával az egymás utáni számcsoportok közvetlenül követhetik egymást, így a sejtter teljes kihasználtsággal működhet.

Ha a számcsoportok közvetlen egymás után érkeznek, akkor az egyes csoportok kezdetét valamilyen módon jelölnünk kell. E célból minden számot balról egy jelzőbittel egészítünk ki, melynek értéke 1 a csoportok első számainál, 0 a többi számnál. Ennek megfelelően a sejtteret is meg kell toldanunk balról egy oszloppal, amely a jelzőbiteket kezeli, és a rendező vezérlőjelek indításáról gondoskodik: 0 jelzőbit esetén II jelet indít, 1 jelzőbitnél /két számcsoport határán/ pedig M jelet, amely megakadályozza a két csoport elemeinek keveredését. A sejtter első oszlopának átmenetfüggvénye pontosan a következő:



Az utolsó számcsoporthoz még egy 1-es jelzőbittel ellátott elemet /pl. 10...0/ kell a sejtterbe küldeni, ez biztosítja a sejtter kiürülését.

Helyigény: n db k -bites szám egyszeri áthajtással történő rendezéséhez n soros sejtter, azaz $n \cdot (k+1)$ 4-bites sejt szükséges. Többszöri áthajtással azonban tetszőleges kisebb sejtterrel is elvégezhető a feladat.

Időigény: n szám és n soros sejtter esetén a rendezés időigénye a teljes áthajtás idejével azonos, vagyis $3n+k$ lépés. Ha azonban egymás után több n elemű számcsoporthoz rendezünk, akkor egy csoportra csak n lépés jut, hisz a csoportok közvetlen egymás után érkezhetnek a sejtterbe.

III. INVERTÁLHATÓ SEJTAUTOMATÁK =====

1. A VIZSGÁLATOK CÉLJA, AZ EREDMÉNYEK ÖSSZEFOGLALÁSA

A sejtautomaták által indukált globális leképezéseket számos automataelméleti dolgozatban vizsgálják. Ezen vizsgálatok célja a sejtautomaták viselkedésének minél pontosabb algebrai leírása, ill. a sejtterek számítási képességeinek feltárása volt. Ebbe a témakörbe tartozik a globális leképezések invertálhatóságát vizsgáló számos tanulmány is /AMOROSO 1971, TOFFOLI 1972, MARUOKA 1976/. Az invertálhatóság vizsgálatánál a kérdés matematikai érdekessége mellett szerepet játszhatott az a gondolat is, hogy a sejttér a fizikai tér egyfajta diszkrét modelljének tekinthető, és így a sejtautomata invertálhatósága a modellezett fizikai folyamatok reverzibilis voltának felel meg. Másszóval, egy sejtautomata invertálhatóságának vizsgálata azzal a kérdésfeltevéssel rokon, hogy a világ jelenlegi állapota egyértelműen meghatározza-e a multbeli állapotokat.

A fentiekkel ellentétben az e fejezetben foglalt invertálhatósági vizsgálatokat egy konkrét gyakorlati cél érdekében végeztük: konkrétan lokálisan invertálható sejt-

automaták konstruálására törekedtünk, amelyeknél a globális leképezés nemcsak invertálható, hanem annak inverze egy másik sejtautomatával realizálható. Így például nagy teljesítményű kódoló-dekódoló automatapárokat nyerhetünk, amelyek már önmagukban is széleskörű alkalmazási területtel bírnak, de Legendi Tamás javaslata alapján alapot jelenthetnek hatékony információ-tömörítő rendszerek kifejlesztéséhez is.

Nemtriviális, lokálisan invertálható sejtautomata konstruálása nehéz feladat. Az első intuitív próbálkozások sikertelenek voltak, sőt bizonyos speciális esetekre sikerült is bizonyítani, hogy csak a triviális inverzek léteznek /KATONA 1978/. A kérdés általános esetben való eldöntéséhez szükségessé vált a lokális invertálhatóság, mint algebrai tulajdonság mélyebb elméleti vizsgálata. Ezzel foglalkozik a jelen fejezet első része. A szükséges definíciók után a 3. pontban megadjuk a lokális invertálhatóság szükséges és elegendő feltételét, amelynek alapján a 4. pontban algoritmust adunk invertálható sejtautomaták konstruálására. Ezen algoritmus azonban az általános esetben még számítógépes végrehajtás mellett is túl nagy tár- és időigénnyel bír, ami gyakorlati alkalmazását korlátozza.

A fenti nehézség miatt irányult a figyelem egy speciális sejtautomataosztály, a lineáris sejtautomaták^{*} felé.

^{*}A "lineáris sejtautomata" elnevezéssel gyakran a sejtter egydimenziós voltára utalnak, itt azonban az elnevezés az átmenetfüggvények lineáris algebrai tulajdonságaira vonatkozik.

Ezen sejtautomaták viselkedése jól leírható a lineáris algebra eszközeivel, és az inverz-konstruáló algoritmus hely- és időigénye is drasztikusan csökkenthető, ha csak lineáris sejtautomatákra szorítkozunk. Ezt a lineáris inverz-konstruáló algoritmust a 6. pontban fejtjük ki. Az algoritmusra számítógépes program készült, melynek futtatása során kiderült, hogy már a lineáris sejtautomaták körében is nagyszámu /inhomogén/ lokálisan invertálható változat létezik. Ha a térbeli inhomogenitás mellett még az időbeli inhomogenitás lehetőségét is figyelembe vesszük, akkor igen gazdag választék áll rendelkezésre lokálisan invertálható sejtautomatákból, melyek a kódolási feladatnak már önmagukban megoldását képezik, és alapot adnak más alkalmazások felkutatásához is.

Az inverzkonstruáló program futási eredményeiből a Függelékben mutatunk be példákat.

2. DEFINÍCIÓK

2.1. A véges, inhomogén sejtautomata fogalma

Az invertálhatóságra irányuló algebrai vizsgálatoknál nehézkes lenne az I. fejezetben bevezetett sejtprocesszor fogalommal dolgozni. Ezért e fejezetben a véges, inhomogén sejtautomata fogalmát használjuk, amely részben szűkebb részben tágabb a sejtprocesszorhoz képest. A két fogalom kapcsolatára az 1. állítás fog rávilágítani.

1. Definíció: Véges, inhomogén sejtautomatán egy (C, N, A, f) négyest értünk, ahol

$C = \{c_1, \dots, c_m\}$ a sejtek /véges/ halmaza,

$N: c_i \mapsto (c_{i_1}, \dots, c_{i_{n_i}})$ a szomszédsági függvény, amely a c_i sejthez n_i db szomszédot rendel*. A szomszédsági struktúrában tehát korlátlan inhomogenitás lehetséges.

$A = \{0, 1, \dots, s-1\}$ a sejtállapotok halmaza,

$f: c_i \mapsto f_i$ a függvényrendszer, amely minden c_i sejthez egy $f_i: A^{n_i} \mapsto A$ lokális átmenetfüggvényt rendel.

Véges, inhomogén sejttéren a (C, N, A) hármast értjük, vagyis eltekintünk a sejtek átmenetfüggvényeitől. A sejtautomaták elméletében szokásos elnevezéseket használva konfigurációnak nevezzük a sejttér egy lehetséges állapotát,

*A definíció szerint $N(c_i)$ egy rendezett n_i -es, a későbbiekben azonban időnként n_i -elemű halmaznak fogjuk tekinteni, ahol ez a tárgyalást megkönnyíti.

vagyis egy $\alpha: C \rightarrow A$ leképezést. Az f függvényrendszer által indukált globális átmenetfüggvényen azt az $F: {}^C A \rightarrow {}^C A$ leképezést értjük, amelyre

$$F(\alpha)|_{c_i} = f_i(\alpha(c_{i_1}), \dots, \alpha(c_{i_{n_i}})).$$

A fenti modell a sejthálózat teljes inhomogenitását engedi meg /hasonló modellt alkalmaz pl. DIERK 1977/. Bár sejtautomatán általában homogén rendszert értenek, az inhomogén modell mellett az alábbi érvek szólnak:

- Az inhomogén modellből speciális esetként könnyen adódik a homogén, ugyanakkor egy homogén modell szükségképpen nem tartalmazna számos gyakran vizsgált "majdnem homogén" típust /ilyen pl. a véges, téglalap alakú sejttér, amely a szélein szükségképpen inhomogén/.

- A homogén és "majdnem homogén" sejtterek számos algebrai tulajdonsága könnyen általánosítható a tetszőlegesen inhomogén esetre is.

- Egyes számítási feladatokra célszerű olyan párhuzamos működésű célgépet készíteni, amely felépítésében épp a fenti, teljesen inhomogén modellnek felel meg /sejtes célhardware/. Ezenkívül a fenti inhomogén modell rokonságot mutat pl. az idegrendszer strukturájával is.

A következőkben a véges inhomogén sejtautomatának a

sejtprocesszorhoz való viszonyát vizsgáljuk.

Legyen adott egy (CUC_d, N, AxB) sejtprocesszor az $(\alpha, \beta, D, \mathcal{F})$ sejttérprogrammal. Ez a sejtprocesszor minden $(t-1) \rightarrow t$ lépés során egy $F^{(t)}: C_A \rightarrow C_A$ globális átmenetet hajt végre, amelyet $\beta, d^{(t)}, f^{(t)}$ határoznak meg.

1. Állítás: Egy (CUC_d, N, AxB) sejtprocesszorhoz $\beta, d^{(t)}, f^{(t)}$ ismeretében megadható olyan (C', N', A', f') inhomogén sejtautomata, amely az $F^{(t)}$ globális átmenetfüggvényt indukálja.

Bizonyítás: Legyen $C'=C$ és $A'=A$. Az N' szomszédsági függvény csak annyiban különbözik N -től, hogy - bábsejtek nem lévén - a szélső sejtekhez kevesebb szomszédot rendel. Az F függvényrendszer minden $c_{i,j}$ belső sejthez az $f^{(t)}$ összevont függvénynek a $\beta(c_{i,j})$ belső állapothoz tartozó részfüggvényét rendeli, míg a szélső sejteknél $d^{(t)}$ figyelembe vételével adódnak a megfelelő, szűkebb szomszédságra definiált átmenetfüggvények. Q.E.D.

Következmény: Ha a sejtprocesszort csak egyetlen lépésben vizsgáljuk, illetve ha minden t -re $d^{(t)} = d$ és $f^{(t)} = f$ /időbeli homogenitás/, akkor a sejtprocesszor egy inhomogén sejtautomatával szimulálható.



2.2. Az invertálhatóság fogalma

Legyen (C, N, A, f) egy véges inhomogén sejtautomata, amely az F globális átmenetfüggvényt indukálja.

