tanulmányok

12/1973

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZET

MAKROPROCESSZOROK
PROGRAMOZÁSI NYELVEK

Szerkesztette és lektorálta:
Legendi Tamás
a Digitális Technika Osztály munkatársa

A Neumann János Számitógéptudományi Társaság és a Magyar Tudományos Akadémia Számitástechnikai és Automatizálási Kutató Intézetének közös kiadványa.

Tanulmányok 12/1973.

A kiadásért felelős:

Dr. Vámos Tibor

az

MTA Számitástechnikai és Automatizálási
Kutató Intézet
igazgatója

Készült: Országos Müszaki Könyvtár és Dokumentációs Központja házi
sokszorositójában F.v.: Janoch Gyula

TARTALOMJEGYZÉK

ELŐSZÓ

Az ebben a gyüjteményben megjelenő dolgozatok szorosan kapcsolódnak az NJSZT-SZTAKI helyi csoport által szervezett előadássorozathoz, mind témájukban, mind céljukban. A cél a modern számitástechnikai eszközök bemutatása és bevezetésük segitése /ez a második feladat sokkal nehezebb, de igen fontos/ és egyben az intézetünk osztályai közötti vertikális információáramlás kiszélesitése.

Témakör szerint a dolgozatok két főbb csoportra oszthatók. Az egyik csoportot a makroprocesszorokkal foglalkozó cikkek alkotják. A 70-es években a makroprocesszorok már a számitástechnikai kultura szerves részét alkotják, szélesebb körü ismertetésük és szövegkezelési, programelőkészitési, forditási, post-processzálási és egyéb célokra való használatba vételük igen fontos lenne. /Ugyanugy mint előzőleg a listakezelő nyelveké és technikáké/. Ehhez a válogatásban szereplő három dolgozat igen szerény segitséget nyujthat, teljesebb megoldást jelentene speciális makroprocesszorokról szóló cikkeket és egy átfogó ismertetést kiadni a makroprocesszorokról magyar nyelven.

A GPM látszólagos egyszerüsége és kötöttségei ellenére rendkivül hajlékony programozási eszköz. A CDC 3300-on implementált FORTRAN alapu processzort Tarján Mihály, a CII 10010-en ASTROL-ban implementált processzort Farkas Ernő ismerteti. A két egymást igen jól kiegészitő leirást több egyszerü és egy-egy bonyolultabb példa kiséri.

Egy speciális, elsősorban forditási célokra igen előnyösen használható, a szerző által kifejlesztett makroprocesszort ir le Farkas Ernő.

A dolgozatok másik főbb csoportja általános programozási nyelvekkel foglalkozik. A számitógépek interaktiv használatának fontossága egyre nő. A számitógéppel folytatott párbeszéd programozása történhet az egyes konkrét nyelveken belül /ennek az eljárásnak is megvannak a maga előnyei/ vagy önálló dialógusnyelvek segitségével. Ez utóbbi eljárás előnyeit demonstrálja a Krammer Gergely - Forgács Tamás által kifejlesztett és a CDC-3300-on,

valamint a CII 10010-en implementált DISTAR-B általános dialógusrendszer leirása. A dolgozat részletesen ismerteti mind a dialógusrendszer kidolgozásához vezető elveket /a szerzők által bevezetett dialógusprocesszor fogalom szemléletes volta különösen szerencsés/, mind az implementálás valamennyi lényeges főbb kérdését. Konkrét alkalmazási példák bemutatása és egy, a rendszer segitségével realizált dialógusnyelvre való hivatkozás teszi teljessé a dolgozatot.

Egy, a hagyományos programozási nyelvektől eltérő nyelv implementálásával foglalkozik Megyesi Katalin dolgozata. Ismerteti a nyelv egyszerüsitett szintaktikáju változatát, a leforditott programok belső ábrázolását /amelyet a zárt szubrutin-fogalom általánositott alkalmazása jellemez/ és a fordító-értelmező program strukturáját. A példákkal jól illusztrált dolgozat szerencsésen mutatja be a felmerülő problémákat és megoldásuk módját.

A Nagy Zsigmond és Nagy Judit által elkészitett és ismertetett forrásnyelvi FORTRAN nyomkövető program komoly hiányosságot pótol. A forrásnyelvi nyomkövetés a programok elkészitésének időtartamát jelentősen megröviditi és általában gépidő megtakaritást is eredményez. Ez a nyomkövető program viszonylag egyszerü funkciókat valósit meg, de ezek nélkülözhetetlenek és viszonylag olcsóak, mivel a forrásnyelvi szöveg előfeldolgozásával történik a nyomkövetés realizálása.

A programozási nyelvek szintaktikájáról és szemantikájáról szóló összefoglaló cikk zárja a gyüjtemény második fő részét. A cikk az ismert módszerek bemutatása mellett erőteljesen kiemeli a leglényegesebb kérdéseket és problémákat, Farkas Ernő eredeti szemléletmódjának segitségével.

A gyüjteményt egy rendhagyó dolgozat zárja. Rendhagyó Gáspár András agymodellje, mind témáját, mind az eredetiség mélységét tekintve. Ez az agymodell felfogható egy számitási rendszerként is - ennek elméleti vizsgálata, és esetleges későbbi realizálása komoly távlatokat nyithat. A dolgozatban leirt agymodellt a szerző a székesfehérvári számitástudományi konferencián /1973/ is előadta. Az ebből a modellből kiindulva megalkotott két további /itt ismertetésre nem kerülő/ modellel együtt a téma igen perspektivikus.

B u d a p e s t , 1973. szeptember 3.

Legendi Tamás

A GPM MAKROPROCESSZOR PROGRAM FORTRAN VÁLTOZATÁNAK HASZNÁLATA

Tarján Mihály

A makroprocesszorokat általában szövegek kompakt előállitására használják. Ezt az ismétlődő részek tárolásával és automatikus bemásolásával érik el. A GPM makroprocesszor az ismert hasonló programok között az egyik legjobban sikerült és legelterjedtebb. A FORTRAN megvalósitás megkönnyiti további használatát.

A program bemenete egy karakterlánc és kimenete egy másik karakterlánc. A program egyenként olvassa be a karaktereket és átmásolja azokat a kimenetre, kivéve ha makrohivás bukkan fel, ezt azonnal végrehajtja.

Egy makro "névből" és "testből" áll. Mindkettő egy-egy karaktersorozat. A testet alkotó karaktersorozatba ágyazva "formális paraméterek" fordulhatnak elő. Ezek

%n

alakuak, ahol n egy egyjegyü szám, ami azt jelenti, hogy kilenc különböző paramétert lehet alkalmazni egy makroban. A makro neve a nulladik paraméternek fogható fel és használható is. A makrotest maga is tartalmazhat makrohivásokat. A név és a test egymáshoz rendelését nevezzük makrodefiniciónak.

A makrogenerátor program a definiciókat eltárolja és a program futása végéig megőrzi azokat. A definiálás módjáról később lesz szó.

Egy előzőleg definiált makro hivható. A hivás egy speciális karaktersorozattal történik. Ez egy ∅ jellel kezdődik. Ezt a hivott makro neve követi, amit vessző zár le. A vessző után az aktuális paraméterek következnek, amelyek vesszővel elválasztott karakterláncok. Ezek tartalmazhatnak makrohivásokat is. Az utolsó aktuális paramétert | (vonás) zárja le.

Példa: /ALFA,BETA,/ GA/DELTA MMA

Itt ALFA ÉS DELTA makronevek, mig BETA és a GA/DELTA|MMA| karaktersorozatok aktuális paraméterek, a második éppen tartalmaz egy makrohivást.

A makrohivás végrehajtása annyit jelent, hogy a hivást reprezentáló karaktersorozat eltünik és a helyére másolódik a makro teste az aktuális paramétereknek a makrotestben a formális paraméter előfordulások helyébe való beirásával együtt. A végrehajtást a hivó sorozat végét jelentő vonás beolvasása után kezdi meg a program. A hivások tetszőleges mélységig skatulyázhatók. Makrohivás lehet akár makrotest, akár makrohivás bármely elemének része.

Ha a bemenő karaktersorozatban vagy egy másolás alatt álló makrotestben ⟶ jel fordul elő, a makrohivások a továbbiakban nem hajtódnak végre addig amig a hozzátartozó ⟵ jel meg nem jelenik, azaz a karaktersorozat egyszerüen lemásolódik. Mivel a lemásolt karaktersorozatban ujabb ⟶ jelek fordulhatnak elő a hozzájuk tartozó ⟵ jelekkel együtt, igy ezeknek az u.n. idéző jeleknek a megfelelő skatulyázását a program ellenőrzi és az esetleges hibát jelzi. A külső idézőjelpárt a program eltávolitja.

A makrodefinició a speciális DEF makro segitségével történik. Ez a makro be van épitve a makrogenerátorprogramba. Hivásakor első aktuális paraméterként kell megadni a makro nevét, második aktuális paraméterként a testet. Mig normálisan a makrohivás végrehajtása után a hivó sorozat (dollártól a vonalig) elvész, a DEF makro hatása a két aktuális paraméter, a makro név-test párjának eltárolása. A makro testben célszerü idézést alkalmazni, hogy az esetleges előforduló vesszőket ne paramétervégnek értelmezze a program.

Ha kétszer azonos névvel de különböző testtel definiálunk makrot, mindig a később végrehajtott definició lesz csak érvényes és a korábbi definició többé nem hozzáférhető, kivéve egy esetet. Ez a lokális érvényü makrodefinició. Ha egy makrohivás paramétereiben valahol egy DEF makro-hivás fordul elő, ez természetesen végrehajtódik még mielőtt az őt tartalmazó hivás végrehajtódna. A külső hivás végrehajtásának befejeződése pillanatában azonban a hivó sorozat törlésével együtt ez a definició is törlődik.

Pl. /A,XX/DEF,B,ZZZ |,YY|

Vegyük észre, hogy az A makro első paramétere csak XX mivel a DEF hivása semmilyen eredményt nem ad a definición kivül. Itt tehát a B definiciója csak az A hivásának a végéig érvényes.

A makrogenerátor programban a DEF makrohoz hasonlóan beépitett más speciális makrók is vannak: UPDA makrotest kicserélő
BIN decimális-bináris konverzió
DEC bináris-decimális konverzió
VAL a test eredeti alakját
BAR bináris aritmetikát

adó makrók.

Az UPDA kicseréli az első paraméterként adott makronév testét a második paraméterben adott testre. Az uj test nem állhat több karakterből mint a régi. A BIN makro az első paraméterben adott egész számot konvertálja binárissá és elhelyezi a hivás helyére. A DEC hatása a BIN hatásának ellentettje. Mind kettő egyparaméteres. A VAL makro az első paraméterben adott makronévhez tartozó testet adja eredményként. A BAR makro három paraméteres. Az elsőnek a négy alapmüveleti jel (+,-,x,/)egyikének kell lennie. A második és harmadik paraméternek bináris számnak kell lennie. A BAR makro hivásának eredménye az elvégzett müvelet eredményeként kapott bináris szám, ami a hivás helyére kerül.

A program a makrodefiníciókat a futás végéig őrzi. Egy futás végét a bemenő karaktersorozatban önmagában előforduló ¬ jel jelzi. Ekkor a program elfelejti az eddigi makrodefiníciókat és tovább müködik. Ha az ekkor beolvasott első karakter ismét egy ¬ , akkor a program leáll.

A GPM program alkalmazható egyszerü vagy bonyolultabb feladatok megoldására. Néhány tipikus alkalmazási példát ismertetünk az alábbiakban.

Egyszerü makrodefiníció

ǾDEF,A,XXXXX|

ǾDEF,B,YYǾA|YYY| ekvivalens ǾDEF,B,YYXXXXXYY -nal.

Makrohivás eredménye uj makro definiciója

ǾDEF,A,XX%1¬ ǾDEF,A,ǾA,%1|¬ %1¬|¬|

Az első ǾA,Y| hiváskor az eredmény XXY lesz. E mellett az uj definició XXY%1 lesz. A második ǾA,Z| hivás eredménye XXYZ lesz.

Feltételes kifejezés megvalósitása

Ennek alapesete a következő

$$\alpha = \beta - \rightarrow \gamma, \delta$$

α, β, γ és δ karaktersorozatok melyekben esetleg makrohivások is előfordulhatnak. A kifejezés értelme: ha α azonos β -val, az eredmény legyen γ, egyébként legyen δ. Ez igy valósitható meg:

$DEF, \alpha, \beta|$

$\$\alpha, \$DEF, \beta, \gamma\|$

A fenti módszer segitségével egy karaktersorozatot lehet rekurzive definiálni.

Pl. Tudjuk, hogy n! = n* (n-1)...1

FACT(1) = 1

FACT(n) = n*FACT(n-1) = n*(n-1) * (n-2) * ... *1

Tehát FACT(α) $\begin{cases} 1 \text{ ha } \alpha = 1 \\ \alpha * FACT(n-1) \text{ ha } \alpha \neq 1 \end{cases}$

Itt α %1

β 1

γ 1

δ %1 *$FACT,$MIN,%1,1|

$DEF,FACT,¬$DEF,%1,%1*¬$FACT,¬$MIN,%1|¬|¬| $%1,$DEF,1,1‖¬|

A MIN makro egy könnyen definiálható makro, mely elvégzi a kivonást az első paraméter és a második között, megvalósitva az n-1 -gyet.

$DEF,MIN,—$DEC,$BAR,—,$BIN,%1| ,$BIN,1‖—¬|

Látható, hogy a bonyolult alkalmazás azt a veszélyt rejti magában, hogy a felhasználó "a saját eszén is tuljár", ezért óvatosan kell vele eljárni.

A program bináris deck formájában áll rendelkezésre a CDC 3300-on. Igényel 39 CORE-t és az input és az output file-okat.

A dsi-k: 3- az input másolata (hibakereséshez alkalmazható)
5- az input file
6- OUT file
7- PUN file

A 6 és 7 csak printer control karakterben tér el egymástól. A felhasználó a 6 és 7 dsi-ket kivánság szerint használhatja *DEF vagy FILE kártyák segitségével.

A program a hibátlan használat végén STOP 77-re áll le.

A hibaüzenetek:

ISMERETLEN MAKRONÉV
BIN PARAMÉTERE NEM SZÁM
HIÁNYZIK EGY AKTUÁLIS PARAMÉTER
UPDA UJ MAKROTEST TUL HOSSZU
BAR MÜVELET NEM SZABÁLYOS
PROGRAM VAGY GÉPHIBA

A programmal kapcsolatosan felvilágositást ad Licskó Ildikó vagy Tarján Mihály.

Irodalom: C. STRACHEY.: A General Purpose Macrogenerator Computer Journal Vol 8. pp 225-241.

A GPM /10010/ MAKROPROCESSZOR

Farkas Ernő

A GPM általános célu, kötött formátumu, karaktermanipulációs makroprocesszszor.
Ez részletesebben a következőket jelenti:

Karaktermanipulációs:

A forrásszöveget karakterenként olvassa be, a tárgyszöveget karakterenként adja ki. Bizonyos karakterek az u.n. figyelmeztető karakterek hatására különböző tevékenységeket hajt végre.

Kötött formátumu:

A makrók definiálását, hivását és minden más tevékenységet az adott figyelmeztető karakterek segitségével szigoruan kötött formában adhatjuk csak meg.

Általános célu:

A forrásnyelvvel szemben egyetlen kikötést teszünk: ASCII kódban legyen irva, és a @ . & . <, > a jeleket csak makroprocesszor vezérlésére használhatjuk.

Ha a makrókat egy nyelv kiterjesztésére kivánjuk felhasználni, akkor egy makró állhat több utasitás helyett, egy utasitás helyett, vagy az utasitás bármely része (pl. operandus müveleti kód stb.) helyett.

A GPM /10010/ szintaktikája és szemantikája

A GPM makrói ugynevezett text makrók:

a program a forrásnyelvi szöveget változatlanul lemásolja, kivéve, ha makró definiciót, vagy makró hivást talál. A makró definició két elemet tartalmaz, a makró nevét és az ugynevezett kódtörzset.

A processzor a makró definició felismerése után a makró definiciót az u.n. definiciós táblában raktározza el és a forrásszöveget a definició kihagyásával másolja le. A makróhivás felismerésekor a processzor a makróhivást a megfelelő kódtörzzsel helyettesiti és a forrásnyelvi szöveget igy másolja le. A makrótörzsben bizonyos helyeket paraméterek számára tartunk fenn, amelyeknek aktuális értékét a hivásban definiáljuk.

A makróhivás formája:

&név, p1, p2,...pn;

Tehát a makró egy "&" jellel kezdődik és ";" zárja be. A "&" jel után következik a makró neve, amely egyben a 0. paraméter is. Ha több paraméter van, akkor ezeket ","-vel választjuk el egymástól. Összesen 9 db paraméter lehet a néven kivül, amelyeket 1-től 9-ig sorszámozunk. Paraméter bármilyen karaktersorozat lehet.

A makródefinició formája:

&DEF, név, kódtörzs;

A DEF név egy speciális rendszermakró neve, melynek két paramétere van, az első a definiált makró neve, a második pedig a kódtörzse.

A kódtörzsben a paraméterek helyét "'n" karakterekkel jelöljük meg, ahol
n a paraméter sorszámát jelenti.

Egyéb vezérlő karakterek:

A "<" és ">" jeleket idézőjeleknek nevezzük, mert a közéjük tett szöveget a makróprocesszor vizsgálat nélkül másolja le, magát a "<" és ">" jelet pedig elhagyja.

"@" a forrásnyelvi adatszalag végét jelző karakter. A processzor munkája ilyenkor még nem feltétlenül ért véget, hanem a munkát a következő szalaggal folytathatjuk, miközben az előző szalag által definiált makrók továbbra is érvényben maradnak. A "@" jel csak egy lezárt egység végén alkalmazható, és hatására ellenőrzésre kerül az, hogy nincs-e bezáratlan idézőjel, vagy befejezetlen makró. Emiatt a "@" karaktert szintaktikus hibák ellenőrzésére is használhatjuk. A kocsivissza és soremelés karakterek nem szerepelnek a makróprocesszor által különlegesen kezelt karakterek között, hanem ugy tekinti őket, mint pl. bármely betüt vagy számot.

A bonyolultabb makró kifejezések kiértékelési módja:

A GPM /10010/ lehetővé teszi, hogy egy makró belsejének /az "&" jel és a ";" között/ bármelyik része helyett egy másik makró álljon.

A GPM /10010/ müködésének pontos szabályait a következőkben foglalhatjuk össze:

1., A forrásszöveget balról-jobbra vizsgáljuk és változatlanul másoljuk át outputra, amig "&" jelet nem találunk.

2., A makró felismerését és a szükséges helyettesitések elvégzését a makró kiértékelésének nevezzük. A kiértékelés balról-jobbra haladó rekurziv eljárás.

3., Amint egy makrónak a paraméterlistája teljes lett, azaz megtaláltuk a hozzátartozó ";"-t a makró kiértékelését azonnal teljesen elvégezzük. Ez egyben azt is jelenti, hogy először mindig a legbelső makrót értékeljük ki és a belőle keletkezett szöveg a külső makró egy vagy több paraméterének lesz része. Az igy létrehozott szöveg azonban soha nem fejezhet be egy külsőbb makrót, és nem lehet egy ujabb makró kezdete sem, annak ellenére, hogy ";" vagy "&" jelet tartalmaz.

4., A ";" után megvizsgáljuk, hogy az adott pillanatban érvényes definiciók között van-e ilyen nevü makró. A makródefiniciókat forditott kronológikus sorrendben vizsgáljuk, azaz, ha több azonos nevü definició van, akkor ezek közül a legutolsót tekintjük érvényesnek. Ha ilyen definició nincs, hibajelzést adunk.

5., Ha egy DEF makró belsejében van valamely másik makró hivásnak vagy makró definiciónak, akkor a definició csak az őt közvetlenül magába foglaló külső makrón belül érvényes, és a definició törlődik mihelyt megtaláljuk a külső makró paraméter listájának végét.

6., Ha a makró definicióját megtaláltuk, akkor a kódtörzs bemásolásra kerül. A kódtörzs bemásolásakor is találhatunk makróhivásokat és makródefiniciókat. Ezek ugy maradhattak benne, hogy a definiáláskor a kódtörzs egy részét idézőjelbe tettük. A kódtörzs bemásolására ugyanazon szabályok vonatkoznak, mint ha az inputról kaptuk volna a jelsorozatot, eltekintve a következő pontban foglaltaktól.
Lehetséges tehát, hogy egy makró hivásának hatására egy uj makródefinició is létrejön, amely véglegesen meg is marad, ezt a makró mellékhatásának nevezzük.

7., Ha a kódtörzs bemásolása közben "'n" jeleket találunk, akkor helyébe az n-ik paramétert másoljuk be. A paramétert minden további vizsgálat nélkül másoljuk. Ha nem volt megfelelő paraméter, hibajelzést adunk.

A GPM /10010/ technikai adatai:

A GPM /10010/ makróprocesszor a szalag elején levő autochargeur programmal olvasható be és minimálisan 4K memóriát foglal el. Ebből maga a program kb. 2K a többi munkamemória. A program további helyfoglalását a makródefiniciók száma és terjedelme határozza meg. Ha a makródefiniciók terjedelme 2 oldalnál kisebb, ugy futtatásához a 4K memória elegendő.

A program a forrásnyelvi szöveget gyorsolvasóból olvassa be és gyorslyukasztót használ outputra. A hibaüzeneteket az irógépen irja ki. Öt különböző hibajelzés lehetséges:

- hivatkozás nem létező makróra,
- hivatkozás nem létező paraméterre,
- több végidézőjel, mint kezdő,
- szalag végén bezáratlan idézőjel,
- szalag végén befejezetlen makró.

A GPM /10010/ célja és felhasználásának lehetőségei:

C.Strachey-nek a GPM megalkotójának az elsődleges célja makróprocesszorának létrehozásával az volt, hogy az eredeti makrófogalmat kiszélesitse a következő értelemben:

A makró fogalma eredetileg az assemblerek köréből származik, ahol arra szolgált, hogy egy általunk definiált utasitásról, az eredeti nyelv több utasitásból álló sorozatát helyettesithessük. A legtöbb makróprocesszor továbbra is ezt a célt szolgálta, egy utasitást az általa reprezentált több utasitásra bont fel. Ezt a gondolatot a GPM olyan irányba fejlesztette tovább, amely lehetővé teszi, hogy a nyelv alapvető egységeiből felépitett bármely nagyobb összefüggő kifejezést egyetlen makróba fogjunk össze. Ez nem csak programirói szempontból jelenthet megtakaritást, de lehetőséget nyujt arra is, hogy uj szintaktikus kategóriákat alkossunk pl. változó tipusokat, müveleti jeleket stb. Tehát nyelvek kiterjesztésére nyujt lehetőséget, bár erősen kötött formában, de a nyelv szabályaitól függetlenül. Ebben az irányban mindezideig ez volt az egyik legsikeresebb kisérlet.

A GPM /10010/ felhasználásának lehetőségeit a következőkben foglalhatjuk össze:

- A programirás megkönnyitése bizonyos bonyolult és sokszor ismétlődő kifejezések makrósitásával.
- Nyelvek kiterjesztése uj szintaktikus kategóriákkal.
- Szövegszerkesztési feladatok. A GPM által nyujtott változatos lehetőségeket lényegében itt lehet jól kihasználni. Lehetőség van arra például,

hogy a beadott makró ne csak a paraméterei által definiált szöveget hozza létre, hanem módositsa velük a meglévő definiciókat is.

- Adott mikroprogram készlethez illeszkedő interpretiv nyelv létrehozása. Ha adott egy mikróprogram készlet, amelyhez különböző interpretiv programokat kivánunk irni, akkor elegendő egyetlen egyszer elkésziteni a makrókészletet, amelyben egy makró egy mikróprogramnak, vagy mikróprogram sorozatnak megfelelő interpretiv programot hoz létre. És ekkor a programokat már csak makrók sorozatával kell leirni. Az utóbbi két alkalmazásra a mellékletben látunk példát.

1. Példa:

```
&DEF,A, < ELMENTEM EN A VASARBA
FELPENZZEL
'1 VETTEM A VASARBAN
FELPANZZEL
'2'3
SEJ!
MEGIS VAN EGY FELPENZEM
 >;
&DEF,VETTEM, <&A, ' 1,,'2; DEF,A, &A, <' 1 > , <'2'3
 >, '2;;> ;
&VETTEM,TYUKOT,A TYUKOM MONDJA KOTTY;

&VETTEM,CSIRKET,CSIRKEM MONDJA CSIP-CSIP-CSIP;
 VETTEM,LIBAT,LIBAM MONDJA GI-GA-GA;
&VETTEM,BIRKAT,BIRKAM MONDJA BE-E-E;
&VETTEM,KECSKET,KECSKE MONDJA MEK-MEK-MEK;
&VETTEM,LOVAT,LOVAM MONDJA NYI-HA-HA; @
```

Definiciók

Hivások

```
ELMENTEM EN A VASARBA
FELPENZZEL
TYUKOT VETTEM A VASARBAN
FELPANZZEL
A TYUKOM MONDJA KOTTY
SEJ!
MEGIS VAN EGY FELPENZEM

ELMENTEM EN A VASARBA
FELPENZZEL
CSIRKET VETTEM A VASARBAN
FELPANZZEL
CSIRKEM MONDJA CSIP-CSIP-CSIP
A TYUKOM MONDJA KOTTY
SEJ!
MEGIS VAN EGY FELPENZEM

ELMENTEM EN A VASARBA
FELPENZZEL
LIBAT VETTEM A VASARBAN
FELPANZZEL
LIBAM MONDJA GI-GA-GA
CSIRKEM MONDJA CSIP-CSIP-CSIP
A TYUKOM MONDJA KOTTY
SEJ!
MEGIS VAN EGY FELPENZEM
```

Eredmény

```
ELMENTEM EN A VASARBA
FELPENZZEL
BIRKAT VETTEM A VASARBAN
FELPANZZEL
BIRKAM MONDJA BE-E-E
LIBAM MONDJA GI-GA-GA
CSIRKEM MONDJA CSIP-CSIP-CSIP
A TYUKOM MONDJA KOTTY
SEJ!
MEGIS VAN EGY FELPENZEM

ELMENTEM EN A VASARBA
FELPENZZEL
KECSKET VETTEM A VASARBAN
FELPANZZEL
KECSKE MONDJA MEK-MEK-MEK
BIRKAM MONDJA BE-E-E
LIBAM MONDJA GI-GA-GA
CSIRKEM MONDJA CSIP-CSIP-CSIP
A TYUKOM MONDJA KOTTY
SEJ!
MEGIS VAN EGY FELPENZEM

ELMENTEM EN A VASARBA
FELPENZZEL
LOVAT VETTEM A VÁSÁRBAN
FELPANZZEL
LOVAM MONDJA NYI-HA-HA
KECSKE MONDJA MEK-MEK-MEK
BIRKAM MONDJA BE-E-E
LIBAM MONDJA GI-GA-GA
CSIRKEM MONDJA CSIP-CSIP-CSIP
A TYUKOM MONDJA KOTTY
SEJ!
MEGIS VAN EGY FELPENZEM
```

2. Példa:

```
&DEF,ADD,ADS OSSZ
      ADS 1;

&DEF,SUB,ADS NEG
      ADS OSSZ
      ADS '1
      ADS NEG;
&DEF,MULTI,ADS SZORZ
      ADS  '1;
&DEF,DIV,ADS OSZT
      ADS  '1;
&DEF,STOP,ADS VEGE;
&DEF,LOAD,ADS .314
      ADS  '1;
&DEF,STORE,ADS .518
      ADS  '1;
&DEF,IF,ADS IF
      ADS  '1
      ADS  '2
      ADS  '3:
```

Definiciók

```
&DEF,CHN, &LOAD, <'1 > ;
      &STORE,MUNKA;
      &LOAD, <'2 > ;
      &STORE, <'1 > ;
      &LOAD,MUNKA;
      &STORE, <'2 >  ;;@
      &LOAD,EGY;
      &STORE,XNM1;
  C   &LOAD,X;
      &DIV,XNM1;
      &ADD,XNM1;
      &DIV,KETTO;
      &STORE,XN;
      &SUB,XNM1;
      &IF,A,B,A;
  A   &CHN,XN,XNM1;
      &IF,C,C,C;
  B   &STOP,;
```

Hivások

```
        ADS .314
        ADS EGY
        ADS .518
        ADS XNM1
  C     ADS .314
        ADS X
        ADS OSZT
        ADS XNM1
        ADS OSSZ
        ADS XNM1
        ADS OSZT
        ADS KETTO
        ADS .518
        ADS XN
        ADS NEG
        ADS OSSZ
        ADS XNM1
        ADS NEG
        ADS IF
        ADS A
        ADS B
        ADS A
A       ADS .314
        ADS XN
        ADS .518
        ADS MUNKA
        ADS .314
        ADS XNM1
        ADS .518
        ADS XN
        ADS .314
        ADS MUNKA
        ADS .518
        ADS XNM1
        ADS IF
        ADS C
        ADS C
        ADS C
B       ADS VEGE
```

Eredmény

AZ MP/O MAKROPROCESSZOR

FARKAS ERNŐ

AZ MP/O FILOZÓFIÁJA

Az utóbbi évek során szerte a világban sok különböző célu és müködésü makroprocesszor született. A makroprocesszorok köre legalább annyira változatos, mint a programozási nyelveké, vannak közöttük nagyon rövid és egyszerü programok, és a magas szintü nyelvek fordítóprogramjaival összemérhető méretü és bonyolultságu programok is. A makroprocesszorok köréről jó áttekintést nyujt P.S. Brown: "Survey of Macro Processors" cimü cikke /1969 Annu-al Review in Automatic Programing Vol 6. Part 2./. Az ott felsorolt ismérvek alapján az MP/O makroprocesszort a következőképpen jellemezhetjük.

Az MP/O makroprocesszor text makrókat dolgoz fel, azaz inputját egy olyan szövegnek tekinti, amely bizonyos röviditéseket /makró hivásokat/ tartalmaz, és ezeket a megfelelő teljes szöveggel /az aktuális paraméterekkel behelyettesitett makró törzzsel/ kell helyettesitenie.

A processzor rekordmanipulációs, azaz a bejövő szöveget soronként vizsgálja és ha szükséges 1 sort valahány /esetleg nulla/ sorral helyettesit.

Az MP/O makroprocesszor nyelvtől független, azaz a bemenő szövegnek semmilyen szabálynak nem kell eleget tennie, azonkivül, hogy a feldolgozás szempontjából összetartozó részeknek egy sorba kell kerülniük.

Az MP/O makroprocesszor a sorokat makró hivást jelentő mintákkal hasonlitja össze. A minta kulcsszavakból és paraméterekből áll. Ezt az eljárást mintaillesztésnek hivjuk.

Az irodalom több olyan makroprocesszort ismer, amelyik a fenti ismérveket kielégiti a két legismertebb az ML/I és a STAGE 2. Mi ujat nyujt ezekhez képest az MP/O makroprocesszor?

A makróprocesszorok határterületen állnak a felhasználói és a készitői software között és igy az eddig készitett processzorok arca kettős, de inkább a felhasználói software-re jellemzően a nagymértékben automatikus, de azért lehetőleg hatékony müködésre törekszenek. Ezért használatuk rendkivül egyszerü, müködésük nehezen befolyásolható.

Az MP/O kifejezetten software fejlesztés céljára készült program: elsősorban nyelvek kiterjesztésére, egyszerübb nyelvek forditására, rendszergenerálásra és hasonló célokra készült.

Készitését az a törekvés hatotta át, hogy az a felhasználója, aki a makrókat definiálja, lehetőleg minél tökéletesebben irányithassa a processzor müködésének minden lépését, formailag esetleg kissé nehézkesen. Ugyanakkor az a felhasználó, aki az igy definiált makró készletet felhasználja, a makró hivásokat a lehető legkényelmesebben, minden formai kötöttség nélkül használhassa.

HOGYAN FORDITHATUNK MP/O-VAL?

A magas szintü programozási nyelvekre az a jellemző, hogy a végrehajtandó tennivalókat nagyon rövid és tömör formában közlik. Forditás alatt itt azt értjük, hogy ezeket a rövid és tömör jelöléseket egy géphez közelebb álló /rendszerint assembler/ nyelven részletesebben leirjuk.

A programozási nyelvek többsége utasitásokból áll, és ezeknek az utasitásoknak megszabott, egymástól formailag jól elválasztható alakjuk van. Az egyes utasitás tipusokat rendszerint az különbözteti meg egymástól, hogy bennük milyen kulcsszavak, milyen sorrendben szerepelnek.

Az utasitás igy definiált tipusa eleve meghatározza bizonyos részeit annak a szövegnek, amellyel a forditás során az utasitást helyettesiteni kell. A tipus ezen kivül azt is meghatározza, hogy a két kulcsszó között elhelyezkedő paraméter mi lehet. A paraméterként szereplő mennyiség, egyes esetekben csak egy meghatározott tipusu mennyiség lehet /un. elemi paraméter: egészszám, valós szám, azonositó, cimke, ujabb kulcsszó stb./. Más esetekben ezek közül bizonyos tipusuak lehetnek - bizonyosak nem. Még általánosabb esetben bizonyos tipusu kifejezések állhatnak az adott helyen /lista, aritmetikai kifejezés, logikai reláció stb./, ezek ismét kulcsszavakkal és közöttük álló paraméterekkel jellemezhetők.