2. Definíció: A (C, N, A, f) sejtautomatát /ill. az f függvényrendszert/ globálisan invertálhatónak nevezzük, ha F egy-egyértelmű, azaz F^{-1} létezik.

3. Definíció: Egy $\alpha: C \rightarrow A$ konfigurációt Édenkert-konfigurációnak szokás nevezni, ha nincs megelőzője, vagyis nincs olyan β konfiguráció, amelyre $F(\beta) = \alpha$.

Nyilvánvaló, hogy egy sejtautomata akkor és csak akkor globálisan invertálható, ha nincs Édenkert-konfigurációja.

4. Definíció: A (C, N, A, f) sejtautomatát /ill. az f függvényrendszert/ lokálisan invertálhatónak /vagy egyszerűen csak invertálhatónak/ nevezzük, ha F egy-egyértelmű, és létezik olyan g függvényrendszer, hogy a (C, N, A, g) sejtautomata az F^{-1} globális átmenetfüggvényt indukálja.

A definíciókból világos, hogy a lokális invertálhatóság erősebb tulajdonság a globálisnál, az utóbbi szükséges feltétele az előbbinek.

A lokális invertálhatósággal kapcsolatban még egy fogalomra lesz szükségünk:

5. Definíció: A (C, N, A, f) sejtautomata /ill. az f

függvényrendszer/ a c_i sejtben invertálható, ha a c_i sejt-
hez megadható egy olyan $g_i: A^{n_i} \rightarrow A$ inverz lokális átme-
netfüggvény, amelyet bármely $F(\alpha)$ képkonfigurációban c_i
szomszédságára alkalmazva visszakapjuk c_i eredeti állapotát,
az $\alpha(c_i)$ értéket.

Fontos a következő:

2. Tétel: Egy (C, N, A, f) sejtautomata lokálisan in-
vertálható $\iff$ ha minden $c_i \in C$ sejtben invertálható.

Bizonyítás: $(\implies)$ triviális.

$(\impliedby)$ Mivel minden $c_i (\in C)$ sejt-
hez van egy g_i inverz
lokális átmenetfüggvény, így ezekből egy g inverz függ-
vényrendszer állítható össze. Világos, hogy g tetszőleges
 $F(\alpha)$ képkonfigurációra alkalmazva α -t adja. Ebből az is
következik, hogy F egy-egy-értelmű, és a g függvényrendszer
 F^{-1} -et indukálja.

3. AZ INVERTÁLHATÓSÁG SZÜKSÉGES ÉS ELEGENDŐ FELTÉTELEI

3.1. A globális invertálhatóság szükséges feltétele

E pontban megmutatjuk, hogy a lokális átmenetfüggvények vizsgálatából milyen következtetéseket vonhatunk le a globális átmenetfüggvény invertálhatóságára, ill. az Édenkert-konfigurációk számára vonatkozóan. Hasonló vizsgálatokat RICHARDSON 1972 és MARUOKA 1976 tartalmaz homogén végtelen sejtterek esetében.

Legyen $f_i: A^{n_i} \rightarrow A$ egy lokális átmenetfüggvény. Ezen függvény jellemzésére az alábbiakban néhány fogalmat vezetünk be.

Legyen $a \in A$ egy sejtállapot. $p_i^{(a)}$ -val jelöljük azon különböző $(a_1, \dots, a_{n_i})$ "szomszédságok" számát, amelyekre $f_i(a_1, \dots, a_{n_i}) = a$. Világos, hogy

$$p_i^{(0)} + \dots + p_i^{(s-1)} = s^{n_i}.$$

A $p_i^{(a)}$ értékek átlagát p_i -vel jelöljük, nyilván

$$p_i = \frac{s^{n_i}}{s} = s^{n_i-1}.$$

6. Definíció: Az f_i átmenetfüggvényt kiegyensúlyozottnak nevezzük, ha minden $a \in A$ állapotra $p_i^{(a)} = p_i$. Az átmenetfüggvény kiegyensúlyozatlanságának mértékén a

$$q_i = \frac{1}{2}(|p_i^{(0)} - p_i| + \dots + |p_i^{(s-1)} - p_i|)$$

összeget értjük. Nyilván

$$q_i = \sum_{\substack{(a) \\ p_i^{(a)} > p_i}} (p_i^{(a)} - p_i) = \sum_{p_i^{(a)} < p_i} (p_i - p_i^{(a)}) .$$

Az alábbi tétel rávilágít a lokális és globális átmenetfüggvények kapcsolatára.

3. Tétel: Ha egy (C, N, A, f) sejtautomatában az f_i átmenetfüggvény kiegyensúlyozatlanságának mértéke q_i , akkor a sejtautomatának legalább $q_i \cdot s^{m-n_i}$ Édenkert-konfigurációja van /m az összes sejtek száma/.

Bizonyítás: Világos, hogy s^{m-n_i} számú olyan konfiguráció van, amelyben a c_i sejt szomszédsága egy adott $(a_1, \dots, a_{n_i})$ állapotban van. Ezért az összes olyan konfigurációk száma, amelyből c_i egy adott "a" állapotba megy át, nyilván $p_i^{(a)} \cdot s^{m-n_i}$. Ugyanakkor az összes olyan konfigurációk száma, amelyekben c_i az "a" állapotban van, éppen $s^{m-1} = p_i \cdot s^{m-n_i}$. Ha tehát $p_i^{(a)} < p_i$, akkor ezen $p_i \cdot s^{m-n_i}$ konfigurációból $(p_i - p_i^{(a)}) \cdot s^{m-n_i}$ szükségképpen Édenkert-konfiguráció. Hasonló a helyzet minden olyan "a" állapotnál, melyre $p_i^{(a)} < p_i$, tehát összesen legalább

$$\sum_{p_i^{(a)} < p_i} (p_i - p_i^{(a)}) \cdot s^{m-n_i} = q_i \cdot s^{m-n_i}$$

Édenkert-konfiguráció van. Q.E.D.

Következmények:

(i) Egy (C, N, A, f) sejtautomatának legalább

$$\max \left\{ q_1 \cdot s^{m-n_1}, \dots, q_m \cdot s^{m-n_m} \right\}$$

Édenkert-konfigurációja van, ahol q_i az f_i átmenetfüggvény kiegyensúlyozatlanságának mértékét jelöli.

(ii) Egy sejtautomata globális invertálhatóságának szükséges feltétele, hogy minden egyes sejt átmenetfüggvénye kiegyensúlyozott legyen.

3.2. A lokális invertálhatóság szükséges-elegendő feltétele

A 2. tétel alapján elegendő az egy sejtben való invertálhatóságot vizsgálni, vagyis meghatározni, hogy egy adott f függvényrendszernek egy adott c_i sejtben való invertálhatóságát hogyan tudjuk eldönteni.

Próbáljuk megkonstruálni a g_i inverz lokális átmenetfüggvényt! Kiindulva egy tetszőleges α konfigurációból, a c_i szomszédainak $f_{i_1}, \dots, f_{i_{n_i}}$ átmenetfüggvényeit ismerve ki tudjuk számítani a szomszédok $F(\alpha)$ -beli állapotát, le-

gyenek ezek rendre $b_{i_1}, \dots, b_{i_{n_i}}$. Mivel a konstruálandó ε_i -nek ezekből az állapotokból kell $\alpha(c_i)$ -t visszaadnia, így felírhatjuk ε_i egy "term"-jét:

$$(b_{i_1}, \dots, b_{i_{n_i}}) \rightarrow \alpha(c_i),$$

ami azt fejezi ki, hogy ε_i -re a $\varepsilon_i(b_{i_1}, \dots, b_{i_{n_i}}) = \alpha(c_i)$ feltételnek teljesülnie kell.

Ha a fenti eljárást minden α konfigurációra elvégezzük, ezáltal termék egy véges halmazát nyerjük. Ha ezek között van ellentmondó, azaz például valamely $b_{i_1}, \dots, b_{i_{n_i}}$ -re

$$\begin{aligned} (b_{i_1}, \dots, b_{i_{n_i}}) &\rightarrow a_1 \\ \text{és} \quad (b_{i_1}, \dots, b_{i_{n_i}}) &\rightarrow a_2 \end{aligned}$$

ahol $a_1 \neq a_2$, akkor a ε_i függvény nyilván nem létezik, és így az f függvényrendszer c_i -ben nem invertálható. Ha viszont nem kapunk ellentmondó termeket, akkor a ε_i függvény létezik, hisz lényegében megkonstruáltuk, és így f a c_i -ben invertálható.

A leírt döntési eljárás minden egyes α konfiguráció megvizsgálására épül. Fontos azonban, hogy a vizsgált α konfigurációnak csak az $N(N(c_i))$ -be eső /vagyis a szomszédok szomszédainak halmazára eső^{*}/ részével kell tényle-

^{*}Egész pontosan $N(N(c_i)) := \{c_k \mid \exists c_j: c_j \in N(c_i) \text{ és } c_k \in N(c_j)\}$.

gesen számolni. Ennek figyelembe vételével az eljárás a 3.1. ábra szerinti táblázat, az un. inverzkonstruáló tábla felépítéséből és vizsgálatából áll.

Az $N(N(c_i))$ -beli sejtek állapota α -ban	Az $N(c_i)$ beli sejtek állapota $F(\alpha)$ -ban	az $\alpha(c_i)$ értékek
0 ... 0 0		
0 ... 0 1		
⋮	⋮	⋮
s-1...s-1 s-1		

3.1. ábra: Az inverzkonstruáló tábla

A táblázat első oszlopában az $N(N(c_i))$ -beli sejtek összes lehetséges állapota szerepel, a második oszlopban rendre kiszámítjuk az $f_{i_1}, \dots, f_{i_{n_i}}$ átmenetfüggvények segítségével a megfelelő $F(\alpha)$ -beli állapotokat, míg a harmadik oszlopba átmásoljuk az első oszlopból az $\alpha(c_i)$ értékeket. A táblázat második és harmadik oszlopa együtt a g_i függvény termjeit adja.

A leírtakból adódik a következő tétel:

4. Tétel: Az f függvényrendszer a c_i sejtben invertálható $\iff$ ha felírva a c_i sejt inverzkonstruáló tábláját, annak második oszlopában minden megegyező $N(c_i)$ -állapothoz a harmadik oszlopban azonos sejtállapot tartozik.

Egy (C, N, A, f) sejtautomata lokális invertálhatóságának eldöntéséhez tehát minden egyes sejthez fel kell írunk az inverzkonstruáló táblát, és azon elvégezni a leírt vizsgálatot. Az f függvényrendszer akkor és csak akkor lesz lokálisan invertálható, ha minden egyes táblára teljesül a 4. tétel feltétele. Ez a hosszadalmasnak tűnő vizsgálat a gyakorlatban jelentősen egyszerűsödik, hisz pl. homogén sejtautomata esetén minden sejt inverzkonstruáló táblája azonos, így egyetlen tábla vizsgálata elegendő.

Tekintve, hogy a c_i sejt inverzkonstruáló tábláját egyértelműen meghatározzák a c_i sejt szomszédainak $f_{i_1}, \dots, f_{i_{n_i}}$ átmenetfüggvényei, így jogos a következő definíció:

7. Definíció: Ha az f függvényrendszer a c_i sejtben invertálható, akkor a c_i sejt szomszédainak $(f_{i_1}, \dots, f_{i_{n_i}})$ átmenetfüggvényeit invertálható függvénykombinációnak nevezzük.

A fenti definíció is kifejezi, hogy a lokális invertálhatóság, mint tulajdonság több lokális feltétel együttes teljesülésével egyenértékű. Erre az észrevételre alapul a következő pontban vázolt inverzkonstruáló eljárás.

4. AZ INVERZKONSTRUÁLÓ ALGORITMUS

Felhasználva az előző pont eredményeit, melyek szerint a sejtautomaták lokális invertálhatósága lokális feltételek teljesüléséhez kötött, invertálható függvényrendszerek konstruálására az alábbi kétfázisu módszert alkalmazhatjuk:

Első fázis: Invertálható függvénykombinációkat keressünk minden egyes sejthez, lehetőleg az összes lehetőséget előállítjuk.

Második fázis: Az invertálható függvénykombinációkból teljes függvényrendszert rakunk össze, úgy, hogy minden egyes sejtre invertálható függvénykombináció illeszkedjen. Ekkor szükségképpen a teljes függvényrendszer is invertálható lesz.

Az első fázisnál a 3. tétel nyújt segítséget: mivel a lokális invertálhatóságnak szükséges feltétele a globális invertálhatóság, ez utóbbihoz pedig a lokális átmenetfüggvények kiegyensúlyozottsága szükséges, így csak kiegyensúlyozott átmenetfüggvények jöhetnek szóba az invertálható függvénykombinációk keresésénél. Azonban már kis állapotszám mellett is igen sok kiegyensúlyozott átmenetfüggvény van, és még több az ezekből képezhető függvénykombinációk száma. Minden ilyen függvénykombinációhoz felírni az inverzkonstruáló táblát, és azt a 4. tétel kritériuma szerint megvizsgálni, ez még számítógéppel is reménytele-

nül nagy feladat.

A második fázisnál sem jobb a helyzet. A tapasztalatok szerint már kis állapotszám esetén is igen sok invertálható függvénykombináció van, ugyanakkor ezek összeillesztése meglehetősen nehéz feladat a többszörös átfedések miatt. Így tehát egy számítógépes összeadó program, amellyel egy adott sejttérhez az összes lehetséges invertálható függvényrendszert generálni kívánjuk, irreálisan nagy gépidőt igényelne.

A fenti nehézségek miatt döntöttünk úgy, hogy az inverzkonstruáló algoritmust egy speciális sejtautomata-osztályra, a lineáris sejtautomatákra fejlesztjük ki. Ezáltal - mint látni fogjuk - az irreálisan nagy tár- és időigények jelentéktelenre csökkentek, ugyanakkor - a futási eredmények tanúsága szerint - nagyszámu invertálható függvénykombinációt és függvényrendszert sikerült előállítani.

A következő pontban a lineáris sejtautomaták alapvető algebrai tulajdonságait tárgyaljuk, majd a 3. pontban kimondott feltételeket visszük át lineáris esetre.

A 6. pont a hatékony lineáris inverzkonstruáló algoritmus részletes ismertetését tartalmazza.

5. LINEÁRIS SEJTAUTOMATÁK

5.1. Definíciók, alaptulajdonságok

Legyen a továbbiakban (C, N, A) olyan sejttér, ahol az A állapothalmaz véges, kommutatív, egységelemes gyűrűt alkot valamely "+" és "." műveletekre. A legegyszerűbb esetben A egy maradékosztály-gyűrű: $A = \{0, 1, \dots, s-1\}$, "+" és "." pedig a modulo- s összeadást illetve a modulo- s szorzást jelenti. A "+" és "." jelek a következőkben mindig az A -beli műveleteket fogják jelölni.

Világos, hogy egy $\alpha: C \rightarrow A$ konfigurációt egy $(a_1, \dots, a_m)$ A -feletti vektornak is tekinthetünk, ahol $a_i = \alpha(c_i)$ minden i -re.^{*} Definíáljuk továbbá két konfiguráció összegét, ill. konfigurációnak konstanssal való szorzatát, a következőképpen:

$$\alpha + \beta := (a_1 + b_1, \dots, a_m + b_m) \quad \text{ahol } a_i = \alpha(c_i), \quad b_i = \beta(c_i) \quad \text{minden } i\text{-re,}$$
$$k \cdot \alpha := (k \cdot a_1, \dots, k \cdot a_m) \quad \text{ahol } a_i = \alpha(c_i) \quad \text{minden } i\text{-re.}$$

Ezekután nyilvánvaló a következő:

5. Állítás: A (C, N, A) sejttér összes lehetséges konfigurációjának ${}^C A$ halmaza m -dimenziós vektorteret képez az A gyűrű felett, ahol m a sejtek számát jelenti.

^{*}A konfigurációk vektorszerű jelölése független azok tényleges - két vagy többdimenziós - topológiájától; a sejttér számára továbbra is tetszőleges szomszédsági viszonyok megengedettek.

8. Definíció: Egy $f_i: A^{n_i} \rightarrow A$ lokális átmenetfüggvényt lineárisnak nevezünk, ha léteznek olyan $k_1, \dots, k_{n_i}$ A -beli konstansok /tk. sejtállapotok/, hogy

$$f_i(a_1, \dots, a_{n_i}) = k_1 a_1 + \dots + k_{n_i} a_{n_i}$$

minden $a_1, \dots, a_{n_i}$ állapotra.

9. Definíció: A (C, N, A, f) sejtautomatát lineárisnak nevezük, ha az általa indukált $F: {}^C A \rightarrow {}^C A$ globális átmenetfüggvény a ${}^C A$ vektortérnek egy lineáris transzformációja, vagyis minden α, β konfiguráció és $k \in A$ esetén

$$F(\alpha + \beta) = F(\alpha) + F(\beta) \quad \text{és}$$

$$F(k \cdot \alpha) = k \cdot F(\alpha) .$$

Az algebrából ismert, hogy minden $F: {}^C A \rightarrow {}^C A$ lineáris transzformációhoz létezik egy egyértelműen meghatározott K $m \cdot m$ -es A -feletti mátrix, amelyre $F(\alpha) = K \cdot \alpha$ minden $\alpha \in {}^C A$ esetén / α -t itt oszlopmátrixnak tekintettük, a " $\cdot$ " mátrixszorzást jelöl/. Egy F lineáris globális átmenetfüggvényhez ily módon kapott K mátrixot a lineáris globális átmenetfüggvény mátrixának fogjuk nevezni.

Alapvető fontossága a következő tétel:

6. Tétel: Egy (C, N, A, f) sejtautomata lineáris $\iff$ ha minden f_i lokális átmenetfüggvény lineáris.

Bizonyítás: $(\implies)$ F lineáris, ezért van olyan K mát-

rix, hogy $F(\alpha) = K \cdot \alpha$ minden α -ra. A mátrixszorzás szabályából adódik, hogy a c_i sejt új állapota $k_{i,1}a_1 + \dots + k_{i,m}a_m$ lesz, ahol $a_i = \alpha(c_i)$. Mivel azonban az új állapot csak a szomszédok régi állapotától függhet, így csak a $k_{i,i_1}, \dots, k_{i,i_{n_i}}$ konstansok lehetnek 0-tól különbözők. Ezzel azt kaptuk, hogy $f_i(a_{i_1}, \dots, a_{i_{n_i}}) = k_{i,i_1}a_{i_1} + \dots + k_{i,i_{n_i}}a_{i_{n_i}}$, vagyis hogy az f_i átmenetfüggvény lineáris.

($\Leftarrow$) Ha minden f_i lineáris, akkor a K mátrixot megkonstruálhatjuk a következőképpen: ha c_j szomszédja c_i -nek, akkor $k_{i,j}$ legyen az f_i lineáris átmenetfüggvényben a c_j együtthatója; ha c_j nem szomszédja c_i -nek, akkor pedig $k_{i,j} := 0$. Világos, hogy az így kapott K mátrixra $F(\alpha) = K \cdot \alpha$, tehát F lineáris. Q.E.D.

A fenti bizonyításból nyilvánvalóvá vált az alábbi összefüggés:

7. Állítás: Valamely lineáris sejtautomata globális átmenetfüggvényének K mátrixában a $k_{i,j}$ elem megegyezik az f_i átmenetfüggvény formulájában c_j együtthatójával.

Gyakran fogjuk használni a következő két fogalmat, és a közöttük fennálló nyilvánvaló összefüggést:

10. Definíció: Egy (C, N, A) sejtter N szomszédsági függvénye által indukált szomszédsági-reláció-mátrixnak nevezzük azt az $\bar{N} = [\bar{n}_{i,j}]$ $m \cdot m$ -es mátrixot, amelyre

$$\bar{n}_{i,j} = \begin{cases} 1 & \text{ha } c_i\text{-nek szomszédja } c_j, \\ 0 & \text{egyébként.} \end{cases}$$

11. Definíció: Egy K $m \cdot m$ -es A -feletti mátrixot globális átmenetfüggvény-mátrixnak nevezünk a (C, N, A) sejttér felett, ha megadható olyan f lineáris függvényrendszer, hogy a (C, N, A, f) sejtautomata globális átmenetfüggvényének mátrixa éppen K .

8. Állítás: Egy $K = [k_{i,j}]_{m \cdot m}$ -es A -feletti mátrix globális-átmenetfüggvény-mátrix a (C, N, A) sejttér felett $\iff$ ha minden i, j -re $\bar{n}_{i,j} = 0$ esetén $k_{i,j} = 0$.

5.2. Lineáris sejtautomaták invertálhatósága

E pontban a globális, lokális és egy sejten való invertálhatóság kérdését vizsgáljuk lineáris esetben. A kapott eredmények képezik majd az alapját a következő pontban ismertetett inverzgeneráló algoritmusnak.

Egy A gyűrű feletti K mátrixot akkor nevezünk invertálhatónak, ha létezik olyan A -feletti K^{-1} mátrix, hogy $K \cdot K^{-1} = K^{-1} \cdot K = E$, ahol E az egységmátrix. Ismert algebrai tétel, hogy egy A -feletti K mátrix akkor és csak akkor invertálható, ha determinánsa nem zérusosztó*.