Ha egy utasitást le akarunk forditani, akkor ezt a következő lépésekben tehetjük:

1. Megállapitjuk, hogy az utasitás melyik utasitás tipus mintájához illeszkedik. /Melyik minta kulcsszavai szerepelnek benne a megfelelő sorrendben/. Ha a megfelelő mintát megtaláltuk, akkor parancsokat adhatunk bizonyos utasitások kiadására, a megfelelő paraméterek behelyettesitésével; bizonyos paraméterek tipusának az ellenőrzésére és ettől függően bizonyos utasitások kiadására; valamint kifejezések felismerésére és feldolgozására.

2. A kifejezések feldolgozása nem jelent mást, mint egy ugyanolyan feladatsorozat végrehajtását, mint amilyennel magát az utasitást dolgoztuk fel. A kifejezést magát is egy mintasorozattal vetjük egybe és a kapott paraméterek megint a fent leirt tipusuak lehetnek, lehet a paraméter természetesen ujabb alkifejezés is. A felismert mintának megfelelően ismét parancsokat adunk ki a forditás kiadására, a paraméterek vizsgálatára stb.

3. A minta felismerése egy parancssorozat kiadását vonja maga után, a parancsok sorrendje teljesen tetszőleges lehet, függetlenül az utasitás egyes részeinek sorrendjétől. A paramétereket bármilyen sorrendben annyiszor és annyiféleképpen használhatjuk, ahogy éppen szükséges. A vizsgálat tárgyául szolgáló paramétereket kiegészithetjük, ha szükséges segédszimbólumokkal a paramétereket összekapcsolhatjuk egy kifejezéssé és azt vizsgálhatjuk stb.

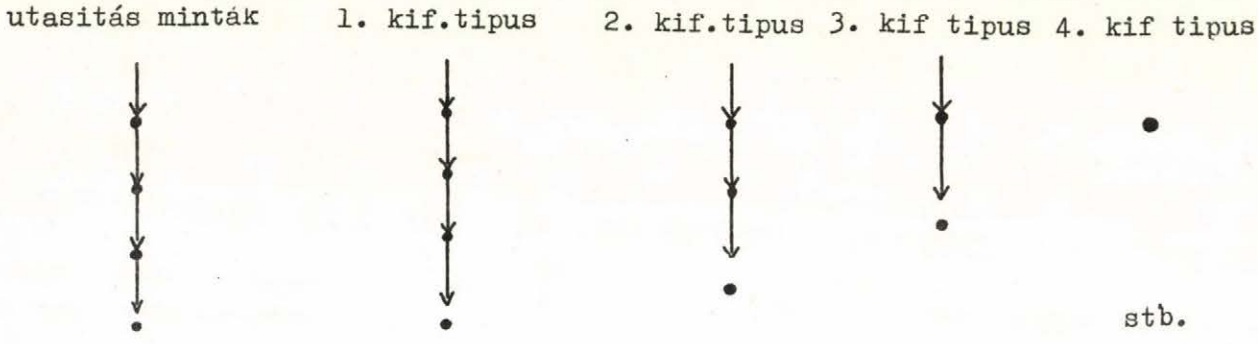
Mint láttuk, az utasitásokat, és a bennük szereplő kifejezéseket, minták alapján ismerjük fel. Az elemi paraméterek közül a kulcsszavakat szintén minták alapján ismerhetjük fel. A többi elemi paramétert szintén felismerhetjük minták sorozatával ez az eljárás azonban feleslegesen bonyolult és nehézkes, célszerünek látszik, hogy ezekre, illetve ezek bizonyos kombinációira közvetlen feldolgozó rutinokat készitsünk. Mivel ezek a rutinok makróknak egy sorozatát helyettesitik, és bizonyos esetekben ugy használhatók fel, mintha valódi makrók lennének, ezért rendszermakróknak hivjuk őket. Ezek közül a rendszermakrók közül különleges szerepet játszik két makró, az egyik bármilyen sorral összevetve forditásként nem ad semmit és mellékhatásként hibajelzést ad: közli, hogy a sor az eddig vizsgált minták alapján nem azonosit-

ható egyetlen makróhivással sem; a másik ezzel szemben bármilyen sorhoz önmagát rendeli fordításként.

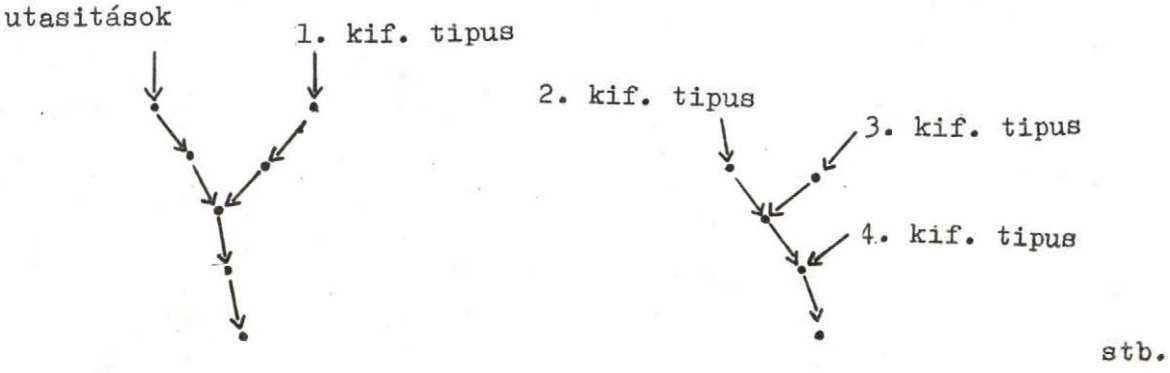
Összefoglalva: szintaktikus felismerés a következőképpen történik:

Először adott az utasitások felismerésére szolgáló mintáknak egy sorozata, ha ezek alapján az utasitás nem azonositható, akkor nyilvánvalóan hibás. Ezután jön az utasitásban szereplő paraméterek vizsgálata, ezeket a paramétereket osztályokba sorolhatjuk az egyes osztályok felismerése ismét egyes minták, vagy mintasorozatok alapján lehetséges.

A fentiek szerint a makróhivásokat jelentő minták halmazát igy ábrázolhatjuk.



A tapasztalatok azonban azt mutatják, hogy ez az ábrázolás általában redundáns, mert a programozási nyelvekben szereplő kifejezések általában valamilyen hierarhikus strukturában ábrázolhatók, azaz ezeket a független láncokat össze lehet kapcsolni. Valahogy igy:



A szintaxis ilyen módon történő megadása kapcsolatba hozható a metanyelvi definicióval is. Ha adott egy nyelvnek a leirása egy operátorgrammatikával, akkor ebből a minták és ezek hierarhikus elrendezése nehézség nélkül megadható. Az esetek nagy részében ez forditva is igaz, de nem mindig. Az a tény, hogy a paramétereket átrendezhetem, összekapcsolhatom, megsokszorozhatom ahányszor kell, lehetővé teszi olyan nyelvek forditását is, amelyek a CF nyelvek szokásos módszereivel nem fordithatók le. Másrészt az is igaz, hogy operátor grammatikával meg nem adható CF nyelvek leforditására nem alkalmas, ilyen programozási nyelvre azonban nehéz példát találni.

Persze egy nyelv egyes utasitásainak leforditása nem jelenti a /teljes/ nyelven irott programok leforditását, szükség van az egyes utasitások közötti kapcsolat megteremtésére is. Ezt a makroprocesszor belső állapotának változtatásaival hozzuk létre. A makroprocesszor belső állapota több elemből tevődik össze: ha egy utasitás forditása a processzor belső állapotát megváltoztatja, akkor ez hatással lesz a többi sorok feldolgozására is.

Az itt vázolt elvek megvalósitásának első fázisát irja le a következő fejezet.

A PROCESSZOR MÜKÖDÉSE

A processzor inputjáról rekordokat, esetünkben kocsi-vissza soremeléssel lezárt sorokat kap. A sorokat két csoportra osztjuk: direktivákra és szövegsorokra. A direktivák az "&" jellel kezdődnek.

- A direktivák hatására a processzor belső állapota és ezáltal müködése megváltozik.
- A szövegsorokat a processzor feldolgozza, és eredményül egy másik szöveget ad ki.

A direktivákat hatásuk szerint a következő csoportokba sorolhatjuk:

A/ Müveletek a processzor belső változóival

A processzorban az abc minden betüjéhez egy egész értékü változó tartozik, ezek a változók a müködés során értéket kaphatnak, illetve hivatkozhatunk a bennük szereplő értékekre. A változók 16 bitesek, kezdőértékük nulla.

Lehetőségünk van arra, hogy a négy alapmüveletet és a maradékképzést elvégezzük $\underline{\text{mod } 2^{16}}$.

A direktivák alakja:

A direktiva "&" jellel kezdődik, utána az a betü következik, amelyhez tartozó változó az értéket kapja, utána egy egyenlőségjel áll, amelynek jobboldalán egy érték vagy két müveleti jellel összekapcsolt érték áll. Az érték egy számjegyekből és spacekből álló egész szám vagy egy másik változó értéke lehet, az utóbbira "@" jellel és a megfelelő betüvel hivatkozhatunk. Az összeadás jele a + jel, a kivonásé a -, a szorzásé *. Mivel az osztás és maradékképzés értelme nem nyilvánvaló, ezeket a következőképen definiáljuk:

$$\underline{a \,/\, b} = \begin{cases} 0 \text{ ha } |a| < |b| \\ \text{sign}(a) \cdot \text{sign}(b) \cdot \text{ent}\left(\frac{|a|}{|b|}\right) \end{cases}$$

ahol feltételezzük, hogy $-(2^{15}) < a, b < (2^{15})-1$ és ahol az ent függvény az egész részt képzi.

$$\underline{a : b} = a - (a/b)\, b$$

azaz a n egész számra nézve 2n-1 db különböző maradékot kaphatunk $-(n-1)$-től $+(n-1)$-ig. A maradék előjele mindig azonos az osztandó előjelével.

Megjegyezzük még, hogy az egész szám állhat csupa space-ből, vagy lehet üres string is, ez esetben a szám értéke nulla.

Hibajelzések:

Ha az egyenlőségjel bal oldala hibás, akkor a konzolirógépen az:

ERROR IN A M.V. NAME
hibaüzenetet kapjuk.

Ha az egyenlőségjel jobb oldalán szereplő érték hibás, akkor az:

ERROR IN A NUMBER
üzenetet kapjuk.

Példák:

Helyes direktivák:

```
&A = 124
&B = - 42
&Z = @ A + 14
&B = - @ B
&P = @ B * @ D
&P = 142 / @ P
&P = 156 : 12
```

Hibás direktivák:

```
&AB = 42
ERROR IN A M.V. NAME
&A = + 12 - 7
ERROR IN A NUMBER
&C = 142 : _
DIVISION BY ZERO
```

B/ Input kijelölő direktivák

A processzornak kétféle inputja van:

- elsődleges az input, ha egy külső adathordozó tartalmát olvassa be,
- másodlagos ha, egy sajátmaga által készitett rekordsorozatot olvas el ujra.

Az elsődleges inputot a felhasználó jelöli ki:

```
&CL
```

A konzolirógép klaviaturájáról olvas.

```
&TAPE
```

A lyukszalagolvasóról olvas.

A másodlagos inputra való áttérés automatikusan történik, a visszatérés ugyancsak automatikus.

C/ Ugró direktivák

A processzor az adott inputról általában egymás után olvassa be a sorokat. Kivéve, ha feltételes vagy feltétlen ugrást irunk elő, ilyenkor megfelelő számu sort kihagyunk.

Ha a beolvasás valamelyik elsődleges inputról történt, akkor az összes többi sort is innen veszi a processzor, amig a következő értékelendő sorra érkezik. Ha azonban a beolvasás a másodlagos inputról történt, akkor lehetséges, hogy több sort kell kihagynia, mint amennyit elkészitett, ilyenkor automatikusan visszatér az elsődleges inputra és a fennmaradó különbségnek megfelelő számu sort onnan hagyja ki. Az utasitások alakjai a következők:

&SKIP n
&SKIP n IF e POSITIV
&SKIP n IF e NEGATIV
&SKIP n IF e ZERO

n és e két értékre való hivatkozás a fentebb emlitett formában. Ezek közül $0 \leqq n < 2^{16}$ e pedig $-(2^{15}) \leqq e \leqq (2^{15}) - 1$.

Példák:

&SKIP 7
&SKIP @B
&SKIP 2 IF @A POSITIV

stb.

D/ Makró definiciók

A processzor munkájához makródefinicíókat kap. A makrodefinició több sorból áll, de funkcionálisan egyetlen egységnek tekinthető. A definició hatására a processzor táblázataiba egy uj makró kerül feljegyzésre. Formailag négy részre osztható a makródefinició.

Az első sor egy direktiva:

&MACRO NO n NEXT k
n és k két értékre való hivatkozás:
$0 < n < 255$ és $0 \leqq k < 255$.

A makrók egy azonositó számmal rendelkeznek, ez nem lehet nulla, ez a szám az "n". A makrók a táblázatban ugy vannak elhelyezve, hogy minden makrónak lehet rákövetkezője, ezt jelöljük "k"-val. Több makrónak is lehet ugyanaz a rákövetkezője, ha egy makrónak nincs rákövetkezője, azt ugy jelöljük, hogy k=1 vagy 0. Két makró nem viselheti ugyanazt a számot. Ha a makró száma zéró /pl. üres string/, akkor a:

ZERO MACRO

hibajelzést kapjuk és a makró definició nem lép életbe.

Ha a makró száma már egyszer fel volt használva, akkor a:

REDECL

hibajelzést kapunk és a makró nem lép életbe, hanem az előző definició marad fenn.

A makró következő sora egy <u>minta</u>, a minta 3 féle különböző elemből állhat:

<u>Kulcsszó:</u> bármilyen nem üres string, amely nem tartalmaz "&", "?", "!" karaktereket. A sort ugy tekinthetjük, mint amelyik mindig ugyanazzal a kulcsszóval kezdődik és a "cr lf" kulcsszóval végződik.

<u>Kötött paraméterjel:</u> egy "!"-ekből álló nem üres string. Csak kulcsszó előtt, vagy után állhat.

<u>Szabad paraméterjel:</u> egy "?"-ekből álló nem üres string. Két kulcsszó, illetve a hozzájuk tartozó kötött paraméterek között állhat. /Általában csak egyet használunk/.

Példák:

!!!!! DIMENSION ? (?), ? (?), ? (?), ? (?)
!!!!! RES ?
!!!!! CH! ?
! ?? ! (!!??!!) !!??!

stb.

A további sorok képezik a <u>makró törzsét.</u> A törzs lehet üres is.
A makró utolsó sora egy záró direktiva
&END OF MACRO

E/ Az azonositást irányitó direktivák

A processzor munkája során megvizsgálja, hogy a beolvasásra kerülő sor összeilleszthető-e valamelyik makrómintával. Azt, hogy az összehasonlitásban melyik minta vegyen részt, a következő módon jelölhetjük ki. A definiciók rendezve vannak oly módon, hogy minden definiciónak van egy rákövetkezője. Amikor tehát egy makró definiciót kijelölünk azonositás céljára, akkor ezzel egy egész sorozatot is kijelöltünk. A vizsgálat addig tart, amig az első megfelelő definiciót megtaláljuk, vagy a sorozat végére érünk. A sorozat végén mindig egy rendszermakró áll, ez tulajdonképpen nem is makró, hanem egy olyan utasitás, amelyet akkor kell végrehajtani, ha a sorozat végére értünk.

Az összehasonlitásban szereplő sorozat első elemét kijelölhetjük tartósan a:

&BEGIN AT n

direktivával ennek, hatása mindaddig tart, amig egy ujabb direktivával más sorozatot jelölünk ki.

A sorozatot kijelölhetjük ideiglenesen is a:

&MATCH WITH n

direktivával, ez csak a következő sorra érvényes és az utána következő sor ismét a tartósan kijelölt mintasorozattal hasonlitjuk össze.

"n" mindkét esetben egy korábban kijelölt makrónak megfelelő érték lehet, ha az értéknek megfelelő makró nincs deklarálva, akkor a:

MACRO NO n UNDEFINED

hibajelzést kapunk.

Ezek után nézzük végig processzor munkájának pontos menetét.

A processzor inputjáról beolvas egy sort. Ezután megvizsgálja, hogy van-e benne hivatkozás a belső változók értékeire. Ha ilyen hivatkozás van, akkor ezt helyettesiti a változók aktuális értékével. A hivatkozás @-val és a megfelelő betüvel történik: @A, @B,..., @X, stb.

Szövegsorokban a változó helyére a megfelelő érték kerül a következő formában: a számok 0-tól 32767-ig annyi karakterben ahány értékes jegyet tartalmaz a szám, a többi szám -1-től -32768-ig hasonló módon "-" jellel a szám előtt.

Ezután a processzor megvizsgálja, hogy a sor "&" jellel kezdődik-e. Ha a sor "&" jellel kezdődik, összehasonlitja a lehetséges direktivákkal és összehasonlitás közben megállapitja a direktiva aktuális paramétereit. Ha megtalálta a megfelelő direktiva tipust, akkor a megfelelő akciókat végrehajtja és uj sor beolvasására tér át. Ha a sor egyetlen direktivával sem azonositható:

UNMATCHABLE

hibajelzést ad megismétli a sort és uj sor beolvasására tér át.