*Egy $a(\in A)$ elem zérusosztó, ha van olyan $b(\in A)$ nemnulla elem, hogy $a \cdot b = 0$.

Ezekután nyilvánvaló a következő két tétel:

9. Tétel: Egy (C, N, A, f) lineáris sejtautomata globálisan invertálható $\iff$ ha az indukált globális átmenetfüggvény mátrixának determinánsa nem zérusosztó.

10. Tétel: Egy (C, N, A, f) lineáris sejtautomata lokálisan invertálható $\iff$ ha az indukált globális átmenetfüggvény mátrixa invertálható /azaz determinánsa nem zérusosztó/, és az inverz mátrix globális-átmenetfüggvény-mátrix (C, N, A) felett.

Az egy sejtből való invertálhatóságról a következőket lehet mondani:

11. Tétel: Egy (C, N, A, f) lineáris sejtautomata a c_i sejtből invertálható $\iff$ ha a sejtautomata globális átmenetfüggvényének K mátrixához megadható olyan

$h_i = (h_{i,1}, \dots, h_{i,m})$ A -feletti vektor, amelyben csak a $h_{i,i_1}, \dots, h_{i,i_{n_i}}$ elemek különbözhetnek 0-tól és amelyre $h_i \cdot K = e_i$, ahol e_i az i -edik m -dimenziós egységvektort jelöli. / h_i -t és e_i -t itt $1 \cdot m$ -es mátrixoknak tekintjük./

Bizonyítás: $(\Rightarrow)$ A feltételezés szerint létezik egy g_i inverz lokális átmenetfüggvény. Először megmutatjuk, hogy g_i lineáris, pontosabban azzá tehető.

Mivel g_i inverz függvény, így c_i -nek csak azon környezetekre értelmezett, amelyek valamely $F(\alpha)$ képkonfigurációban fellepnek. Világos, hogy ezek a környezetek egy

lineáris alteret képeznek az összes környezetek n_i -dimenziós vektorterében. Erre az altérre ε_i linearitása könnyen igazolható /az egyszerűbb írásmód kedvéért ε_i -t mint $C_A \rightarrow A$ leképezést tekintettük/:

$$\begin{aligned}\varepsilon_i(F(\alpha)) + \varepsilon_i(F(\beta)) &= \alpha(c_i) + \beta(c_i) = \\ &= (\alpha + \beta)(c_i) = \varepsilon_i(F(\alpha + \beta)), \text{ valamint} \\ k \cdot \varepsilon_i(F(\alpha)) &= k \cdot \alpha(c_i) = \varepsilon_i(F(k \cdot \alpha)).\end{aligned}$$

Mivel ε_i értékei csak a kérdéses altéren érdekesek számunkra, így ε_i -t tetszőlegesen lineárisan kiterjeszthetjük az összes lehetséges környezetekre, ezáltal egy $\varepsilon_i: A^{n_i} \rightarrow A$ lineáris függvényt kapunk. Ebből következik, hogy

$$\varepsilon_i(a_{i_1}, \dots, a_{i_{n_i}}) = h_{i,i_1} a_{i_1} + \dots + h_{i,i_{n_i}} a_{i_{n_i}}$$

alakban írható, és mivel ε_i inverz függvény, így az együtthatókból képezett h_i vektorra $h_i \cdot K = e_i$ teljesül.

($\Leftarrow$) A h_i vektor egy ε_i függvényt definiál, amely a $h_i \cdot K = e_i$ feltétel miatt a c_i sejtre vonatkozóan inverz lokális átmenetfüggvény. Q.E.D.

A következőkben h_i -ből és K -ból kiemeljük azokat az elemeket, amelyeken a $h_i \cdot K = e_i$ feltétel teljesülése mulik. A feltétel szerint h_i -ben csak az $N(c_i)$ -nek megfelelő elemek 0-tól különbözőek, ebből adódóan a K mátrixban

is csak az $N(c_i)$ -nek megfelelő sorok érdekesek. Belátható továbbá, hogy ezen sorokban csak az $N(N(c_i))$ elemeinek megfelelő oszlopok tartalmazhatnak nemnulla elemet, hisz K globális-átmenetfüggvénymátrix. Így a K mátrixból elég az $N(c_i)$ -nek megfelelő sorok és $N(N(c_i))$ -nek megfelelő oszlopok metszetét megtartani, ezt a részmátrixot Q -val jelöljük a továbbiakban. Jelölje továbbá p azt a vektort, amelyet h_i -ből az $N(c_i)$ -nek megfelelő elemek kiemelésével kapunk, e pedig legyen az e_i egységvektor $N(N(c_i))$ -nek megfelelő szűkítése. Ezáltal a $h_i \cdot K = e_i$ feltételi egyenlet egy $p \cdot Q = e$ egyenletre egyszerűsíthető /3.2. ábra/.

$$\begin{array}{c}
 \overbrace{\begin{array}{|c|c|c|} \hline 0 & p & 0 \\ \hline \end{array}}^{N(c_i)} \cdot \begin{array}{c} \overbrace{\begin{array}{|c|c|c|} \hline & 0 & \\ \hline 0 & Q & 0 \\ \hline & 0 & \\ \hline \end{array}}^{N(N(c_i))} \\ N(c_i) \left\{ \right. \\ K
 \end{array} = \begin{array}{c} \overbrace{\begin{array}{|c|c|c|} \hline 0 & e & 0 \\ \hline \end{array}}^{N(N(c_i))} \\ e_i
 \end{array}$$

3.2. ábra: A $h_i K = e_i$ és a $pQ = e$ feltételi egyenletek kapcsolata. A szemléletesség kedvéért feltételeztük, hogy a K mátrixban az $N(c_i)$ -nek megfelelő sorok és az $N(N(c_i))$ -nek megfelelő oszlopok egymás mellett /egy tömbben/ helyezkednek el.

A leírtak alapján világos, hogy a Q mátrix az invertálható függvénykombinációt képviseli /minden sora egy átmenetfüggvénynek felel meg/, a p vektor az inverz lokális átmenetfüggvényt adja meg, és a sejtautomata a c_i sejtben akkor és csak akkor invertálható, ha a $p \cdot Q = e$ feltétel teljesül. A következő pontban ismertetett algoritmus első fázisában tulajdonképpen ennek a feltételnek eleget tevő (p, Q) párokat generálunk.

6. A LINEÁRIS INVERZKONSTRUÁLÓ ALGORITMUS

A következőkben a 4. pontban vázolt inverzkonstruáló eljárást alkalmazzuk lineáris sejtautomaták esetére. A linearitás feltételezése csak az első fázisnál /invertálható függvénykombinációk generálása/ lényeges, itt az 5.2. pont eredményeire építünk. A második fázis /összerakás/ tárgyalása után röviden ismertetjük a kétfázisu eljárást realizáló INVGEN programot. A futási eredményekből jellemző példákat tartalmaz a Függelék.

6.1. Első fázis: invertálható függvénykombinációk generálása

Az algoritmus segítségével egy adott (C, N, A) sejttér valamely c_i sejtjére vonatkozó összes invertálható függvénykombináció előállítható, a megfelelő inverz függvényekkel együtt. Ha (C, N, A) szomszédságban homogén, akkor ezzel egy teljes függvénykombináció-készletet nyerünk, amellyel megkezdhetjük a sejttér kiparkettázását.

A szükséges bemenő adatok a következők:

(i) Az A véges gyűrű strukturája. Ez az s állapot-szám megadását és a gyűrűműveletek /általában a modulo- s műveletek/ specifikálását jelenti.

(ii) A c_i sejt "környékének" szomszédsági strukturája. Ehhez az $\bar{N}$ szomszédsági-reláció-mátrixnak /lásd

lenne, hogy végignézzük az összes lehetséges Q függvény-kombinációt, és mindegyikhez megpróbálunk p inverzet keresni. Mivel azonban a lehetséges Q mátrixok száma igen nagy, ezért a fordított eljárás célszerűbb: az összes lehetséges p vektoron haladunk végig, és mindegyikhez generáljuk az összes, a $p \cdot Q = e$ feltételnek eleget tevő Q mátrixot.

Legyen tehát p rögzített. Jelöljük q_j -vel a Q mátrix j -edik oszlopát, e_j -vel pedig az e vektor j -edik elemét. A $p \cdot Q = e$ feltétel ekkor úgy is írható, hogy minden j -re $(1 \leq j \leq \tilde{n})$ $p \cdot q_j = e_j$ kell hogy teljesüljön. Mivel a q_j oszlopban - az R mátrixnak megfelelően - általában csak néhány elem különbözhet 0-tól, így könnyen végignézzhetjük az összes lehetséges j -edik oszlopot, és kiválogathatjuk a $p \cdot q_j = e_j$ feltételnek eleget tevőket. Az így kapott q_j -k halmazát Q_j -vel jelöljük: $Q_j := \{q_j \mid p \cdot q_j = e_j\}$.

Az összes oszlop megvizsgálásával a $Q_1, \dots, Q_{\tilde{n}}$ oszlophalmazokat nyerjük. Ha most minden egyes oszlophalmazból egy tetszőleges oszlopot választunk, akkor az ezekből összeállított Q mátrixra $p \cdot Q = e$ teljesül. Az adott p -hez tartozó összes Q mátrixok halmaza így a $Q_1 \times \dots \times Q_{\tilde{n}}$ Descartes-szorozattal képezhető.

Összefoglalva, az invertálható függvénykombinációkat generáló algoritmus vázlata a következő:

1. p -nek valamilyen kezdőértéket adunk.
2. Képezzük a $p \cdot q_j = e_j$ feltételnek /és a szomszédsá-

gi-reláció-mátrixnak/ eleget tevő oszlopvektorok Q_j halmazát minden $1 \leq j \leq \tilde{n}$ -ra.

3. Generáljuk a $Q_1, \dots, Q_{\tilde{n}}$ oszlophalmazokból képezhető összes Q mátrixot.

4. Képezzük a következő p vektort, és 2.-vel folytatjuk az eljárást.

Ha végigmegyünk az összes lehetséges p vektoron, akkor a leírt algoritmus generálja egy adott (C, N, A) sejtter c_i sejtjéhez az összes létező (p, Q) párt. Ezek száma azonban rendszerint igen nagy, ugyanakkor viszonylag kevés lényegesen különböző van közöttük. Például legyen p egy egyszerű jobbra shiftelő függvény /Neumann szomszédság mellett/, ehhez s^{20} különböző Q mátrix generálódna! Ugyanis a jobbra shiftelő függvény minden olyan függvénykombinációnak inverze, amelyben a bal szomszéd balra shiftelő függvénnyel rendelkezik, a többi függvény pedig tetszőleges.

A fenti probléma az invertálható függvénykombinációkban közömbös függvények bevezetésével oldható fel, ennek általános elve a következő. Tegyük fel, hogy az inverz függvény p vektorának j -edik komponense 0. Ez azt jelenti, hogy a $p \cdot Q = e$ feltétel teljesülése szempontjából a Q mátrix j -edik sora közömbös, azaz a függvénykombinációban a j -edik átmenetfüggvény tetszőleges lehet. Nyilván felesleges, hogy a j -edik sor összes lehetséges értéke generálódjon. Ehelyett a j -edik sort egy speciális "közömbös függvény" jellel

töltjük ki, és kikapcsoljuk a megfelelő q_j oszlopvektorok keresésekor. A módszer segítségével elérhető, hogy a fenti toló függvénykombináció csak egyetlen példányban generálódjon.

A generált (p, Q) párok száma tovább csökkenthető, ha törölünk közülük minden olyan Q függvénykombinációt, amely valamilyen ok miatt nem illeszthető be egy teljes invertálható függvényrendszerbe. Ilyen ok lehet a Q mátrix sorainak lineáris függősége. Ekkor ugyanis akárhogyan egészítjük ki a Q mátrixot egy teljes K mátrixszá /3.2. ábra/, K sorai is szükségképpen lineárisan függőek lesznek. Ez azt jelenti, hogy K determinánsa zérusosztó kell hogy legyen, ami a 9. tétel alapján kizárja az invertálhatóságot.

6.2. Második fázis: a függvénykombinációk összerakása

Az összerakó eljárás során nem használjuk ki a függvénykombinációk lineáris voltát, így az algoritmus nemlineáris esetben is alkalmazható.

A szükséges bemenő adatok:

- (i) A sejttér szomszédsági strukturájának megadása. Ez egy (C, N, A) sejttér esetén C és N megadását jelenti.
- (ii) Invertálható függvénykombinációk egy /lehetőleg teljes/ készlete. Inhomogén szomszédsági struktúra esetén

minden különböző "környezetű" sejthez külön függvénykombináció-készlet kell hogy rendelkezésre álljon.

Az összerakó algoritmushoz néhány egyszerűsített jelölést kell bevezetnünk. A c_i sejthez megadott függvénykombinációkat $v_{i,1}, \dots, v_{i,r_i}$ -vel jelöljük, ahol a második indexet a függvénykombináció sorszámának nevezzük.

Az összerakás során az átmenetfüggvények - mint függvények - pontos ismerete nem szükséges, így az egyes függvények egyszerűen azonosítási számokkal jelölhetők. Egy

$$f_i(a_1, \dots, a_{n_i}) = k_1 a_1 + \dots + k_{n_i} a_{n_i}$$

lineáris átmenetfüggvény egyértelműen azonosítható a

$$z_i = a_1 + s \cdot a_2 + \dots + s^{n_i-1} a_{n_i}$$

számmal, ahol s az állapotszámot jelenti. Így egy $v_{i,k}$ invertálható függvénykombináció egy rendezett n_i -essel adható meg. A függvénykombinációban esetleg szereplő különbözős függvényeket -1 -el jelöljük.

Az összerakó algoritmus során két m elemű $/m$ a sejtek száma/ adattömböt töltünk ki párhuzamosan:[‡]

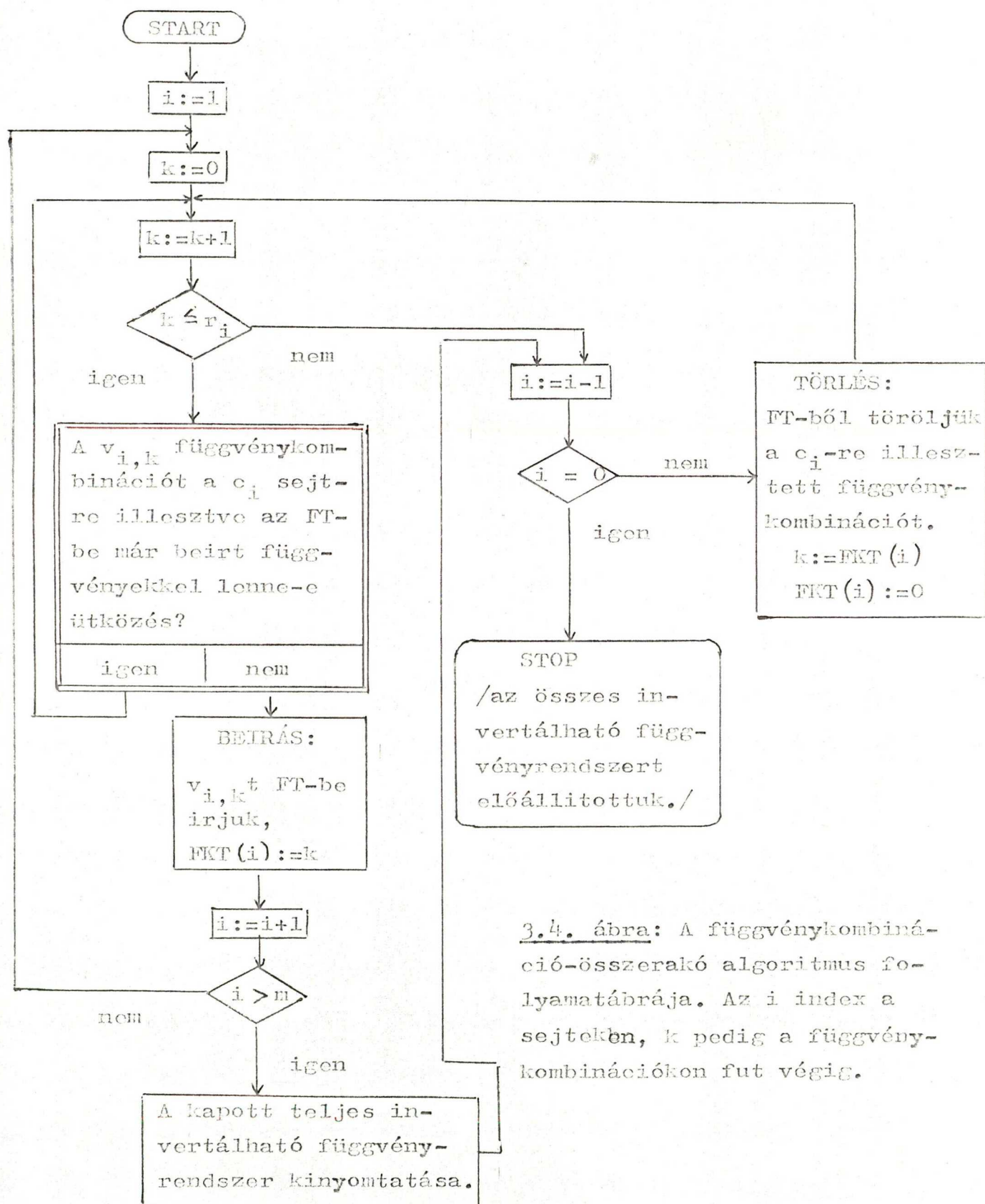
[‡] Megjegyezzük, hogy még egy harmadik, JEL nevű tömbre is szükség van. Ennek i -edik elemébe j -t írunk, ha a c_i sejthez akkor rendelünk átmenetfüggvényt, amikor a c_i sejtre illesztünk függvénykombinációt. A JEL tömbre a függvénykombinációk szabályszerű visszabontásához /lásd később/ van szükség, mivel a különbözős függvények miatt a visszabontás egyébként nem lenne egyértelmű.

- Az FT átmenetfüggvény-tömb i -edik eleme a c_i sejthez rendelt átmenetfüggvény azonosítási számát tartalmazza. Kezdetben FT minden eleme -1 /definiálatlan, azaz közömbös függvény/.

- Az FKT függvénykombináció-tömb i -edik elemébe a c_i sejtre illesztett $v_{i,k}$ függvénykombináció k sorszáma kerül. FKT elemeinek kezdőértéke 0.

Az összerakó algoritmus a $C = \{c_1, \dots, c_m\}$ sejthalmaznak valamilyen, például az indexezésnek megfelelő végigjárására épül. Először a c_1 sejtre illesztünk függvénykombinációt, majd a c_2 -re, és így tovább. Ha az építéssel a c_j sejtnél elakadunk, azaz a $v_{j,1}, \dots, v_{j,r_j}$ függvénykombinációk egyike sem illeszkedik a már beépített függvényekhez, akkor a c_{j-1} sejtre illesztett függvénykombinációt visszabontjuk, és más függvénykombináció beépítésével próbálkozunk. Az algoritmus vázlatos folyamatábrája a 3.4. ábrán látható. Az eljárás lényegéhez tartozik, hogy az FKT tömb $(k_1, \dots, k_m)$ kitöltéseit lexikografikusan növekvő sorrendben^{*} állítjuk elő, ami garantálja, hogy minden invertálható függvényrendszert pontosan egyszer rakunk össze.

^{*}Egy $(x_1, \dots, x_m)$ vektor lexikografikusan megelőzi az $(y_1, \dots, y_m)$ vektort, ha van olyan i , hogy $x_i < y_i$, és minden $j < i$ -re $x_j = y_j$.



3.4. ábra: A függvénykombináció-összerakó algoritmus folyamatábrája. Az i index a sejtekben, k pedig a függvénykombinációkon fut végig.

12. Állítás: A 3.4. ábra szerinti algoritmus a megadott függvénykombinációkból képezhető összes invertálható függvényrendszert előállítja, és minden függvényrendszer pontosan egyszer generálódik.

Bizonyítás /vázlat/: Először módosítsuk az algoritmust úgy, hogy a 3.4. ábrán kettős kerettel jelzett bloknál mindig a "nem" ágon haladunk tovább. Ekkor a függvénykombinációk "ütközéseit" figyelmen kívül hagyjuk, így a generált függvényrendszerek /FT vektorok/ értelmetlenek lesznek. Belátható viszont, hogy az algoritmus ekkor az FKT tömb összes lehetséges $(k_1, \dots, k_m)$ kitöltését előállítja lexikografikusan növekvő sorrendben.

Tekintsük most ismét az eredeti algoritmust! Ha a kettős keretű bloknál "igen" ágra futunk, ezzel a lexikografikus felsorolásból valahány FKT-vektort kihagytunk, ezek azonban szükségképpen mind átmenetfüggvény-ütközést tartalmaznak. Ezek szerint csak ütközést tartalmazó FKT-vektorok maradnak ki a felsorolásból, ami azt jelenti, hogy minden - az adott függvénykombinációkból előállítható - függvényrendszert generál az algoritmus. Q.E.D.