Ha a direktiva egy makró definició eleje, akkor a következő sorokat a processzor a záró direktiváig változtatás nélkül /a változókat sem értékeli ki/ másolja be a táblázataiba.

Ha a feldolgozandó sor szövegsor, akkor összehasonlitásra kerül az adott pillanatban összehasonlitásra kijelölt minta sorozattal. Azt mondjuk, hogy a szövegsor illeszkedik a mintához, ha a sor felbontható olyan módon, hogy a kulcsszavakhoz kötött és szabad paraméterekhez egy-egy része tartozik a sornak ugy, hogy:

1. A kulcsszóhoz tartozó rész karakterenként megegyezik a kulcsszóval.

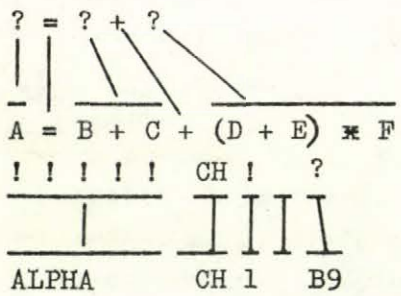
2. A kötött paraméterhez tartozó karaktereknek a száma megegyezik a ! jelek számával és az adott részben nincs "(" vagy ")".

3. A szabad paraméterhez tartozó rész bármilyen hosszu /üres is/ lehet, de a benne szereplő zárójelek kiegyensulyozottak.

4. A megfelelő részek sorrendje a két sorban azonos.

Ha ilyen illeszkedés többféleképpen is létrejöhet, akkor közülük az lesz érvényes, amelyikben a kulcsszók elhelyezkedése a legjobboldalibb. Természetesen az utóbbi megjegyzés csak egy mintára érvényes, mégpedig az első illeszkedőre, azok közül, amelyeket illeszteni próbáltunk.

Példa:



Fontos megjegyezni, hogy mind a mintában, mind a szövegsorban a szóköz szignifikáns.

Mint a definicióból látszik, az illeszkedés a szabad és kötött paraméterekhez egyértelmüen hozzárendel bizonyos stringeket, ezeket a paraméter aktuális értékének nevezzük. A paramétereket balról jobbra előfordulásuk sorrendjében sorszámozzuk. Ezek szerint a fenti példákban:

1. példa:

1. paraméter: A
2. paraméter: B + C
3. paraméter: (D + E) * F

2. példa:

1. paraméter: ALPHA
2. paraméter: 1
3. paraméter: B9

A processzor munkája közben a szövegsort sorban összehasonlitja az egyes mintákkal, az összehasonlitás az első illeszkedő mintánál megáll. Ha ilyen minta mincs, akkor a sorozat végén egy rendszer makró kerül végrehajtásra.

A 0. rendszer makró: hatására kiiródik, hogy:

UNMATCHABLE:

és megismétli a vizsgált szövegsort, majd uj sor beolvasására tér át.

Az 1. rendszermakró: hatására a sort változatlanul kiadja az outputon.

Ha sor illeszkedett egy makró mintához, akkor egy stackbe bemásolásra kerül a megfelelő makró törzse olyan formában, hogy benne a paraméterekre való hivatkozás helyére a megfelelő aktuális paraméter kerül. A paraméterekre a "@" karakterrel és a paraméter sorszámával /1-től 9-ig/ hivatkozhatunk.

A törzs a stackbe olyan módon kerül elhelyezésre, hogy a stack tetején a makrótörzs első sora áll. A továbbiakban a sorokat a stackből olvassuk ki mindaddig, amig a stack kiürül, azaz a stack a másodlagos input.

A kiolvasott sorokkal ugyanolyan módon járunk el, mint az elsődleges inputról származókkal, egyetlen kivétellel, amikor a 0. számu rendszermakróra hivatkozunk akkor másodlagos input esetén a sort az adott formában kiadjuk az outputon. Ha a stack kiürült visszatérünk az elsődleges inputra.

A processzor munkája elején a következő állapotban van:

a változók értéke nulla,
a definiciós táblázatban csak a rendszermakrók vannak,
az azonositás az 1. számu rendszermakrónál kezdődik,
az olvasás a konzolirógép klaviaturájáról történik.

A processzor munkája elvileg soha nem fejeződik be, gyakorlatilag egy

&CL

utasitással ér véget, amelyre nem kap ujabb inputot.

A használattal kapcsolatban megjegyezzük, hogy a definiciók sorrendje tetszőleges, de azonositása kijelölt mintasorozatban ismeretlen makró nem lehet, ez esetben:

MACRO NO n UNDEFINED

hibajelzést kapunk.

A makróprocesszor felhasználása

A makróprocesszor olyan nyelvek manipulálására alkalmas, amelyek nagyjából követik az "egy sor - egy utasitás" elvet. Lehetőségünk van arra, hogy bármely ilyen nyelvben makrókat irjunk a gyakrabban ismétlődő utasitásokra.

1. Példa: a CII 10010 assemblerében nincs kivonás, és szubrutin hivás és visszatérés a processzor lehetővé teszi, hogy formailag pontosan az Astrol assemblernek megfelelő formáju uj utasitásokat vezessünk be és ezeket programozás közben használjuk a forditás előtt egy megelőző fázisban a megfelelő gépikóddal behelyettesitsük.

Pl. kivonás: az operandusként adott cimen elhelyezett értékből kivonja az akkumulátor tartalmát és az eredményt az akkuba teszi.

```
&MACRO NO 12 NEXT 13
!!!!! SBC ?
 @1   ECT .CO      KIVONAS BYTERA
      ASC  @2
&END OF MACRO
&MACRO NO 13 NEXT 14
!!!!! SBM ?
 @1   ECT .F8      KIVONAS SZORA
      ECT .78
      ASM @ 2
&END OF MACRO
```

A szubrutin hivás a következő:

```
&MACRO NO 14 NEXT 15
!!!!! BSR ?
 @1   APM ⌀CI      SZUBRUTIN HIVAS
      RTM .24
      APM ⌘+4
      BRI ⌘⌘
      ADS @2
&END OF MACRO
```

A visszatérés:

```
&MACRO NO 15 NEXT 16
!!!!! RTN
 @1    IM3 .24
       IM3 .24
       IM2 .24
       BRI **
&END OF MACRO
```

Az itt megadott szubrutin hivást és visszatérést persze nem lehet többszörös mélységben használni, de lehetne olyat is késziteni.

Megjegyezzük még, hogy az igy definiált utasitásokat nem lehet SCN utasitás után használni, ezért vezessünk be még egy makrót:

```
&MACRO NO 16 NEXT 1
!!!!! SCN ?
 @1    SCN @2
&MATCH WITH 1
&END OF MACRO
```

Ha több makrónk nincs, akkor csak ezt a sorozatot kell vizsgálnunk:

```
&BEGIN AT 12
```

Ha ezek után a következő "A" programot adjuk inputként, akkor outputként a "B" programot kapjuk:

```
 "A"                                  "B"
       .                                    .
       .                                    .
       .                                    .
       .                                    .
      APC .17                              APC .17
      SCN .08          ------------>       SCN .08
      BRI LOFUL                            BRI LOFUL
      SBC CA 8         ------------>       ECT .CO   KIVONAS BYTRA
LOFUL SCN .20                              ASC CA 8
```

```
     BRI KENCE                    LOFUL  SCN .20
BENCE BSR SUBR 1                         BRI KENCE
KENCE APM .7 A                   BENCE  APM ØCI
                                         RTM .24
     RTM .42¤                            APM ¤+4
     BRI BENCE                           BRI ¤¤
                                         ADS SUBR 1
                                 KENCE  APM .7 A
                                         RTM .42¤
                                         BRI BENCE
```

Az a nyelv, amelyet makrókkal kiterjesztünk nem csak assembler szintü lehet.

Tegyük fel, hogy például egy fortran programban többször fordul elő olyan helyzet, amikor bizonyos változók értékét 1-gyel növelni kell, ha a változók száma fix ezt szubrutinnal is megoldhatjuk elvileg, de nem tul hatásosan; a következő makróval viszont általánosan is megoldhatjuk:

```
&MACRO NO 44 NEXT 1
!!!!! INCREMENT: ?
 @1   CONTINUE
&MATCH WITH 45
# @2
&END OF MACRO
&MACRO NO 45 NEXT 46
# ?, ?
     @2 = @ 2 + 1
&MATCH WITH 45
# @1
&END OF MACRO
&MACRO NO 46 NEXT 0
# ?
     @1 = @1 + 1
&END OF MACRO
```

Megjegyzés: A # jelet csak segédszimbólumként használjuk azért, hogy a minta ne egyetlen szabad paraméterből álljon, hiszen az a minta bármilyen sorra alkalmazható lenne. A # helyett bármilyen más karakter is állhatna.

Ha most a Fortran programba a következő utasitást irjuk:

```
124   INCREMENT A, B, C, D, E
```

akkor eredményül a következőt kapjuk:

```
124   CONTINUE
      E = E + 1
      D = D + 1
      C = C + 1
      B = B + 1
      A = A + 1
```

2. Példa: Fortran programoknál egy másik alkalmazási lehetőség: A közönséges Fortran programok csak fix méretü tömböket használhatnak, ez a programok irásánál és felhasználásánál kényelmetlenségeket okozhat.

Sokkal kisebb a kényelmetlenség akkor, ha a programot fix méretre irjuk meg, de az adott méretnek megfelelő konstans helyeire egy-egy a makróprocesszor belső változójára vonatkozó hivatkozást teszünk. Ilyenkor lehetőségünk van arra, hogy a processzor belső változóinak értékeket adva a programot lefuttassuk és igy a kivánt fix méretü programot gyorsan előállitsuk.

3. Példa: A következőben egy egyszerü nyelvfordító programjának egy részét és egy példaprogramot mutatunk be, amelyet egy aritmetikai és input-output konverziós program csomag segitségével lehet végrehajtani.

A nyelv szintaxisa:

A program deklarációs utasitásokkal kezdődik.

DECL lista

A listában fel kell sorolnunk a programban szereplő változókat egy és két dimenziós tömböket.

DATA lista

A listában a programban felhasználásra kerülő adatokat kell felsorolni.

A deklarációs DATA és DECL utasitások sorát egy:

PROGRAM

nevü utasitással kell lezárni.

Ez után jönnek a végrehajtandó utasitások, ezek:

READ lista

a listában változó vagy tömb nevek vannak elhelyezve, hatására a DATA utasitásokban elhelyezett adatok közül a soron következőket elhelyezi a lista változóiba.

PRINT lista

kiirja a konzolirógépen a listában elhelyezett változók tartalmát, egy sorba négy adat kerül utasitás végén automatikusan sort emel.

STOP

A program logikai vége.

LET változó 1. = változó 2. $\begin{matrix} + \\ - \\ * \\ / \end{matrix}$ változó 3.

A 2., 3. változóval elvégzi a megfelelő müveletet és értékét az első változónak adja.

END

A program fizikai vége.

REM string

kommentek elhelyezésére szolgál.

Az IF, GOTO stb. utasitásokat nem ismertetjük, mert a programban nem szerepelnek. A végrehajtható utasitásoknak lehet referencia számuk a DATA, DECL, PROGRAM, END, REM utasitásnak nem lehet.

A forditó program gyanánt szolgáló makrókészlet:

```
&MACRO NO11NEXT13
PROGRAM
BEGE   APM KKK
       RTM .34
       APM **
       RTM .61
       BRI .62*
KKK    ADS *+2
&BEGIN AT 12
&END OF MACRO
&MACRO NO12NEXT14
?READ ?
&MATCH WITH 20
 @2
&END OF MACRO
&MACRO NO13NEXTO
DATA?
&SKIP3IF@APOSITIV
&A=1
POIN   ADS DAT
DAT    CH1 20
&MATCH WITH 25
 @1
&END OF MACRO
&MACRO NO14NEXT15
?PRINT ?
&MATCH WITH 30
 @2
```

```
      ADS .32AD
&END OF MACRO
&MACRO NO15NEXT17
END
      ADS *+2
FIN   ADS .A00
      HLT
      FIN BEGE
&END OF MACRO
&MACRO NO20NEXT21
  ?,?
&MATCH WITH 20
 @1
      ADS .22B8
      ADS POIN
      ADS X@2
&END OF MACRO
&MACRO NO21NEXTO
 ?
      ADS .22B8
      ADS POIN
      ADS X@1
&END OF MACRO
&MACRO NO 25 NEXT 26
 ?,?
&MATCH WITH25
 @1
      CA8 @2        D
&END OF MACRO
&MACRO NO26 NEXTO
 ?
      CA6 @1        D
&END OF MACRO
```

```
&MACRO NO 30 NEXT31
 ?,?
&MATCH WITH30
 @1
      ADS .127A
      CH1 55
      ADS X@2
      ADS .32AB
&B=@B+1
&B=@B:4
&SKIP1IF@BPOSITIV
      ADS .32AD
&END OF MACRO
&MACRO NO31NEXTO
 ?
      ADS .127A
      CH1 55
      ADS X@1
      ADS .32AB
&END OF MACRO
&MACRO NO1ONEXT11
DECL ?
&MATCH WITH35
 @1
&END OF MACRO
&MACRO NO35NEXT36
 ?,?
&MATCH WITH35
 @1
&MATCH WITH36
 @2
&END OF MACRO
&MACRO NO36NEXT37
 ?(?)
&A=5*@2
X@1   RES @A
&END OF MACRO
```

```
&MACRO NO37NEXTO
 ?
X@1     RES 5
&END OF MACRO
&MACRO NO17NEXT18
?STOP
        ADS FIN
&END OF MACRO
&MACRO NO18NEXT19
REM?
&END OF MACRO
&MACRO NO19NEXTO
?LET ?=?
&MATCH WITH40
 @3
        ADS .F4E
        ADS X@2
&END OF MACRO
&MACRO NO40NEXTO
 ?+?
        ADS .FOO
        ADS X@1
        ADS .1662
        ADS X@2
&END OF MACRO
&BEGIN AT10
&CL
A leforditandó program:
DECL A,B,C,D,E,F,G,H,I,J,K
DATA 11,21,31,41,51,61,71
DATA 81
PROGRAM
REM EZ ITT KEREM EGY PROGRAMM
READ A,B,C,D
LET F=A+B
LET F=C+D
LET K=F+E
```

```
READ E,F,G,H,I,J
PRINT K
PRINT A,B,C,D,E,F,G,H,I,J,K
STOP
END
&CL
```

A leforditott program:

```
XA    RES 5
XB    RES 5
XC    RES 5
XD    RES 5
XE    RES 5
XF    RES 5
XG    RES 5
XH    RES 5
XI    RES 5
XJ    RES 5
XK    RES 5
POIN  ADS DAT
DAT   CH1 20
      CA6  11        D
      CA8 21       D
      CA8 31,41        D
      CA8 51       D
      CA8 61       D
      CA8 71       D
      CA6  81       D
BEGE  APM KKK
      RTM .34
      APM **
      RTM .61
      BRI .62 *
KKK   ADS *+2
      ADS .22B8
      ADS POIN
      ADS XA
```

```
ADS .22B8
ADS POIN
ADS XB
ADS .22B8
ADS POIN
ADS XC
ADS .22B8
ADS POIN
ADS XD
ADS .FOO
ADS XA
ADS .1662
ADS XB
ADS .F4F
ADS XE
ADS .FOO
ADS XC
ADS .1662
ADS XD
ADS .F4E
ADS XF
ADS .FOO
ADS XF
ADS .1662
ADS XE
ADS .F4E
ADS XK
ADS .22B8
ADS POIN
ADS XE
ADS .22B8
ADS POIN
ADS XF
ADS .22B8
ADS POIN
ADS XG
ADS .22B8
```

```
ADS POIN
ADS XH
ADS .22B8
ADS POIN
ADS XI
ADS .22B8
ADS POIN
ADS XJ
ADS .127A
CH1 55
ADS XK
ADS .32AB
ADS .32AD
ADS .127A
CH1 55
ADS XA
ADS .32AB
ADS .127A
CH1 55
ADS XB
ADS .32AB
ADS .127A
CH1 55
ADS XC
ADS .32AB
ADS .127A
CH1 55
ADS XD
ADS .32AB
ADS .127A
CH1 55
ADS XE
ADS .32AB
ADS .32AD
ADS .127A
CH1 55
ADS XF
```

```
        ADS .32AB
        ADS .127A
        CH1 55
        ADS XG
        ADS .32AB
        ADS .127A
        CH1 55
        ADS XH
        ADS .32AB
        ADS .127A
        CH1 55
        ADS XI
        ADS .32AB
        ADS .32AD
        ADS .127A
        CH1 55
        ADS XJ
        ADS .32AB
        ADS .127A
        CH1 55
        ADS XK
        ADS .32AB
        ADS .32AD
        ADS FIN
        ADS *+2
FIN     ADS .A00
        HLT
        FIN BEGE
@
@
```

MP/0 továbbfejlesztése

Ha összevetjük a 2. és 3. rész tartalmát, rögtön láthatjuk az MP/0 a rendszer makrók egy részét még nem valósitotta meg. A továbbiakban szükség van még a következő rendszer makrókra.

1. A nullás makró az elsődleges inputról vett sor esetén hibajelzést ad, a másodlagos input esetén azonban nem. Szükség van egy olyanra is, amely mindkét esetben hibajelzést ad.

2. Rendszer makró természetes számok konvertálására.

   Az inputként szolgáló sor egy előjeltelen számjegyekből és spacekből álló string. Ha a string az előírásoknak megfelel, akkor a számot elfogadja eredményként egy paraméter keletkezik, erre a paraméterre hivatkozni lehet a szokásos módon. A paraméter egy string, amely a szám hexadecimális alakjának megfelelő karakterekből áll.

   A makró emellett a következő mellékhatással is rendelkezik: a processzor a természetes számokat egy szótárban tartja nyilván, ha az adott szám a szótárban még nem szerepel, akkor felveszi, ad neki egy sorszámot, elhelyezi ezt a Z makróváltozóba, ugyanakkor elhelyezi a W makróváltozóban a számértékét is. Ezen kivül a szótárba feljegyzi a szám mellett az X változó értékét, ugyanezt az értéket elhelyezi az Y változóba is.

   Ha a szám már korábban előfordult, akkor a rendszermakró hatására Z-be az előfordulás sorszáma kerül, W-be a szám értéke, X értéke beirásra kerül a szótárba, a szótárban levő korábbi érték pedig Y-ba.

   Ha a bemenő string nem természetes szám, Z-be nulla kerül, Y és W értéke pedig meghatározatlan lesz, a szótár pedig nem változik.

3. Rendszermakró egész számok konvertálására.

   Lényegében megegyezik az előbbivel, de a bemenő string előjellel is kezdődhet, W értéke pedig nulla lesz.

4. Rendszermakró valós számok konvertálására.

   Inputként opcionális előjelből egész részből, tizedespontból és törtrészből álló stringet fogad el. Az egész rész, vagy a törtrész, vagy a tizedespont és a törtrész elmaradhat. Eredményként egy paramétert kapunk, ez a szám hexadecimális alakjának karaktereit tartalmazza.

A szám szintén egy szótárba kerül az eddigiekkel megegyező módon. Z értéke a szám sorszáma lesz. X értéke bekerül a szótárba és a régi érték Y-ba, W értéke nulla lesz.

5. Azonositók felismerése és elraktározása.

A rendszermakró az inputként szolgáló azonositót elteszi egy szótárba, illetve kikeresi belőle. X és Y szerepe a szokásos Z értéke mindig nulla lesz.
A sorszámot W fogja tartalmazni.

6. Rendszermakró egész szám és azonositó felismerésére, illetve valós szám és azonositó felismerésére. A 3. és 5, illetve a 4. és 5. pontokban leirtak közül valamelyiket végrehajtja.

A rendszermakrókon kivül a következő direktivák látszanak még szükségesnek:
Direktivák a szótárak kiüritésére.
Ezek a direktivák az egyes szótárakból egyenként kivennék a bennük elhelyezett tételeket.

Ciklus direktiva.
A direktiva hatására a következő sor nem egyszer, hanem többször szolgálna input céljára.
Olyan direktivák, amelyekkel egy soros hozzáférésü file-ra lehet irni, arról olvasni, és azt visszatekerni.

Lehetőséget szeretnénk nyujtani ezen kivül arra is, hogy egy makró törzsében ujabb makródefiniciót helyezzünk el. Ez a definició akkor aktivizálódna, amikor a külső makrót meghivjuk, és az igy deklarált makró természetesen függne a hivás paramétereitől.

Igy foglalhatjuk össze azokat a kiegészitéseket, amelyeket a közeljövőben remélhetőleg megvalósitunk és reméljük, hogy ilyen módon egy hasznos software eszközhöz jutunk.

ÁLTALÁNOS DIALÓGUS-PROGRAMRENDSZEREK. A DISTAR-B

/DIalogue STAte Records - Basic/ általános dialógusrendszer

Forgács Tamás - Krammer Gergely

BEVEZETÉS

Az ember és a számitógép közvetlen kapcsolatát lehetővé tevő on-line perifériák megjelenésével egyre jobban elterjedtek az ember-gép dialógusra épülő interaktiv programrendszerek - illetve olyan programok, amelyeknek jelentékeny része ilyen jellegü. A jövőben ilyen programok további elterjedésével lehet számolni. Jogosan felmerül az igény egy általános dialógusrendszer kidolgozására, amely a dialógus tárgyától függetlenül megfelelő segitséget nyujt a dialógusprogram készitőjének. [3], [4], [5]. Ebben a cikkben egy, a fenti célkitüzésnek megfelelő általános dialógusrendszert ismertetünk. A rendszer egyaránt alkalmazható kis- és nagy számitógépes rendszerekben, amit bizonyit az, hogy VIDEOTON 1010B és CDC 3300 számitógépekre egyaránt implementáltuk.

Az első részben először ismertetünk egy általános barkochbaprogramot. A barkochba-dialógust célszerü volt az általánositás kiindulópontjául választani, mert egyrészt - elméletileg - minden dialógus visszavezethető barkochbára /ezért várható, hogy a barkochba jellemző tulajdonságai az általánosabb eset legfőbb jellemzőit alkotják/, másrészt a szabályai egyszerüek és jól leirhatók, igy a feladat könnyen megfogalmazható.

Ezután a dialógusok, dialógusprogramok általános jellemzőit irjuk le, definiálunk néhány alapfogalmat, majd röviden ismertetjük az általános dialógusrendszer, a DISTAR-B alapelemeit.

A 2. részben részletesen leirjuk a megvalósitott DISTAR-B rendszert. Ebben a fejezetben mutatjuk meg azt is, hogy a felhasználó hogyan adaptálhatja saját feladatához az általános rendszert.

A 3. rész egy példát mutat be: a FIDI fixválaszu dialógusrendszet. Ez lényegében a barkochba-dialógus egy közvetlen általánositása. Lényege az, hogy a dialógus során feltett kérdésekre adható válaszok előre rögzitettek.

A 4. rész a DISTAR-B rendszer továbbfejlesztésének két nagyon jelentős állomását, a dialógus-szubrutintechnika és a dialógus-szegmentálás /dialógus-overlay/ alkalmazására kidolgozott módszereket ismerteti.

Az 5. rész a rendszer értékelését tartalmazza.

Köszönetet mondunk Pikler Gyula tudományos munkatársnak, aki a DISTAR-B első felhasználója volt és észrevételeivel erősen visszahatott magára a rendszerre is, Gerhardt Géza és Kocsis József tudományos segédmunkatársaknak, akik a programozás-technikai munkákat végezték elsősorban, de részt vállaltak a tervezésből is.

1. DIALÓGUSRENDSZEREK ÁLTALÁNOSITÁSÁNAK ALAPELVEI

1.1 Általános barkochba-program

A barkochba közismert dialógus-játék, amelyben az egyik játékos /A/ gondol egy tárgyra, személyre vagy elvont fogalomra, a másik /B/ pedig megpróbálja kitalálni azt. A játék során B kérdéseket tesz fel, amelyre A igennel vagy nemmel válaszol. B törekvése természetesen az, hogy fokozatosan leszükitse a lehetséges objektumok halmazát.

Abban az esetben, ha a lehetséges objektumok egy előre meghatározott véges halmaz elemei, a kérdező véges számu lépésben kitalálhatja a kérdéses tárgyat. A kérdező feladatát egy egyszerü bakochba-program is elvégezheti.

Könnyü olyan általános barkochba-programot is irni, amely először inputként meghatározott formában elfogadja a lehetséges objektumok megnevezését, a feltenni kivánt kérdések szövegét, valamint a kérdéseknek a válaszoktól függő sorrendjét. A kérdéseket és a kérdések feltevésének sorrendjét tehát előre meg kell tervezni. Ezeket a program első része beolvassa és elhelyezi a memóriában. Ezután a program kész a játékra.

A második fázisban a program kérdést ir ki, beolvassa a választ, elemzi azt, majd meghatározza a következő kérdést. Ezután a fenti folyamat ismétlődik. Végül a program kérdés helyett a kitalált objektum nevét irja ki.

Látható tehát, hogy a dialógus lefutása elemi események láncolata, ahol egy elemi esemény az alábbi tevékenységeket foglalja magában: kérdezés, válaszfogadás, válaszelemzés, uj kérdés kijelölése. Ha a dialógusprogram müködése során az i-ik kérdést irja ki, a válaszra pedig az i-ik módon reagál /vagyis igen, ill. nem válasz esetén az adott módon irányitja tovább a dialógust/, azaz az i-ik elemi eseményt hajtja végre, akkor azt mondjuk, hogy a dialógus az i-ik állapotban van.

Fentiek alapján igen egyszerü programot lehet irni, ha az adatokat a következőképpen adjuk meg:

- először megadjuk a dialógusban előforduló szövegeket, amelyeket a beolvasó program egy szövegtáblában helyez el
- ezután háromelemü állapotrekordokat adunk meg, amelyek tartalmazzák:
  1. a barkochba előrehaladása során egy adott állapotban kiirandó kérdés sorszámát /szövegtáblabeli helyét/
  2. az erre a kérdésre adott igen, illetve
  3. nem válasz esetén a következő állapotrekord sorszámát.

Ezek az állapotrekordok végzik tehát a barkochba program vezérlését a dialógus különböző állapotaiban.

1.2 Dialógusprogramok jellemzése. Az általánositott dialógusrendszer alapelemei

Először néhány alapfogalom pontos definicióját kell lerögzitenünk.

Dialógusnak nevezzük magát a folyamatot, amikor a számitógép és az operátor üzenet-váltások sorozatát végzi egy adott feladat elvégzése érdekében [5], [6].

Dialógusprogramnak nevezzük azt a számitógépes programot, amely egy adott dialógus realizálását végzi.

A dialógusprogram két fő részre osztható: a dialógus leirásra és a dialógus vezérlő programra. Az előbbi a dialógus összes lehetséges lefutásának - valamely alkalmas eszközzel - való leirása, az utóbbi lényegében egy interpreter, amely a dialógus-leirást értelmezi és végrehajtja /a végrehajtásnál természetesen az operátor-válaszokat is figyelembe veszi/.

Kérdésnek nevezzük a számitógép, válasznak az operátor üzeneteit.

A dialógusok /és igy a dialógusprogramok/ háromfélék lehetnek:

Statikus dialógus esetén a dialógus kezdetekor a folyamat összes lehetséges lefutása eleve adott és ezek közül egy fog realizálódni, az operátor válaszaitól függ, hogy melyik.

Dinamikus dialógusnál a program csak egy kezdeti dialógus-vázat anticipál, amelynek későbbi pontjai a dialógus során alakulhatnak ki /adaptiv, tanuló rendszerek/.

Pszeudodinamikus dialógusról beszélünk akkor, ha a dialógus lehetséges lefutásai ugyan adottak, de a kérdések szövegei módosulhatnak a dialógus során /esetleg néhány kérdés szövege éppen a dialógus alatt alakul ki/.

A dialógus során adható válaszok három csoportba sorolhatók:

a/ alapszó: egy előre rögzitett válaszhalmaz egyik eleme
b/ paraméter: tetszőleges szöveg vagy szám-válasz
c/ kevert: alapszavak és paraméterek kombinációja.

Ezek után rátérhetünk a dialógusprogramok általános jellemzésére.

A gép-ember között lezajló dialógus tulajdonképpen csak eszköz egy adott feladat elvégzéséhez. A feladatok igen sokfélék lehetnek, bizonyos dialógusok esetén a dialógus célja csak a dialógus végén realizálódik /barkochba, információ-visszakeresés/, más esetekben folytonosan képződik a dialógus során /számitógépes tervezés, oktatás/. Ez azt jelenti, hogy a dialógusprogramnak a dialógus lebonyolitásán kivül /ami általánositható/ egy sereg egyéb akciót is végre kell hajtania /ami feladatspecifikus/.

Általánosságban elmondható azonban a következő: Egy adott feladathoz tartozó összes dialógus leirható egy állapothalmazzal, és ezen állapotok közötti lehetséges átmenetekkel /dialógus-gráf/. A dialógus maga pedig reprezentálható - a lehetségesek közül egy - állapotátmenet sorozattal [8].

A dialógusprogramnak minden egyes állapotában a következőket kell végrehajtania:
- állapot-kezdő akció
- kérdezés
- válasz-beolvasás
- válasz-elemzés
- uj állapot meghatározása.

Fentiek közül két olyan lépés van, amelyiknél erősen hangsulyt kaphat a dialógusprogram feladatspecifikus jellege:

- Állapotkezdő akció egyszerü kérdés-felelet dialógusoknál nincs. Egy általános dialógusrendszernek azonban meg kell engednie, hogy a dialógus-tervező különféle /az éppen adott feladathoz szükséges/ akciókat definiálhasson.

- A válaszelemzés csak alapszó-válaszu /pl. barkochba/ dialógus esetén egyszerüen a válasz és az előre rögzitett válaszhalmaz elemeinek összehasonlitásából áll. Bonyolultabb feladatoknál viszont a válaszelemzéshez különféle aritmetikai, logikai vagy stringmanipulációkra lehet szükség.

Fentiekből belátható - mivel a dialógussal megoldható feladatok száma elvben végtelen -, hogy olyan zárt rendszer /pl. dialógusnyelv/, ami minden feladatra alkalmas, nem készithető.* Az általános dialógusrendszer létrehozói számára két ut lehetséges: Elkészithető egy olyan zárt rendszer, ami a lehetőség szerint igyekszik felölelni minél több alkalmazást, ez persze csak megszoritással nevezhető általánosnak.

A másik ut: elkészithető egy nyilt rendszer /pl. szubrutincsomag/, ami önmagában még nem alkalmas dialógusprogram irására, de a dialógusprogram tervezőjének jelentős segitséget ad, bármilyen adott tipusu dialógusprogram megirásához. A DISTAR-B rendszer ez utóbbi utat követi.

A DISTAR-B rendszer alapelemei a következők:

1. Egy dialógus-leirási koncepció: a szövegtábla és az állapottábla használata.
   A szövegtábla tárolja a dialógusprogramban szereplő összes szöveget, a szövegparaméterek kivételével. A dialógus menetét, a dialógus állapotainak egymásutánját állapotrekordokkal irjuk le. Az állapotrekordokat egy állapottábla tárolja.

2. A szövegtábla és az állapottábla gépi tárolásának konvenciója

   A szövegtáblát és az állapottáblát a rendszer egy belső tömbben tárolja. Mivel mindkettő változó hosszuságu lehet, ugyanabban a tömbben tárolunk még két pointer-vektort, a szövegvektort és az állapotvektort is. Ezen vektorok i-ik eleme, az i-ik szöveg-, ill. állapotrekord első elemére mutató pointer.

* Zárt rendszer alatt nem azt értjük, hogy nem bővithető, hanem hogy önmagában véve egységes egész, tehát minden külön kiegészités nélkül működtethető.

3. A szövegtábla és az állapotrekordok elemeit kezelő, valamint a dialógust végrehajtó általánosan megfogalmazható szubrutinok.

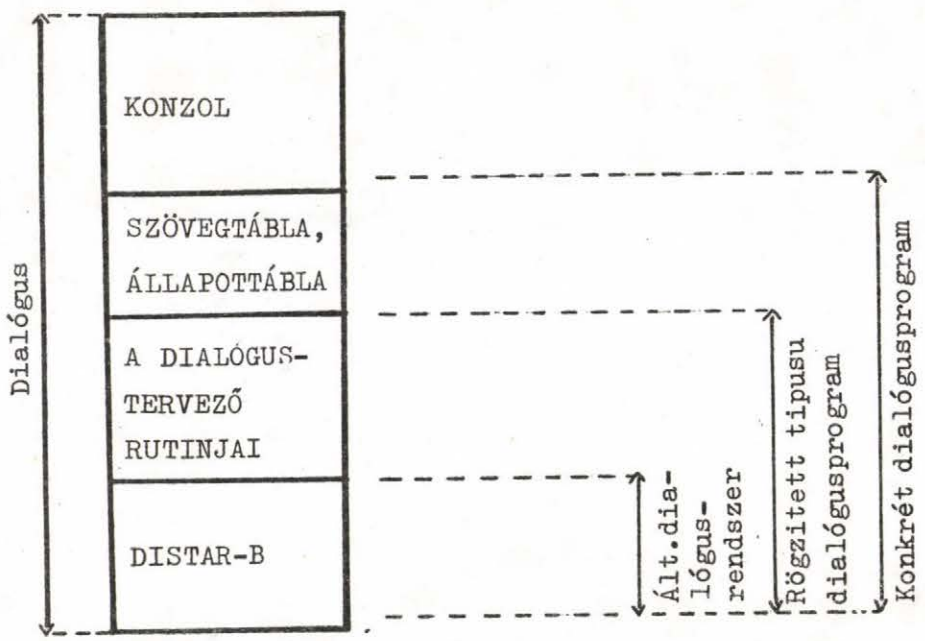
   A dialógus-vezérlő programot maga a programtervező köteles megirni, minden adott tipusu dialógusprogramhoz, tipusonként egyet. Ehhez használhatja fel az emlitett szubrutinokat.