Megjegyezzük, hogy ha az összerakó algoritmus számára az összes invertálható függvénykombinációt megadjuk, akkor az algoritmus várhatóan igen sok függvényrendszert

fog előállítani. Ezek számának csökkentésére, az érdektelen változatok kiszűrésére két lehetőséget említünk:

(i) Csökkenteni kell a felhasználható átmenetfüggvények /és így a függvénykombinációk/ számát. Ez a megszorítás összhangban van a sejtprocesszorok korlátozott inhomogenitásával /korlátozott belső állapotszám/.

(ii) A sejtter egyes helyeire fix függvénykombinációkat kell elhelyezni, és az összerakó algoritmust a többi területre korlátozni. Ezzel elérhető, hogy az invertálható függvényrendszerben bizonyos függvények /ill. függvénykombinációk/ mindenképp szerepeljenek.

6.3. Az INVGEN program

Az INVGEN program az előző két pontban leírt inverzkonstruáló algoritmust realizálja. E pontban vázoljuk a program lehetőségeit és felépítését. Futási eredmények a függelékben találhatók.

A programmal az első fázisban tetszőleges, legfeljebb 10 szomszédos sejtterre generálhatók invertálható függvénykombinációk. Az R szomszédsági-reláció-mátrix legfeljebb 30 oszlopos lehet. Az s állapotszám szabadon választható /kizárólag megjelenítési okokból $s \leq 9$ /, a "+" és "." gyűműveletek azonban csak a modulo-s műveletek lehetnek. A generált függvénykombinációk nyomtatási formátuma sza-

badon választható, ezáltal különféle szemléletes megjelölések válnak lehetővé /lásd a Függelékét/.

A második fázisban téglalap alakú, homogén szomszédságu sejtteret feltételezünk, így csak egyetlen függvénykombináció-készlettel kell dolgoznunk. A szomszédsági index lényegében a Moore-szomszédság* tetszőleges megszorítása lehet. A szomszédság homogenitására a sejtteret bábsejtekkel vesszük körül, melyekhez identikus átmenetfüggvényt rendelünk. /Ez utóbbi a szélső aktív sejtekre illesztett függvénykombinációk miatt érdekes./ Belátható, hogy ezzel a feltételezéssel lefedjük a bábsejtek nélküli, a széleken korlátozott szomszédságu sejtterek esetét is, hisz ez utóbbiaknak olyan "bábsejtes" függvényrendszerek felelnek meg, amelyeknél a szélső sejtek függvényei nem függnek a bábsejtektől.

Az INVGEN program 12 FORTRAN nyelven írt szubrutinból áll, melyek egy alkalmanként írt FORTRAN főprogrammal aktivizálhatók. Ez a szervezési mód jelentős rugalmasságot biztosít a program használatában. Az alábbiakban áttekintjük a 12 szubrutint. A főprogramból lényegében csak az aláhúzással jelölt rutinok hívása szükséges, a többi "belső" rutinnak tekinthető.

1. BEOLV: adatbeolvasó rutin az első fázishoz. Beolvassa az R szomszédsági-reláció-mátrixot, a függvénykom-

*A Moore-szomszédságnál $N(c_{i,j}) = \{c_{u,v} \mid |u-i| \leq 1, |v-j| \leq 1\}$.

binációk nyomtatási formátumát, és - ha szükséges - bizonyos "geometriaai transzformációkat" /lásd SZÜRÖ/.

2. KOMBGEN: invertálható függvénykombinációkat generál. Az INVOLV, OVGEN, OVKOMB rutinokat hívja, a generáló algoritmusokat a két utóbbi tartalmazza. A szubrutin kétféle generálást tesz lehetővé: a/. az összes Q invertálható függvénykombináció generálását, b/. kártyán megadott $p_1, \dots, p_k$ inverz függvényekhez állítja elő az invertálható függvénykombinációkat.

3. INVOLV: kártyáról egy p inverz függvényt olvas.

4. OVGEN: egy adott p inverz függvényhez előállítja a $Q_1, \dots, Q_n$ oszlopvektor-halmazokat.

5. OVKOMB: a $Q_1, \dots, Q_n$ oszlopvektor-halmazokból invertálható függvénykombinációkat képez. Minden egyes függvénykombináció generálása után felhívja a SZÜRÖ rutint, amely a függvénykombinációt bizonyos előre megadott szempontok szerint megvizsgálja. Ha a vizsgálat a függvénykombinációt érdektelennek találja, akkor az nem kerül output-ra.

6. SZÜRÖ: Az OVKOMB rutin által aktuálisan generált függvénykombinációt vizsgálja. A lehetséges szempontok, amelyek egymástól függetlenül be- és kikapcsolhatók, a következők:

a/. Nullafüggvény figyelése. Ha a függvénykombináció azo-



nosan nulla függvényt tartalmaz, akkor nem illeszthető be invertálható függvényrendszerbe, mivel a függvényrendszer K mátrixa nulla-sort tartalmazna.

b/. A függvénykombináció sorainak lineáris függőségét vizsgáljuk. Ez lefedi az a/. vizsgálatot is, azonban annál lényegesen időigényesebb, ezért választottuk külön attól.

c/. Geometriai ekvivalensek felderítése. Ha az éppen generált függvénykombináció egy már korábban generáltnak - a sejttér geometriája szempontjából - tükrözöttje vagy elforgatottja, akkor érdektelennek minősítjük. A vizsgálathoz szükséges geometriai transzformációkat a BEOLV rutinnal kell beolvastatni. Maga a vizsgálat a korábban generált függvénykombinációk tárolása nélkül végezhető, így tárigénye jelentéktelen.

7. FEJIR: az invertálható függvénykombinációk listájához fejléceket nyomtat.

8. SEJTTER: a paraméterekben megadott méretű, téglalap alakú sejttérrel definiál. A sejttér maximális mérete 10.10, ez az invertálható függvényrendszerek vizsgálatához elegendőnek bizonyult.

9. KOMBOLV: Invertálható függvénykombinációkat olvas kártyáról vagy háttértárolóról. Az utóbbiról abban az esetben, ha a KOMBGEN rutinnal a generált függvénykombinációkat háttértárolóra irattuk.

10. FLXBEIR: a sejtter egy megadott helyére fix függvénykombinációt ír be, amely az összerakó algoritmus során nem változhat meg /lásd a 6.2. pont végét/.

11. ÉPIT: Invertálható függvényrendszereket épít a 3.4. folyamatábrának megfelelően a KOMBOLV által beolvasott függvénykombinációkból.

12. KIIR0: A generált függvényrendszerek /pontosabban az FT és FKT tömbök/ kinyomtatását végzi.

Az egyes szubrutinok használatát és pontos működését itt nem részletezzük.

2. szimulációs példa: Bináris szorzás: A szemléltetés kedvéért az adatcsatornákat szétválasztottuk. A jobbra shiftelő "A" adatcsatornát az összeadó feletti sejtsorban, a balra shiftelő "B" adatcsatornát pedig az összeadó alatti sejtsorban helyeztük el. A helykitöltő nullákat /a szorzandók előtt és után, valamint az egyes szorzandóbitok között/ "." karakterrel jelenítettük meg. A szimuláció a $13 \cdot 5 = 65$ szorzás elvégzését mutatja.

LEPES= 0, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. 1 . 1 . 0 . 1 . . . . .
    0 0 0 0 0 0 0 0
. . . . . 0 . 1 . 0 . 1

```

LEPES= 1, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. . 1 . 1 . 0 . 1 . . . . .
    0 0 0 0 0 0 0 0
. . . . . 0 . 1 . 0 . 1 .

```

LEPES= 2, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. . . 1 . 1 . 0 . 1 . . . . .
    0 0 0 0 0 0 0 0
. . . . . 0 . 1 . 0 . 1 . .

```

LEPES= 3, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. . . . 1 . 1 . 0 . 1 . . . . .
    0 0 0 0 0 1 0 0
. . . . . 0 . 1 . 0 . 1 . . .

```

LEPES= 4, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. . . . . 1 . 1 . 0 . 1 . . . .
          0 0 0 0 0 1 0 0
. . . . . 0 . 1 . 0 . 1 . . . .

```

LEPES= 5, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. . . . . 1 . 1 . 0 . 1 . . . .
          0 0 0 1 0 1 0 1
. . . . . 0 . 1 . 0 . 1 . . . .

```

LEPES= 6, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. . . . . 1 . 1 . 0 . 1 . . . .
          0 0 1 1 0 1 0 1
. . . . . 0 . 1 . 0 . 1 . . . .

```

LEPES= 7, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. . . . . 1 . 1 . 0 . 1 . . . .
          0 0 1 1 0 2 0 1
. . . . . 0 . 1 . 0 . 1 . . . .

```

LEPES= 8, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. . . . . 1 . 1 . 0 . 1 . . . .
          0 0 1 1 2 0 0 1
. . . . . 0 . 1 . 0 . 1 . . . .

```

LEPES= 9, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. . . . . 1 . 1 . 0 . . . .
          0 0 1 2 0 0 0 1
. . . . . 0 . 1 . 0 . 1 . . . .

```

LEPES= 10, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. . . . . 1 . 1 . 0 . . . .
          0 0 2 0 0 0 0 1
. . . . . 1 . 0 . 1 . . . .

```

LEPES= 11, MERET= 3 * 16, CSUCSOK=(2, 2)(4, 17),

```

. . . . . 1 . 1 . . . .
          0 1 0 0 0 0 0 1
. . . . . 1 . 0 . 1 . . . .

```



```
LEPES= 23, MERET= 12 * 30, CSUCSOK=( 2, 2)( 13, 31),***PR
```

[illegible]

LEPES= 28, MERET= 12 * 30, CSUCSOK=(2, 2)(13, 31),***PR

[illegible]