4. A szövegtábla és állapotrekordok megadására szolgáló külső direkt kódrendszer.
   Ha egy dialógus-vezérlő program kész, erre tetszőleges számu dialógus-leirás készithető /mindegyik esetben persze be kell tartani a dialógus-vezérlő program által meghatározott feladatspecifikus jelleget/, miáltal tetszőleges számu dialógusprogram készithető. A dialógus-leirás elkészitésére szolgál ez a külső direkt kódrendszer.

5. Loader program, amely a külső direkt kódban megfogalmazott dialógus-leirást betölti a gépi tárolási konvencióknak megfelelően.

6. Dialógus-leirást megkönnyitő lehetőségek: dialógus-szubrutin alkalmazását és nagyméretü dialógusok szegmentálását lehetővé tevő szubrutinok.

A DISTAR-B rendszer felépitését mutatja az 1. sz. ábra.



1. ábra

Az általános dialógus-rendszer felépitése

1.3 Absztrakt dialógusgép

Az előző pontban leirt rendszert most egy másik szemléletmóddal vizsgáljuk meg. Tesszük ezt pedig azért, mert néhány fogalom bevezetése, ill. szemléltetése ebben az uj szemléletmódban egyszerübb, valamint a rendszer továbbfejlesztésénél számos uj szempontot nyerhetünk ezáltal.

Tekintsük a következő programozható gépet és nevezzük ezt absztrakt dialógusgépnek: a gép rendelkezik egy konzolperifériával és egy input perifériával. Valamilyen jelöléssel leirt /pl. az előző pontban emlitett külső direkt kódban/ dialógust az input perifériával beolvasunk. Ezután a dialógusgép készen áll a dialógus végrehajtására a konzolon keresztül.

A dialógusgép a következő egységekből áll:

- memória: programtár és munkatár
- fixtár: mikroprogramozott müveleti egység
- vezérlőegység
- mikroprogramozott loader, a külső direkt kódban leirt dialógus beolvasására.

A memóriának a programtár részében helyezkedik el a dialógus leirása belső kódban, amely állapotrekordokból /dialógus-utasitás/ és szövegrekordokból /dialógus konstans/ áll. A munkatárban helyezkednek el /általában csak időlegesen/ a dialógus során kapott paraméter-válaszok és egyéb közbenső adatok. A fixtár-ban helyezkednek el a dialógus végrehajtásának elemi müveletei, a szövegtábla és állapottábla részeit kezelő mikroprogramok. Ezek nagyrészét a DISTAR-B tartalmazza, a tervező további mikroprogramokat alakithat ki.

A vezérlőegység tulajdonképpen a dialógus-vezérlő program, amelyet általában magasabb szintü nyelven /FORTRAN/ lehet megirni, felhasználva hozzá a fixtár moduljait; a tervező készíti.

A dialógus leirását külső direkt kódban kell elkésziteni és a loader programmal a memóriába bevinni. /A dialógus leirásához magasabb szintü nyelv is tervezhető, hozzá forditóprogram irható. Ez a forditóprogram általában nem a dialógusgépen futva állitja elő a direkt kódu programot. Ha a dialógusgépre készitjük az ilyen programot, maga a dialógus is dialógus utján áll elő/.

A dialógusprogram összeállitásának menete ezen a dialógusgépen:

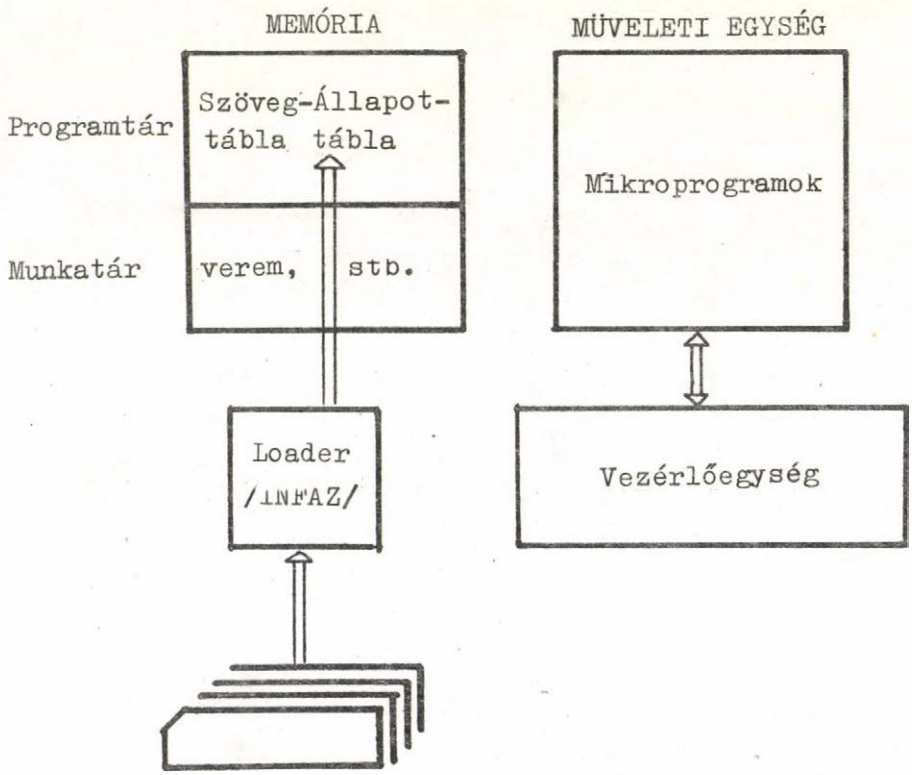
A mikroprogramok nagyrészét és a loadert a DISTAR-B tartalmazza. Az utóbbi meghatározza a programirás külső kódját.

A dialógustervező irja meg a dialógusgép vezérlő egységét és esetleg néhány mikroprogramot, majd összeállitja a dialógusgépet. Ehhez valamilyen digitális számológépet és annak programrendszerét veszi igénybe.

Végül ugyanez - vagy egy másik tervező - a külső kódban elkésziti a dialógus-leirást /állapottábla, szövegtábla/, amelyet a loaderrel a programtárba betölt.

Ezután a dialógusgép készen áll egy dialógus végrehajtására.

A dialógusgép vázlatos rajzát a 2. sz. ábra szemlélteti.



A vendéglátó számitógép programozási rendszere

2. ábra

Az absztrakt dialógusgép sematikus rajza

2. A DISTAR-B ÁLTALÁNOS DIALÓGUSRENDSZER

2.1 A szövegtábla és állapottábla belső tárolása

A dialógusgépre irt program /dialógus-leirás/, - amely előirja a dialógusgép müködését - a szövegtáblából és állapottáblából áll. Ezeket a dialógusgép loader-je tölti be a memóriába, amelyet egy ITAB nevü tömb képvisel.

Az ITAB tömb /3. sz. ábra/ első részében a szövegrekordok következnek egymás után. Ezt követik az állapotrekordok, majd a szövegtábla-vektor, illetve az állapottábla-vektor. Az utóbbiak k-ik, ill. l-ik eleme megadja a k-ik szöveg, illetve l-ik állapotrekord ITAB-beli helyét.

Egy szövegrekord n $c_1c_2 \ldots c_n$ alaku: az elől álló egész szám a szöveg karaktereinek száma; ezt követik maguk a karakterek.

Az állapotrekordok két fő részből állanak: akciós rész és elemző rész.

Az akciós rész tárolja egyrészt az u.n. állapot-kezdő akcióra vonatkozó információkat: programkapcsolókat, amelyek az adott állapotban aktivizálandó akciós rutinokra mutatnak, ezen akciós rutinok paramétereit, illetőleg ezen paramétertömbökre mutató pointereket, stb., másrészt a kiirandó kérdéssel kapcsolatos szövegtábla-mutatókat, esetleg a kérdés kiirásának módjára utaló számot /pl. alfanumerikus display esetén a kiirandó kérdés helyét a képernyőn/. Ha a dialógusprogram többféle állapotrekordot tartalmaz, akkor az akciós rész első eleme tipusjelző szám, amely megmutatja, hogy melyik tipusu állapotrekordról van szó. Az akciós részt a következő formában tároljuk:

$$n\ s_1 \ldots s_n, \text{ ahol n pozitiv egész szám,}$$

az akciós rész hosszát jelöli, s_i pedig az akciós rész i-ik eleme /l. 3. sz. ábra/.

Az elemző rész a válaszelemzést és az uj állapot kijelölését vezérli, ezért többnyire szövegtábla mutatókból /amelyek az egyes válaszalternativákra mutatnak/ és állapottábla-mutatókból /amelyek az egyes válasz-alternativákhoz tartozó uj állapotokra mutatnak/ áll. Esetleg tartalmazhat az elemzés módjára utaló valamilyen számot is.

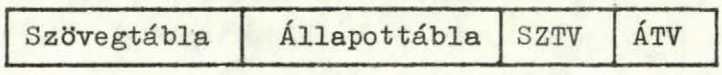
A válaszelemzés általában több lépésben történik, több alternativa megvizsgálása utján. Ezért az elemző részt ujabb egységekre, intervallumokra osztjuk fel, a válaszelemzés minden lépéséhez tartozik egy intervallum.

Az elemző rész tárolási módja tehát a következő:

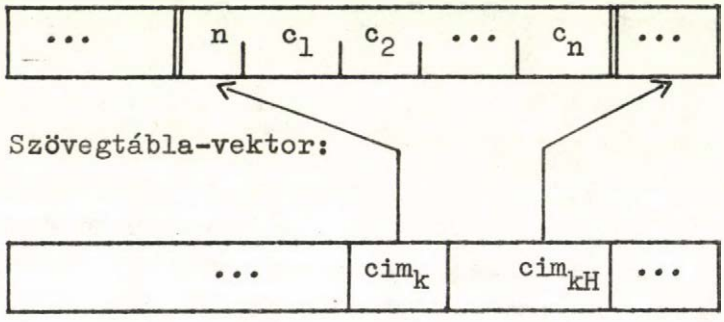
$$j\ m_1\ s_1^{/1/} \ldots s_{m1}^{/1/} \ldots \ldots m_j\ s_1^{/j/} \ldots \ldots s_{mj}^{/j/},$$

ahol j az intervallumok száma, m_i az i-ik intervallum hossza, $s_i^{/1/}$ az 1-ik intervallum i-ik eleme /1. 3. sz. ábra/.

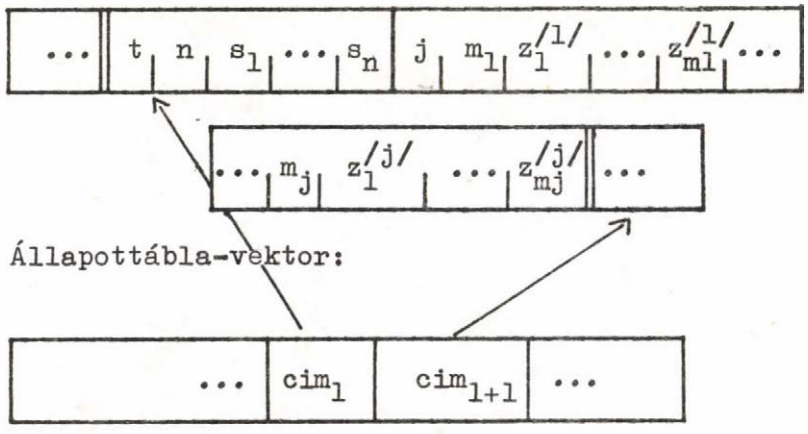
a/ ITAB:



b/ Szövegtábla:



c/ Állapottábla:



3. ábra

Mint fentiekből látható, a DISTAR-B az állapotrekordnak csak a fő strukturáját irja elő, a struktura részeinek /pl. akciós rész, intervallum/ belső szerkezete nem kötött, a dialógusprogram tervező határozhatja meg, és azután ennek alapján irhatja meg a dialógus-vezérlő programot.

2.2 A dialógus leirásának külső direkt kódja és a loader: INFAZ

A dialógus szövegrekordjainak és állapotrekordjainak megadására szolgál a szigoru formátumu direkt kód. Az ebben leirt program egy DIAL1 sorral kezdődik és egy DIAL9 sorral ér véget.

DIAL1 után megadható 10 kitüntetett szerepü karakter, amelyet INFAZ tárol és a dialógusrendszer rutinjai felhasználnak.

Az ezután következő DIAL2 a szövegtábla kezdetét jelenti: a szövegrekordok következnek $nHc_1 \ldots c_n$ alakban. A szövegrekordok után a DIAL3 sor jelzi az állapottábla kezdetét: minden állapotrekord uj sorban kezdődik; felépitése:

$$n\ As_1 \ldots s_n\ j\ E\ m_1\ I\ z_1^{/1/} \ldots z_{m1}^{/1/} \ldots mj\ I\ z_1^{/j/} \ldots z_{mj}^{/j/}$$

/lásd 2.1 rész/. Az ebben szereplő adatok mind egész számok:

n = az akciós rész hossza,
s_i = szövegsorszám, vagy más paraméter,
j = az elemző rész intervallumainak száma,
m_1 = az első intervallum hossza,
$z_k^{/1/}$ = az első intervallumbeli szövegsorszám stb.

Az INFAZ szubrutin a külső direkt kódban leirt sorokat beolvassa, majd az elválasztó DIAL direktiváknak és a rekordokban szereplő H, A, E, I elválasztójeleknek megfelelően elhelyezi az ITAB tömbben. Beolvasás közben alakitja ki az állapottábla és a szövegtábla-vektort. A DIAL9 sor hatására tárolja a szövegtábla és az állapottábla kezdő cimét.

DIAL2 és DIAL3 részek váltakozhatnak DIAL1 és DIAL9 között.

2.3 A szervező-program /vezérlőegység/ alapvető moduljai

A DISTAR-B rendszer öt szubrutint nyujt a dialógusrendszer tervezőjének. Ezek a szubrutinok az állapotrekordok és a szövegtábla kezelésére szolgálnak. Feladatuk az, hogy segitségükkel a vezérlő program oly módon hivhassa le az állapotrekordok vagy a szövegtábla valamelyik elemét,

hogy a programtervezőnek ne kelljen ismernie sem az ITAB tömb tényleges helyét a memóriában, sem a belső szerkezetét. Három csoportra oszthatjuk ezeket a rutinokat:

a/ Logikai állapotrekord-kezelő rutinok

Ezek a rutinok az állapotrekordok "értékes" elemeit veszik csak figyelembe, tehát az akciós rész hosszát, az intervallumok számát, illetőleg az egyes intervallumok hosszát jelölő elemeket figyelmen kivül hagyják. Ezek:

- a JAKP /IR, N, L/ függvényeljárás, amely az IR-ik állapotrekord akciós részének N-ik elemét adja függvényértékként, L-be az akciós rész hossza iródik be;
- a JELP /IR, INT, N, L/ függvényeljárás, amely az IR-ik állapotrekord elemző részének INT-ik intervalluma N-ik elemét adja függvényértékként, L-be az intervallum hossza kerül;
- a NER /IR/ függvényeljárás IR-ik állapotrekord intervallumainak a számát adja függvényértékként.

b/ Formai állapotrekord-kezelő rutin

az IARF /IR, N, L/ rutin az IR-ik állapotrekord N-ik fizikai elemét adja. Ez lehet egy akciós rész vagy elemző rész elem, esetleg valamelyiknek a hosszát jelölő elem; L-be az állapotrekord teljes hossza kerül.

c/ Szövegtábla-kezelő rutin

A MOVES /IPUF, I, ISP/ eljárás a szövegtábla ISP sorszámu szövegét olvassa be az IPUF tömb I-ik elemétől kezdve, majd I értékét az IPUF tömb következő szabad elemére állitja. Ez utóbbi elősegiti azt, hogy az IPUF tömbben esetleg több szövegrekordból állitsuk össze a kiirandó kérdést.

Feltünhet, hogy az alapvető modulok között nincsenek I/O rutinok, valamint az is, hogy az emlitett rutinok mind az állapotrekordokból vagy szövegtáblából információ olvasásra szolgálnak, és nincsenek ezekkel szimmetrikus irási rutinok.

Az első problémára az a válasz, hogy I/O rutinokat azért nem adunk meg, mert nem akarunk kötődni semmilyen konkrét perifériához, másrészt a dialógusprogram tervezője feltehetően olyan számitógépes rendszerben fog dolgozni, amelynek saját I/O rutinjai vannak, igy célszerüen azokat használhatja.
A második problémára pedig azt válaszolhatjuk, hogy irási rutinok csak dinamikus dialógusprogramoknál szükségesek, mi pedig - mint emlitettük jelen dolgozatunkban ezekkel nem foglalkozunk.