```
LEPES= 33, MERET= 12 * 30, CSUCSOK=( 2, 2)( 13, 31),***PR
```

[illegible]

LEPES= 38, MERET= 12 * 30, CSUCSOK=(2, 2)(13, 31),***PR

0	0	0	1	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	X
.	.	.	.	.	.	.	.	Y	.	X	.	Y	.	.	.	.	X	X	.	Y	.	X	Y	Y
0	0	1	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	.
.	.	.	.	.	.	.	.	.	X	X	.	Y	.	X	Y	.	X	.	Y	.	X	X	Y	X
0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	X	0	0	0	0	1	0	0	0
Y	.	X	.	.	.	.	.	.	X	.	X	.	X	X	Y	.	.	.	Y	.	X	.	Y	X

LEPES= 43, MERET= 12 * 30, CSUCSOK=(2, 2)(13, 31),***PR

0	0	0	1	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	1	1	0	0	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	X	Y	.	X	.	X
0	0	1	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	K
.	.	.	.	.	.	.	.	.	X	.	Y	.	.	.	.	.	X	.	Y	.	X	Y	.	X
0	1	0	0	0	0	0	1	0	0	1	0	2	0	0	0	0	0	1	1	0	0	1	0	L
.	.	.	.	.	.	.	.	.	X	X	.	Y	.	X	Y	.	.	Y	.	X	X	.	Y	X

LEPES= 48, MERET= 12 * 30, CSUCSOK=(2, 2)(13, 31),***PR

0	0	0	1	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	1	1	1	0	0	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	X	.	X	.	.	.
0	0	1	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	1	1	0	0	1	0	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	Y	.	X	.	X	.	X
0	1	0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	0	2	0	0	1	1	0	X
.	.	.	.	.	.	.	.	.	X	.	Y	.	.	.	.	.	Y	.	X	.	X	Y	.	X

LEPES= 53, MERET= 12 * 30, CSUCSOK=(2, 2)(13, 31),***PR

0 0 0 1 1 1 1 0	0 0 0 0 0 1 0 1	0 0 0 1 1 1 0 0	Λ
0 0 1 0 1 1 0 1	0 0 0 0 1 0 0 0	0 0 1 1 0 0 1 0	Λ
0 1 0 0 0 0 0 1	0 0 1 1 0 0 1 0	0 1 0 1 0 1 1 0	Λ
		X X	
		Λ Λ B Λ Λ	
		X Y X X X	

LEPES= 58, MERET= 12 * 30, CSUCSOK=(2, 2)(13, 31),***PR

0 0 0 1 1 1 1 0	0 0 0 0 0 1 0 1	0 0 0 1 1 1 0 0	
0 0 1 0 1 1 0 1	0 0 0 0 1 0 0 0	0 0 1 1 0 0 1 0	
0 1 0 0 0 0 0 1	0 0 1 1 0 0 1 0	0 1 0 1 0 1 1 0	Λ
		X X	

LEPES= 63, MERET= 12 * 30, CSUCSOK=(2, 2)(13, 31),***PR

0 0 0 1 1 1 1 0	0 0 0 0 0 1 0 1	0 0 0 1 1 1 0 0	
0 0 1 0 1 1 0 1	0 0 0 0 1 0 0 0	0 0 1 1 0 0 1 0	
0 1 0 0 0 0 0 1	0 0 1 1 0 0 1 0	0 1 0 1 0 1 1 0	

4. szimulációs példa: rendezés. Két rendezendő szám-
csoportot hajtunk át egy négysoros rendező sejttéren. Az
inputot az első lépésnél láthatjuk a sejttér fölé helye-
zett, lefelé shiftelő sejteken. A további lépésekben csak
a rendező sejttér került kiíratásra. Egy sejttállapotot
3 karakter reprezentál: az első két karakter az adatsa-
torna-biteket, a harmadik a vezérlőjelet jelenti. A sejt-
tér outputja a 23. lépésnél látható.

LEPES= 1, MERET= 30 * 6,

0	0	0	0	0	0
0	0	1	1	0	1
1	1	0	1	0	1
0	0	0	1	0	0
0	1	1	1	1	1
0	1	0	0	0	0
1	1	1	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	1	1	0	0	0
0	0	0	0	0	0
1	0	0	0	0	0
..H	..H	..H	..H	..H	..H
..H	..H	..H	..H	..H	..H
..H	..H	..H	..H	..H	..H
..H	..H	..H	..H	..H	..H

LEPES= 2, MERET= 30 * 6, LEPES= 8, MERET= 30 * 6,

1.M ..H ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H

.1H 11M ..H ..C ..C .1M
 ..H 1.M 11H 1.C 1.M ..H
 ..H ..H ..M ..H ..H ..H
 1.M ..H ..H ..H ..H ..H

LEPES= 3, MERET= 30 * 6, LEPES= 9, MERET= 30 * 6,

.1H ..M ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H

..H 11H 1.M 1.M .1C .1C
 1.M 11H .1M 1.C 1.C 1.M
 ..H ..H 1.M ..M ..H ..H
 .1H ..M ..H ..H ..H ..H

LEPES= 4, MERET= 30 * 6, LEPES= 10, MERET= 30 * 6,

..H 1.M 1.M ..H ..H ..H
 1.M ..H ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H

..H 1.C 1.C 11M ..M ..C
 .1H 11M ..H .1M 11C 11C
 ..H 1.M 11H ..M ..M ..H
 ..H ..H ..M ..H ..H ..H

LEPES= 5, MERET= 30 * 6, LEPES= 11, MERET= 30 * 6,

..H 1.C .1M 1.M ..H ..H
 .1H ..M ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H

1.M 11H 1.C 11C ..M 1.M
 ..H 1.C ..M 1.M .1M 1.C
 1.M 11H .1M 1.M 1.M 1.M
 ..H ..H 1.M ..M ..H ..H

LEPES= 6, MERET= 30 * 6, LEPES= 12, MERET= 30 * 6,

..H 1.C ..C .1M 1.M ..H
 ..H ..H 1.M ..H ..H ..H
 1.M ..H ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H

.1H .1M 11H 11C ..C 11M
 ..H 11H ..C 11M ..M .1M
 .1H .1M ..H .1M 11M .1M
 ..H 1.M 11H ..M ..M ..H

LEPES= 7, MERET= 30 * 6, LEPES= 13, MERET= 30 * 6,

1.M 11H .1C ..C .1M 1.M
 ..H ..H 1.C 1.M ..H ..H
 .1H ..M ..H ..H ..H ..H
 ..H ..H ..H ..H ..H ..H

..H ..H .1M 1.C ..C 1.C
 1.M 11H 1.M 11C ..M 1.M
 ..H 1.M ..M 1.M .1M 1.M
 1.M 11H .1M 1.M 1.M 1.M

LEPES= 14, MERET= 30 * 6,

```

..H ..H ..H .lM ..C l.C
.lH .lM llH .lM ..C .lM
..H llH ..M llM ..M .lM
.lH .lM ..H .lM llM .lM

```

LEPES= 20, MERET= 30 * 6,

```

..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H .lM
..H ..H ..H .lM ..M ..M

```

LEPES= 15, MERET= 30 * 6,

```

..H ..H ..H ..H ..M l.C
..H ..H .lM l.M ..M ..C
l.M llH l.M llM ..M l.M
..H l.M ..M l.M .lM l.M

```

LEPES= 21, MERET= 30 * 6,

```

..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..M l.M

```

LEPES= 16, MERET= 30 * 6,

```

..H ..H ..H ..H ..H .lM
..H ..H ..H .lM ..M ..M
.lH .lM llH .lM ..M .lM
..H llH ..M llM ..M .lM

```

LEPES= 22, MERET= 30 * 6,

```

..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H .lM

```

LEPES= 17, MERET= 30 * 6,

```

..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..M l.M
..H ..H .lM l.M ..M ..M
l.M llH l.M llM ..M l.M

```

LEPES= 23

```

..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H ..H
0 0 0 0 0 1
0 0 0 0 0 0
0 0 0 1 0 0
0 0 1 0 0 0
1 1 1 1 0 1
0 1 0 1 0 0
0 1 0 1 0 1
0 0 0 0 1 0
1 1 0 1 1 1
0 1 1 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
1 0 0 0 0 0

```

LEPES= 18, MERET= 30 * 6,

```

..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H .lM
..H ..H ..H .lM ..M ..M
.lH .lM llH .lM ..M .lM

```

LEPES= 19, MERET= 30 * 6,

```

..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..H ..H
..H ..H ..H ..H ..M l.M
..H ..H .lM l.M ..M ..M

```

2. FUTÁSI EREDMÉNYEK A III. FEJEZETHEZ

A következőkben kiragadott példákon próbáljuk szemléltetni az INGEN program használatát és lehetőségeit. A teljes futási listák eléggé terjedelmesek, így azok közlése itt nem lenne célszerű. A futtatásoknál 2 állapot, Neumann szomszédságu sejtteret definiáltunk.

1. lista: a lineáris inverz konstruáló algoritmus első fázisát mutatja be. Az átmenetfüggvények megjelenítése szemléletes formában történt: az

$$f(a_1, \dots, a_5) = k_1 a_1 + \dots + k_5 a_5$$

átmenetfüggvényt $\begin{matrix} & k_1 \\ k_2 & k_3 & k_4 \\ & k_5 \end{matrix}$ alakban ábrázoljuk, ahol k_1 a felső szomszéd együtthatója, k_2 a bal szomszédé, stb.

A mintapéldában az $\begin{matrix} 1 & & & 1 \\ 1 & 0 & 0 & \\ & 0 & & 0 \end{matrix}$ és az $\begin{matrix} 1 & & & 1 \\ 1 & 1 & 0 & \\ & 0 & & 0 \end{matrix}$

inverz függvényekhez állítottuk elő az összes invertálható függvénykombinációt. A generált függvénykombinációkból kiszelektáltuk a lineárisan függőket és a geometriailag ekvivalenseket /lásd 6.3. pont, SZÜRŐ szubrutin/. Ilyen feltételek mellett az első inverz függvényhez 1, a másodikhoz 10 "lényegesen különböző" invertálható függ-



vénykombináció adódott. /A 2-esekből álló számkombinációk közömbös függvényeket jelölnek./

Megjegyezzük, hogy a program futtatása során 2-állapotú Neumann szomszédságu sejttér esetén összesen 409 lényegesen különböző invertálható függvénykombinációt találtunk. Nagyobb állapotszám esetén ezek száma jóval több.

INVERZ = $\begin{matrix} & 1 \\ 1 & 0 & 0 \\ & 0 \end{matrix}$
 =====

$\begin{matrix} & 0 \\ & 1 & 0 & 0 \\ & 1 \end{matrix}$
 $\begin{matrix} 1 & & 2 & & 2 \\ 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & & 2 & & 2 \end{matrix}$
 $\begin{matrix} & 2 \\ & 2 & 2 & 2 \\ & 2 \end{matrix}$
 =====

A PROGRAM 1 FG.KOMBINACIOT GENERALT