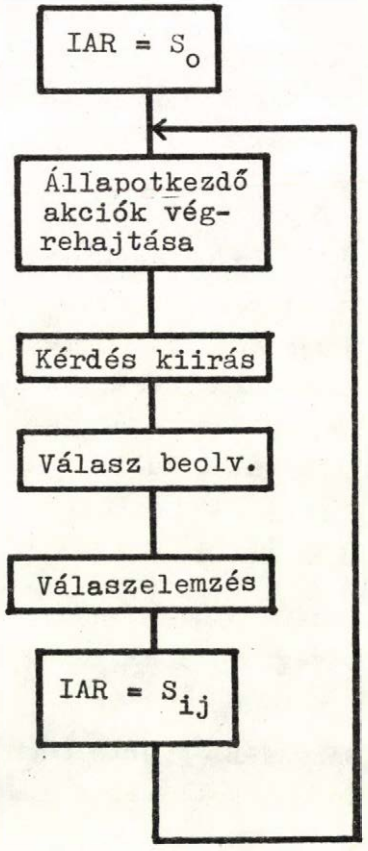
A fentieken kivül a DISTAR rendszer még több szubrutint tartalmaz, amelyeket a dialógustervező vagy használ, vagy a saját szubrutinjaival helyettesit.

Az IANALS rutin szolgál például a válasz elemzésére, a szövegtáblában tárolt szövegekkel való összehasonlitásra és az uj állapotrekord cimének meghatározására. Az IANALS használata további konvenciók betartását irja elő az állapotrekord elemeinek felhasználására vonatkozóan.

2.4 A szervezőprogram általános felépitése

A szervezőprogram vázlatos felépitése a 4. ábrán látható. Az IAR állapotregiszter tartalmazza a mindenkori állapotrekord kezdő cimét.

Az állapotkezdő akciók egyrészt a minden állapot elején végrehajtott müveletek és az adott állapotrekord akciós része által kiváltott akciók. Az akciós rész tartalmazza a kiirandó kérdés szövegének cimét is. A DISTAR rutinokkal a kiirandó kérdés átvihető egy pufferba, ahonnan a tervező output rutinja irja ki.
A választ is a tervező egy rutinja olvassa be egy másik pufferba. Ezt a szöveget az aktuális állapotrekord elemző része alapján a DISTAR rutinok segitségével lehet vizsgálni.
Az elemzés végén egy ujabb állapotrekord-cimet nyerünk, melyet be kell irni IAR-be és kezdődhetnek is ujra az állapotkezdő akciók.



4. ábra

3. EGY PÉLDA: FIDI /FIXVÁLASZU DIALÓGUSRENDSZER/

Az 1. fejezetben megmutattuk, hogy a DISTAR-B egy absztrakt dialógusgép felépitésének koncepcióját határozza meg egyrészt, másrészt egy konkrét célu, rögzitett tipusu dialógusgép tervezőjének rendelkezésére bocsátja a gép alapvető moduljait, néhány opcionális modult és egy általános /valamennyi DISTAR-B rendszerü, konkrét dialógusgépen használható/ loader programot.

Ebben a fejezetben példaként specifikálunk egy ilyen konkrét célu, rögzitett tipusu dialógusgépet, bemutatjuk ezen dialógusgép tervezésének, ill. megvalósitásának menetét, /és ezen keresztül a müködését/, végül közöljük a dialógusgép "vezérlőegységet" leiró FORTRAN programot.

3.1 A FIDI dialógusgép specifikációja

A FIDI dialógusgépen olyan programok irhatók, melyekben a dialógusgép kezelője /továbbiakban operátor/ a dialógus egyes állapotaiban a gép által feltett kérdésre egy adott /állapotonként általában különböző/ válaszhalmazból választhatja ki aktuális feleletét. Az egyes kérdésekre adott válaszok alapján a dialógus továbbfutási lehetőségei a program /dialógus leirás/ által eleve meghatározottak. A dialógusgép konzolperifériája alfanumerikus display. A válaszok külső formája háromféle lehet:

- szöveges válasz
- választás egy menüből az alfanumerikus display képernyője adott pontjának identifikálásával /fénytoll, cursormozgatás/
- választás egy menüből az alfanumerikus display képernyője adott sorának identifikálásával.

Tulajdonképpen létezik még egy válasz-forma. Ha az operátor olyan választ ad /a fent emlitett három formátum bármelyikének is felel meg formailag/, amely nem tagja az adott állapothoz rendelt válaszhalmaznak, ez u.n. "hibás" válasznak minősül. A dialógusprogram irója ehhez a speciális válaszhoz is megadhatja azt az állapotrekordot, amelyikkel a dialógust ez esetben folytatni kell.

3.2 A FIDI dialógusgép megvalósitása, müködése

Első lépésként meg kell határoznunk, hogy hogyan kell irni a FIDI gépre a programot, tehát a szövegrekordokat és állapotrekordokat.

A szövegrekordokról a DISTAR-B egyértelmüen rendelkezik, ezt minden dialógusprogramnál egyformán kell megadni, tehát:

n H c_1 ... c_n alakban /lásd 2. fejezet/.

Az állapotrekord elemei nem ennyire meghatározottak /a DISTAR-B által/, tehát itt már szükséges a pontosabb specifikáció. A FIDI négyféle állapotrekordot fogad el, ennek értelmében az állapotrekordok első elemében jelölni kell az állapotrekord tipusát.

Az akciós rész mind a négy állapotrekordban megegyzik, alakja a következő:

lA
t, KC_1, S_1 ... KC_n, S_n, KC_{n+1}

Az egyes betük jelentése:

l az akciós rész elemeinek száma, értéke: 2n+2
A jelzőkarakter /lásd 2. fejezet/
KC_i cursorcim
S_i szövegtábla-pointer
"t" értékét - mivel az egyes tipusoknál csak az elemző részben van különbség - az elemző rész feldolgozásánál vesszük figyelembe.

A "KC_i, S_i" számpárok a kiirandó kérdést határozzák meg. "S_i" a kiirandó kérdés i-ik szövegrésze, KC_i pedig az a cursorcim, amely meghatározza ezen szövegrész helyét a display képernyőjén. KC_{n+1} cursorcim pedig megadja, hogy a teljes kérdés kiirás után a cursor hova álljon be.

A négy állapotrekord-tipusnak megfelelően az elemző részek az alábbi módon alakulnak.

1. Szöveg-válasz tipus /tipus száma: 1/

/n+1/E
1I
a_h
2I
$S_1 a_1$
....
2I
$S_n a_n$

Ezt a tipusu állapotrekordot akkor használjuk, ha a kérdésre szövegválaszt várunk.

a_n - állapotrekord mutató, ebbe az állapotba kerül a dialógus, ha a válasz hibás

S_i, a_i - számpár, ahol S_i szövegrekord - a_i állapotrekord-mutató. Ha a válasz az S_i sorszámu szövegrekorddal egyezik meg, akkor a dialógus az a_i állapotban folytatódik.

Ha az S_i-k között vannak olyanok, amelyek szomszédos szövegrekordokat jelölnek, akkor az S_i a_i számpárok összevonhatók egy intervallumba oly módon, hogy csak a legkisebb sorszámu szövegrekord-mutatót kell megadni az alábbi módon:

/n+1/I

S_i a_1 a_2 ... a_n

A fenti intervallum ekvivalens az alábbi intervallumsorozattal:

2I

S_i, a_i

2I

S_{i+1}, a_2

...

2I

S_{i+n+1}, a_n

2. Cursorcim-válasz tipus /tipus száma: 2/
Ezt a tipusu állapotrekordot akkor használjuk, ha az operátor a képernyő adott pontjának identifikálásával válaszol.
Az állapotrekordok külső alakja megegyezik az előzővel, csak itt S_i adott cursorcimet jelent, amely az ernyő egy adott pontját identifikálja.

3. Sorcim-válasz tipus /tipus száma: 3/
Ezt a tipusu állapotrekordot akkor használjuk, ha az operátor a képernyő adott sorának identifikálásával válaszol. Az állapotrekordok külső alakja megegyezik az előzőekkel, S_i itt sorcimet jelent, amely az ernyő egy adott sorát identifikálja.

4. Stop válasz /tipus száma: 4/
Ebben az esetben az akciós részben meghatározott kérdés kiiródik, válaszelemzés nincs, a dialógusprogram leáll.

Miután a dialógusprogram utasitásait megterveztük, a DISTAR-B alapmoduljait /esetleg opcionális moduljait/ felhasználva meg kell irnunk a szervezőprogramot.

A FIDI szöveg- és állapotrekordokat az 5. ábra foglalja össze.

a/ FIDI szövegrekord:

$$n\ H\ c_1\ c_2\ \dots\ c_n$$

b/ FIDI állapotrekord akciós része:

$$n\ A\ t\ a_1\ S_1\ a_2\ S_2\ \dots\ a_k\ S_k\ a_{k+1}$$

n : az akciós rész hossza; n = 2k+2
A : jelző karakter
t : állapotrekord tipus; t = 1,2,3,4
a_i: a képernyőre vonatkozó cursorcim
S_i: szövegtábla-mutató

c/ FIDI állapotrekord elemző része:

$$m\ E\ 1\ I\ h\ 2\ I\ z_1\ b_1\ 2\ I\ z_2\ b_2\ \dots\ 2\ I\ z_r\ b_r$$

m : az elemző rész intervallumainak hossza
E, I: jelző karakterek
h : hibás válasz esetén következő állapot sorszáma
z_i: lehetséges válasz elemzése adja az uj állapot rekord sorszámát
b_i: az ilyen válasz esetén következő állapot-rekord sorszáma

5. ábra

A FIDI rendszert VIDEOTON 1010B számitógépre és az ORION alfanumerikus display kisérleti berendezésre valósitottuk meg. Első tervezési lépésként azonban a szervezőprogramot CDC 3300-as gép FORTRAN-jában irtuk meg, az alfanumerikus display-t pedig kártyaolvasóval és sornyomtatóval szimuláltuk. Ezt a FORTRAN programot mutatja a 7. ábra.

A DISTAR-B alapmoduljain kivül felhasználtuk az IANALS és az IANALD opcionális modulokat, és a display inputját és outputját kezelő DIN és DOUT programokat. Ezen programok specifikációját a 6. sz. ábra mutatja.

IANALD /ISTAT, IVAL, N/ - direkt analizis rutin
ISTAT: állapotrekord sorszáma
IVAL: választ tartalmazó buffer
N: válasz hossza
Az intervallumok első elemeit összehasonlitja IVAL tartalmával és ennek alapján IANALD-ben adja az uj állapotrekord sorszámát

IANALS /ISTAT, IVAL, N/ - szövegtáblás analizis rutin
ISTAT: állapotrekord sorszáma
IVAL: választ tartalmazó buffer
N: válasz hossza
Az intervallumok első elemeit szövegtábla-mutatónak tekinti, majd a szövegtábla adott elemeivel hasonlitja össze IVAL tartalmát és ennek alapján IANALS-ben adja az uj állapotrekord sorszámát

DIN /ISC, PC, IVAL, N/ - display input
ISC: ernyő-sor cime
IPC: ernyő-oszlop /pozició/ cim
IVAL: válaszbuffer
N: válasz hossza
valamennyi paramétert a rutin állitja

DOUT /SC, PC, IVAL, N/ - display output
ISC: ernyő-sor cime
IPC: ernyő-oszlop cime
IVAL: üzenet-buffer
N: üzenet hossza
valamennyi paramétert a hivóprogram állitja

6. ábra

Rutinspecifikációk

```
C   FIXVALASZU DIALOGUS VEZERLOPROGRAM
    PROGRAM FIDI
    COMMON ITAB (500)
    DIMENSION IUZEN (64)
    ISTAT = 1
C   AKCIOS RESZ FELDOLGOZASA
1   CALL JAKP (ISTAT, 1, L)
    K = L-1
    DO 2 J = 1, L, 2
    ISC = JAKP (ISTAT, J, L) /100
    IPC = JAKP (ISTAT, J, L) - ISC * 100
C   FENTI KET UTASITAS A CURSORCIMET BONTJA SOR
C   ES POZICIOCIMRE
    I = 1
    CALL MOVES (IUZEN, I, JAKP ISTAT, J+1, L)
    CALL DOUT (ISC, IPC, IUZEN, I-1)
  2 CONTINUE
C   ELEMZO RESZ FELDOLGOZASA
    CALL DIN (ISC, IPC, IVAL, N)
    GO TO (3, 4, 5, 6) JAKP (ISTAT, 1, L)
  3 ISTAT = IANALS (ISTAT, IVAL, N)
    GO TO 1
  4 IVAL = 100 * ISC + IPC
    ISTAT = IANALD (ISTAT, IVAL, 1)
    GO TO 1
  5 ISTAT = IANALD (ISTAT, ISC, 1)
    GO TO 1
  6 STOP
    END
```

7. ábra

4. DIALÓGUS SZUBRUTIN-TECHNIKA ÉS DIALÓGUS SZEGMENTÁLÁS

Ebben a részben a DISTAR rendszer két olyan szolgáltatásával fogunk foglalkozni, amelyek már a DISTAR-B bővitésének tekinthetők, és egy flexibilisebb, hatékonyabb dialógus leirási lehetőséget tesznek lehetővé.

4.1 Dialógus szubrutin-technika. Az ISUBR szubrutin

Egy nagyobb dialógus-gráfon belül általában mindig találhatók olyan részgráfok, amelyek lényegében teljesen azonos funkciót töltenek be, illetve a különbség közöttük véges számu, szabadon választható paraméterrel áthidalható. Jogosan merül fel az igény, hogy a közönséges programozástechnika szubrutinhivási lehetőségét a DISTAR rendszeren belül is biztositani kellene. Ezt a problémát a következőképpen oldottuk meg:

A szubrutintechnika bevezetése mindig három probléma megoldását jelenti:

a/ hogyan lehet szubrutint definiálni,
b/ hogyan lehet a szubrutint hivni,
c/ hogyan lehet visszatérni a szubrutinból.

Ad a/ Szubrutin definiálása a következőképpen történhet: Egy dialógus szubrutin két vagy több állapotrekordból áll, ahol az első állapotrekord a szubrutinfej, a többi a szubrutin-törzs /8. ábra/.

A szubrutin-fej egy tipuselem nélküli, csak akciós részből álló állapotrekord, amely a következő alaku:

n A e p_{11}, p_{12}, p_{22} ··· p_{k1}, p_{k2} ahol
OE

n - az akciós rész hossza: n = 2k+1
e - a szubrutin-törzs első /bemeneti/ állapotrekordja
p_{i1}, p_{i2} - az i-ik formális paraméter helyét jelöli ki, p_{i1} a megfelelő állapotrekord sorszáma, p_{i2} az állapotrekordon belül az elem sorszáma.

A szubrutin-törzs lényegében teljesen közönséges állapotrekordokból áll, azzal a különbséggel, hogy az állapotrekordok bizonyos elemei formális paraméterek, ezek helyére a szubrutin definiálásakor nem kerül értékes elem, ezek a szubrutin hivásakor kerülnek kitöltésre.

Ad b/ A szubrutinhivó állapotrekord egy O tipusu állapotrekord /az akciós rész 1. eleme: O/, amely csak akciós résszel rendelkezik és a következő alaku /lásd 8. ábra/:

n A O f, a_1, a_2, ... a_{n-1} ahol
OE

n - az akciós rész hossza
O - szubrutinhivás-tipust jelez
f - a szubrutinfej-állapotrekord sorszáma
a_i - az i-ik aktuális paraméter értéke.

Ad c/ Igazi visszatérésről itt nem beszélhetünk, hogy a szubrutin után melyik állapotrekordot kell "végrehajtani", az az operátor válaszától, ill. a válaszelemzéstől függ.

Ezen állapotrekordok sorszámai a szubrutin-törzs megfelelő állapotrekordjaiba vannak beirva, ill. mivel az esetek többségében formális paraméterek, szubrutin-hiváskor iródnak be. /Ez a visszatérés az Algol-60 label tipusu specifikációjával hozható analógiába/.

Ha a dialógus-tervező a szubrutin lehetőséget használni akarja, akkor a szervező programjában, mielőtt az éppen aktuális állapotrekord feldolgozásához hozzákezd - hivnia kell az ISUBR/N/ függvényeljárást, amely a következőket végzi el:
A N. sorszámu állapotrekordot megvizsgálja, ha nem szubrutinhivó állapotrekord ISUBR = N. Ha szubrutinhivó állapotrekord, a következőket teszi: a szubrutinfej állapotrekord alapján a szubrutin-törzsbe /a p_{i1}, p_{i2} helyeken/ beirja az aktuális paramétereket /a_i/, majd ISUBR értéke a szubrutin első állapotrekordjának sorszáma /e/ lesz.

Látható tehát, hogy ISUBR értéke /hivás után/ mindig annak az állapotrekordnak a sorszáma lesz, amit éppen fel kell dolgozni.