```

      1
I N V E R Z   =   1 1 0
                    0
=====
      0
      1 1 0
      1
      1
      1
      1
      0 1 0   1 0 0   2 2 2   0 1 1   1 1 0   2 2 2
      0         0         2         0         0         2
      2
      2 2 2
      2
=====
      0
      1 1 0
      1
      1
      1
      1
      0 1 0   1 0 0   2 2 2   0 1 1   1 1 0   2 2 2
      0         0         2         0         0         2
      2
      2 2 2
      2
=====

```

A PROGRAM 10 FG.KOMBINACIOT GENERALT (MAXDB

A PROGRAM OSSZESEN 11 FG.KOMBINACIOT GENERALT

2. lista: az inverzkonstruáló algoritmus második fázisát mutatja be. Az összerakó program számára 13 invertálható függvénykombinációt adtunk meg. Az átmenetfüggvényeket tömörített formában, egyetlen számmal jelöltük /6.2. pont/, a

$$\begin{array}{cccc} & k_1 & & \\ k_2 & k_3 & k_4 & \text{átmenetfüggvény számkódja} \\ & k_5 & & \end{array}$$

$$z = 16 \cdot k_5 + 8 \cdot k_4 + 4 \cdot k_3 + 2 \cdot k_2 + k_1.$$

A közömbös függvényeket -1 jelöli.

A mintaprogram esetében 4•4-es sejtterre 12 különböző invertálható függvényrendszert raktunk össze /a megadott 13 függvénykombinációból ennél jóval több építhető/. A bal oldali mátrixok ábrázolják az átmenetfüggvények elrendezését /a sejtteret körülvevő bábsejtekkel együtt/, a jobb oldali mátrixok pedig az egyes sejtekre illesztett függvénykombinációk sorszámainak tartalmazzák.

1.	24	-1	-1	1	-1
2.	8	-1	-1	3	-1
3.	-1	-1	-1	18	8
4.	-1	-1	-1	16	9
5.	-1	16	-1	-1	3
6.	-1	24	-1	-1	2
7.	18	1	-1	-1	-1
8.	2	9	-1	-1	-1
9.	16	-1	-1	-1	-1
10.	-1	8	-1	-1	-1
11.	-1	-1	-1	2	-1
12.	-1	-1	-1	-1	1
13.	-1	-1	4	-1	-1

1.FGR:	4	4	4	4	4	4
	4	16	18	18	18	4
	4	8	8	8	1	4
	4	16	3	3	3	4
	4	8	8	8	1	4
	4	4	4	4	4	4

0	0	0	0	0	0
0	3	3	3	12	0
0	9	10	10	10	0
0	2	2	2	12	0
0	9	10	10	10	0
0	0	0	0	0	0

2.FGR:	4	4	4	4	4	4
	4	16	18	18	18	4
	4	8	8	8	1	4
	4	16	3	3	18	4
	4	8	8	8	1	4
	4	4	4	4	4	4

0	0	0	0	0	0
0	3	3	3	12	0
0	9	10	10	10	0
0	2	2	3	12	0
0	9	10	10	10	0
0	0	0	0	0	0

3.FGR:	4	4	4	4	4	4
	4	16	18	18	18	4
	4	8	8	8	1	4
	4	16	3	3	16	4
	4	8	8	9	1	4
	4	4	4	4	4	4

0	0	0	0	0	0
0	3	3	3	12	0
0	9	10	10	10	0
0	2	2	4	12	0
0	9	10	10	9	0
0	0	0	0	0	0

4.FGR:	4	4	4	4	4	4
	4	16	18	18	18	4
	4	8	8	8	1	4
	4	16	3	3	2	4
	4	8	8	9	1	4
	4	4	4	4	4	4

0	0	0	0	0	0
0	3	3	3	12	0
0	9	10	10	10	0
0	2	2	11	12	0
0	9	10	10	8	0
0	0	0	0	0	0

5.FGR:	4	4	4	4	4	4
	4	16	18	18	18	4
	4	8	8	8	1	4
	4	16	3	3	2	4
	4	8	8	8	1	4
	4	4	4	4	4	4

0	0	0	0	0	0
0	3	3	3	12	0
0	9	10	10	10	0
0	2	2	11	12	0
0	9	10	10	10	0
0	0	0	0	0	0

6.FGR:

4 4 4 4 4 4
4 16 18 18 18 4
4 8 8 8 1 4
4 16 3 3 18 4
4 8 8 1 1 4
4 4 4 4 4 4

0 0 0 0 0
0 3 3 3 12
0 9 10 10 10
0 2 2 12 12
0 9 10 10 7
0 0 0 0 0

7.FGR:

4 4 4 4 4 4
4 16 18 18 18 4
4 8 8 8 1 4
4 16 3 3 16 4
4 8 8 1 1 4
4 4 4 4 4 4

0 0 0 0 0
0 3 3 3 12
0 9 10 10 10
0 2 2 12 12
0 9 10 10 9
0 0 0 0 0

8.FGR:

4 4 4 4 4 4
4 16 18 18 18 4
4 8 8 8 1 4
4 16 3 3 4 4
4 8 8 1 4 4
4 4 4 4 4 4

0 0 0 0 0
0 3 3 3 12
0 9 10 10 10
0 2 2 12 13
0 9 10 10 13
0 0 0 0 0

9.FGR:

4 4 4 4 4 4
4 16 18 18 18 4
4 8 8 8 1 4
4 16 3 18 3 4
4 8 8 8 1 4
4 4 4 4 4 4

0 0 0 0 0
0 3 3 3 12
0 9 10 10 10
0 2 3 2 12
0 9 10 10 10
0 0 0 0 0

10.FGR:

4 4 4 4 4 4
4 16 18 18 18 4
4 8 8 8 1 4
4 16 3 18 18 4
4 8 8 8 1 4
4 4 4 4 4 4

0 0 0 0 0
0 3 3 3 12
0 9 10 10 10
0 2 3 3 12
0 9 10 10 10
0 0 0 0 0

11.FGR:

4 4 4 4 4 4
4 16 18 18 18 4
4 8 8 8 1 4
4 16 3 18 16 4
4 8 8 9 1 4
4 4 4 4 4 4

0 0 0 0 0
0 3 3 3 12
0 9 10 10 10
0 2 3 4 12
0 9 10 10 9
0 0 0 0 0

12.FGR:

4 4 4 4 4 4
4 16 18 18 18 4
4 8 8 8 1 4
4 16 3 18 2 4
4 8 8 9 1 4
4 4 4 4 4 4

0 0 0 0 0
0 3 3 3 12
0 9 10 10 10
0 2 3 11 12
0 9 10 10 8
0 0 0 0 0

IRODALOMJEGYZÉK

- Aladyev, V. /1980/: Mathematical theory of homogeneous structures. Valgus, Tallinn.
- Amoroso, S. - Cooper, G. - Patt Y. /1971/: Some clarifications of the concept of a garden-of-eden configuration. U.S. Army Electronics Command.
- Beyer, W. T. /1969/: Recognition of Topological Invariants by Iterative Arrays. Doktori disszertáció, MIT, Cambridge, Mass.
- Codd, E. F. /1968/: Cellular automata. Academic Press, Inc., New York.
- Dierk, U. /1977/: Hibavizsgálat inhomogén sejthálózatokban. A Braunschweigi Sejtautomata Találkozó kiadványa. Magyarul megjelent: Párhuzamos számítási rendszerek 1979/3, NJSzT, Budapest.
- Dióslaki F. /1979/: Sejtprocesszorok software eszközei. Diplomamunka, JATE, Szeged
- Drommerné Takács V. /szerk./: Sejtautomaták. Cikkgyűjtemény, Gondolat Kiadó, Budapest, 1978
- Fáy Gy. /1973/: Circuitry realization of Codd's cellular Automaton's Cell. I. Nemzetközi Számítástudományi Konferencia, Székesfehérvár

- Fáy Gy. /1978/: Cellular design principles: a case study of maximum selection in CODD-ICRA cellular space. Computational Linguistics and Computer Languages XI, pp. 73-124 és XII, pp. 165-231
- Jevreinov, Z. V. - Horosevszkij, V. G. /1978/: Odnorodnűe vűcsiszlitzelnűe szisztemű. Nauka, Novoszibirszk
- Katona E. /1978/: Local and global reversibility of finite inhomogeneous cellular automaton. Acta Cybernetica 3, pp. 287-292.
- Katona E. /1979/: Linear parallel maps of tessellation automata. Acta Cybernetica 4, pp. 263-268.
- Katona E. /1981a/: The application of cellprocessors in conventional data processing. A III, Magyar Számítástudományi Konferencia kiadványa, Budapest, 295-306. old.
- Katona E. /1981b/: Cellular algorithms for binary matrix operations. A CONPAR '81 konferencia kiadványa, Nürnberg, 203-216. old.
- Katona E. /1981c/: Sejtprocesszorok alkalmazása fixpontos vektor- és mátrixszorzásra. A Programozási Rendszerek '81 konferencia kiadványa, 263-280. old.
- Katona E. /1981d/: Sejtalgorithmusok. Válogatás az MTA Automataelméleti Kutatócsoportnál elért eredményekből. Párhuzamos számítási rendszerek 1981/5, NJSzT, Budapest.

- Katona E. /1982/: Binary addition and multiplication in cellular space. Acta Cybernetica, Szeged
- Kung H. T. /1980/: The structure of parallel algorithms. Advances in Computers 19., Academic Press, New York
- Legendi T. /1977a/: Cellprocessors in computer architecture. Computational Linguistics and Computer Languages, XI, pp. 147-167.
- Legendi T. /1977b/: Programming of cellular processors. Proceedings of the Braunschweig Cellular Meeting, June 2-3, pp. 53-66.
- Legendi T. /1979/: A 2D transition function definition language for a subsystem of the CELLAS cellular processor simulation language. Computational Linguistics and Computer Languages XIII, pp. 169-194.
- Legendi T. - Molnár G. - Székely L. /1979/: A CELLAS sejttérszimulációs processzor felhasználói kézikönyve. Kézirat.
- Maruoka, A. - Kimura, M. /1976/: Condition for injectivity of global maps for tessellation automata. Information and Control 32, pp. 158-162.
- Molnár G. /1979/: Sejtautomaták programozása és szimulációja. Diplomamunka, ELTE

Neumann J. /1966/: Theory of Self-Reproducing Automata.

Edited and completed by Arthur W. Burks.

Urbana: University of Illinois Press.

Nishio, H. /1975/: Real time sorting of binary numbers
by 1-dimensional cellular automaton.

Proceedings of the International Symposium on
Uniformly Structured Automata and Logic, Tokyo

Pálvölgyi L. /1978/: Sejtautomata átmenetfüggvények definiálásának módszerei és alkalmazásuk. Diplomamunka, ELTE

Richardson, D. /1972/: Tessellations with local transformations. J. Comput. System Sci. 6, pp. 373-388.

Smith, A. R. III /1969/: Cellular automata theory.

Technical Report no. 2, Stanford University,
Stanford, Cal.

Szabó Zs. /1979/: Modellezési feladatok megoldása sejttérben. Diplomamunka, ELTE

Toffoli, T. /1972/: On the computability of backward steps in cellular spaces. CNR, Inst. per le Appliziori del Calcolo, Roma.

Traub, J. F. /ed./: Complexity of Sequential and Parallel Numerical Algorithms. Academic Press, New York and London, 1973

Vollmar, R. /1979/: Algorithmen in Zellularautomaten.

B. G. Teubner, Stuttgart

Wunsch, G. /1977/: Zellulare Systeme. Akademie Verlag,
Berlin.