4.2 Szegmentált dialógusok

Nagyméretü dialógusok esetén előfordulhat, hogy a teljes állapottábla és szövegtábla nem fér be a memóriába, ekkor a dialógusgráfot alkalmas pontokon részgráfokra kell bontani, az igy kialakitott szegmenseket háttértárolón /pl. mágneslemezen/ tárolni, és gondoskodni kell arról, hogy mindig az éppen "éles" szegmens legyen benne a memóriában. Nézzük, milyen lehetőséget biztosit a DISTAR rendszer ehhez a dialógus-szegmentáláshoz.

1. A mágneslemezes loader /DINFAZ/. Ez a loader az INFAZ loader szolgáltatásain kivül még elvégzi az egyes szegmensek lemezre töltését, valamint létrehoz egy directory-t, amely az egyes szegmensek diszk kezdő cimét tartalmazza. /Ezt a directory-t a DISTAR-B rutinjai használják/.

2. A csatoló-állapotrekord /lásd 8. ábra/.
Ha valamelyik állapotrekordban olyan állapotrekordra akarunk hivatkozni, amelyik nem a core-ban lévő szegmensben fordul elő, akkor egy u.n. csatoló-állapotrekordra kell előbb hivatkozni, amely információt tartalmaz a behivandó szegmensre nézve. /A csatoló rekordot természetesen az aktuális szegmensben kell elhelyezni/. A csatoló-állapotrekord egy -1 - tipusu rekord, aminek csak egy háromelemü akciós része van, amely a következőképpen néz ki:

```
3A -1, s, a
OE          ahol
```

3 : az akciós rész hossza
-1 : a csatoló-állapotrekordot jelző tipusszám
s : a behivandó szegmens sorszáma
a : a szegmensen belül annak az állapotrekordnak a sorszáma, amire a "vezérlés" átadódik.

3. Az ICSAT /N/ szubrutin
Ha a dialógus-tervező szegmentált dialógus programot ir, az állapotrekord feldolgozása előtt - az ISUBR rutinhoz hasonlóan - hivnia kell az ICSAT függvényeljárást, amely a következőket végzi el:

Ha az N. állapotrekord nem csatoló-rekord, ICSAT = N, ha igen, betölti a memóriába /a csatoló-rekord alapján/ a hivandó szegmenst /s/ és ICSAT értéke a szegmens megfelelő állapotrekordjának sorszáma lesz /n/. Látható tehát, hogy ICSAT értéke is /hivás után/ mindig annak az állapotrekordnak a sorszáma lesz, amit fel kell dolgozni, csak közben - ha szükséges - szegmentálás történik.

Szubrutinfej:

n A e, p_{11}, p_{12} ··· p_{k1}, p_{k2}

OE

n: akciós rész hossza, n + 2k+1

e: a szubrutintörzs első állapotrekordja

p_{11}, p_{12}: az i-ik formális paraméter helye

Szubrutinhivás:

n A O, f, a_1, ··· a_{n-1}

n : az akciós rész hossza

O : szubrutinhivás-tipust jelez

f : a szubrutinfej-állapotrekord

a_i: az i-ik aktuális paraméter

Csatoló állapotrekord:

3 A -1, s, a

-1: a csatoló-rekord tipust jelzi

s : a behivandó szegmens sorszáma

a : a szegmensen belüli aktivizálandó állapotrekord.

8. ábra

5. A DISTAR-B RENDSZER ÉRTÉKELÉSE

A DISTAR-B rendszert sikerrel alkalmaztuk különböző alfanumerikus display demonstrációs programokban és egy speciális gépipari tervező dialógus-program előállitására [1], [2].

A rendszer jelenlegi állapotában statikus és pszeudo-dinamikus dialógusok előállitására használható, a tervező által irt input-output rutinok által meghatározott perifériákon.

A dolgozatban leirtakon kivül időközben elkészült, illetőleg megvalósitás alatt áll a rendszer szélesebb körü és jobb hatásfoku használatát elősegitő néhány program, ill. programcsomag.

Eddig az alkalmazásoknál a szövegtáblát és az állapotrekordokat a 2. pontban ismertetett direkt kódban kellett megadni. Jelenleg megvalósitás alatt áll egy dialógus assembly nyelv, és fordító programja, amely lehetővé teszi többek között az egyes szövegekre, ill. állapotokra való szimbólikus hivatkozást. Az assembly nyelvnél figyelembe vettük a szegmentálási és szubrutin-irási lehetőségeket is.

Mindezeken kivül tervezzük az alapvető programmodulok bővitését paraméterkezelő rutinokkal. Ezek lehetővé teszik paraméteres dialógusok előállitását, amelyekben a válaszok rögzitett alapszavakon kivül numerikus- és szövegparamétereket is tartalmazhatnak.

Tervezzük továbbá, hogy a paraméter-válaszok és egyéb közbenső adatok tárolására létrehozott munkamezőt "freecore"-ként lehessen kezelni, és ehhez kifejlesztünk "allocate" és "release"-rutinokat, valamint garbage collection programot.

A szövegtábla és az állapottábla dinamikus módositásával adaptiv rendszerekben is használható a DISTAR-B felfogás. Hasonló bővitések szükségesek a CAI rendszerekben való felhasználáshoz.

A DISTAR-B rendszer a dialógus menetét állapotrekordokkal irja le. Az állapotrekordok tulajdonképpen egy speciális adatstruktura cellái. Meg lehetne vizsgálni, hogy az állapotok leirására valamilyen listakezelő rendszer mennyiben használható.

A dialógus menetét döntési táblákkal is le lehetne irni. A két módszer között gazdaságossági összehasonlitást eddig nem végeztünk.

A dialógusok elméleti vizsgálatánál jól alkalmazhatók az automata-elmélet módszerei, hiszen az állapotrekordok egy dialógus-automata állapotait irják le. A dialógusok leirásának ez a módja ekvivalens /leképezhető/ egy Moore automatával.

Végezetül összefoglalnánk, hogy véleményünk szerint melyek a DISTAR-B rendszer legfőbb előnyei:

a/ Az absztrakt dialógusgép fogalma lehetővé teszi, hogy a tervező valamely dialógusprogram irásánál, csak magára a dialógusra koncentráljon, azaz megfelelő szintaktikus szabályok betartásával magát a dialógust irhassa le. Ezáltal mentesit számos programozástechnikai munkától.

b/ A szöveg- és állapottábla lehetővé teszi a DISTAR-B-beli rendszerben a dialógusprogramok módositását, ill. bővitését.

c/ A dialógusprogram felosztása dialógus leirásra és dialógusvezérlő programra lehetővé teszi, hogy a nagyméretü dialógusok esetén a vezérlő programot rezidensként a core-ban tartsuk, az állapot- és szövegrekordok overlay-je viszont az előzőekben ismertetett módon egyszerüen elvégezhető.

d/ A DISTAR-B rendszer moduláris felépitésü, ami egyrészt lehetővé teszi, hogy a dialógus tervezője a saját céljaira legalkalmasabb dialógusgépet állitsa elő, másrészt magának a rendszernek a bővitésére is egyszerü lehetőséget biztosit.

e/ A DISTAR-B rendszer koncepciója segitséget nyujt a dialógusprogramok, interaktiv programrendszerek elméleti vizsgálatainál.

IRODALOMJEGYZÉK

[1] Gy.Pikler: A Minicomputer Based Conversaitonal Program Writing System. 2d International Conference on Programming Machine Tools, Budapest, 1973.

[2] T.Forgács, Gy.Hermann, Gy.Pikler: Software Methods in Design of CAD-programs.
Working Conference of IFIP: Principle of CAD Eindhaven (Holland), 1972.

[3] C.I.Johnson: Principles of interactive systems IBM Systems Journal, 7, 1968, pp. 147-173.

[4] A.von Dam, J.W.Brockett. Lecture Notes: Interactive Computer Graphics Cybex Associates Inc., Great Neck, N.Y.

[5] Forgács T., Vörös G.: Alfanumerikus display rendszerek. Mérnöktovábbképző Intézet jegyzete, 4. fejezet, Budapest, 1971.

[6] D.C.Aaronson: ICU/PLANIT The All-Purpose Machine-Transferable CAI-System. Reference Manual, System Development Corporation Santa Monica, California, 1971.

[7] Forgács T., Gerhardt G., Kocsis J., Krammer G.: DISTAR-B felhasználói dokumentáció. MTA SZTAKI belső dokumentáció, Budapest, 1970.

[8] Wolfberg M.S.: An Interactive Graph Theory System University of Pennsylvania (Moore School) Philadelphia, Pa., 1969. Technical Report No. 69-25.

[9] G.M.Weinberg: Programming and Compiling Strategies for Paging Systems, Software Practice and Experience Volume 2, No. 2, 1972.

A CHANGE NYELV IMPLEMENTÁLÁSA A MINSZK-32 SZÁMITÓGÉPEN

Megyesi Katalin

BEVEZETÉS

A dolgozat a /2/-ben ismertetett CHANGE nyelv elektronikus számitógépen való realizálásának egy lehetséges módját ismerteti.

A dolgozat három részből áll.

"A CHANGE programok belső ábrázolása" cimü fejezetben a CHANGE programok belső strukturájának, az egyes utasitások és paramétereik belső ábrázolásának részletes ismertetésére kerül sor.

A második fejezet az assembly formátumu CHANGE utasitások szintaktikáját tárgyalja.

A harmadik fejezet a MINSZK-32-es elektronikus számitógépen realizált CHANGE nyelv fordító és értelmező rendszerét ismerteti.

A CHANGE PROGRAMOK BELSŐ ÁBRÁZOLÁSA

A belső struktura rendkivül tömör. Ha egy utasitás vagy egy paraméter a programban többször is előfordul, leirása csak egyszer kerül tárolásra.

Ez az ábrázolás megkönnyiti

- a programok lineáris végrehajtási módjának feloldását,
- a programot módositó utasitások végrehajtását,
- memória nyereséget tehet lehetővé és
- időmegtakaritást eredményez a nyelv jellemző és gyakorlati utasitásainál /a végrehajtási módot vezérlő utasitások, CHANGE utasitások/.

A belső struktura hat főbb tömbre épül.

Ezen tömbök:

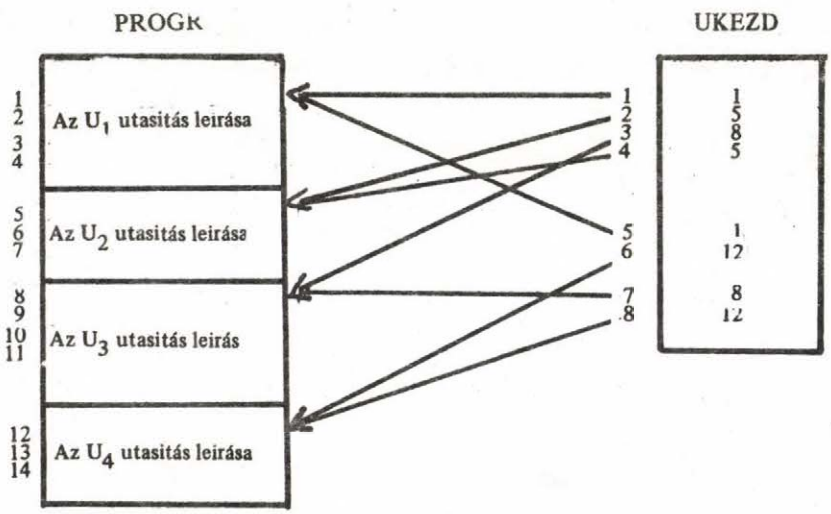
MCIM, PARA, PCIM, PROGR, UKEZD, CIMKE

A program minden utasitásának egy egyértelmüen meghatározott belső sorszáma van. /A program i-dik utasitásának belső sorszáma i./ Az UKEZD tömb i-dik elemében az i-dik utasitás PROGR tömbbeli leirásának kezdőcime található.

1. Például az

U_1
U_2
U_3
U_2
U_1
U_4
U_3
U_4 utasitásokból álló program esetén a

PROGR és UKEZD tömbök:

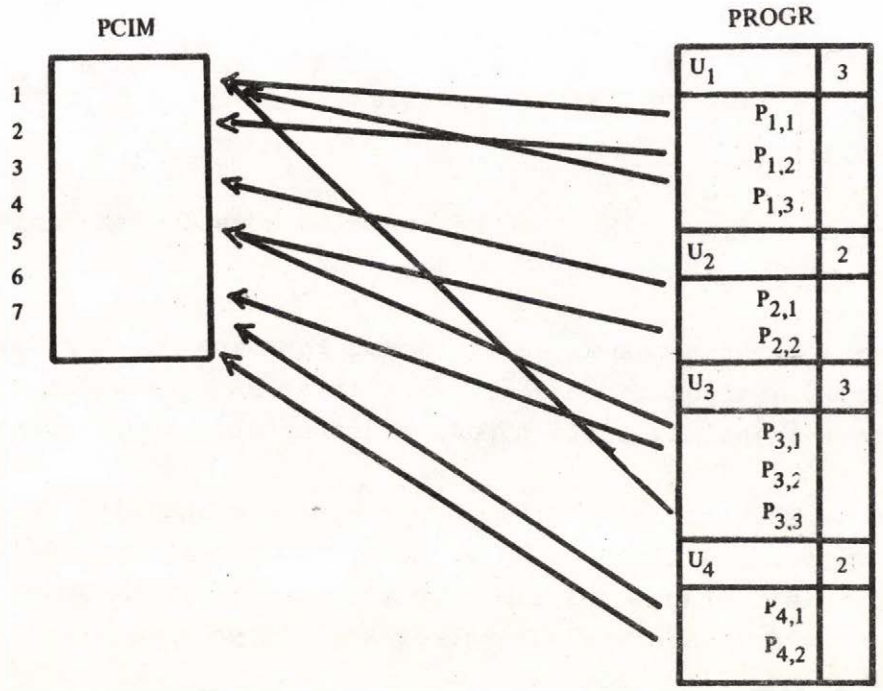


1. ábra

A PROGR tömbben a programban előforduló összes utasitás leirása megtalálható, az ismétlődő utasitások leirása csak egyszer kerül tárolásra.

Egy utasitás leirásában tárolásra kerül az utasitáskód, amely az utasitásnak megfelelő könyvtári szubrutin sorszáma /jelölés: U_i/, az utasitás paramétereinek száma /jelölés: PDB_i/ és a paraméter leirások cimeinek PCIM tömbbeli cimei /jelölés: $P_{i,k}$ az i-dik utasitás k-dik paraméterének leirására vonatkozó cim/.

A példában szereplő program esetén a PCIM és PROGR tömbök:



2. ábra

Különböző utasitások paraméterei között lehetnek azonosak is. A példában

$P_{1,1} = P_{1,3} = P_{3,3}$ $P_{2,2} = P_{3,1}$

Az utasitások kódjai között is lehetnek azonosak, például $U_1 = U_3$, de az utasitások teljes $U_{i,1}$, $P_{i,2}$, $P_{i,3}$ i=1,3 leirása különböző.

Egy PCIM tömbbeli cim egy paraméter leirásának PARA tömbbeli kezdőcimére mutat, és a paraméterben szereplő egyszerü változó nevek /tömbnevek ill. konstansok/ számát is tartalmazza. Például az a^n/3/ általános változó esetén az egyszerü változók /a,n,3/ száma három. /Attól függetlenül, hogy a változók között vannak-e egymással megegyezők. Igy az a^j/a/ és a^2/2/ egyszerü változóinak száma is három/.

A PARA tömb az egyes paraméterek egyszerü változóinak azonositóit tartalmazza jobbról balra haladva.

Egy általános változó egyszerü változói közül a kitevőket azzal különböztetjük meg, hogy a kitevőnek megfelelő egyszerü változóhoz rendelt azonositót negativ előjellel látjuk el.

Egyszerü változó azonositója pozitiv egész szám, amely az MCIM tömb egy elemének sorszáma.

Egy konstans azonositója a PARA tömb két egymást követő elemében kerül tárolásra negativ előjellel.

Az első elemben a konstansnak az adott tipusu konstanstáblán /tömbön/ belüli sorszáma kerül tárolásra, a második elemben az MCIM tömb egy elemének sorszáma található, ahol az adott tipusu konstanstábla /tömb/ cime van.

A CHANGE utasitásokban szereplő paraméterek esetén a legbelső zárójelben álló egyszerü változó, a kitevő vagy az index nélküli változó előtt * jel állhat. /Konstansleirásban * jel nem szerepelhet/. Ennek hatására CHANGE utasitásokban az utasitás aktuális értékét kell figyelembe venni. Más utasitásokban a * jelnek nincs hatása.

Valamely * jellel ellátott egyszerü változó azonositója a PARA tömb két egymást követő elemében kerül tárolásra.

Az első elemben az egyszerü változó MCIM tömbbeli sorszáma, a második elemben +0 kerül tárolásra.

Az MCIM tömb funkciója kettős. Az előforditás alatt az MCIM tömb elemeiben az egyszerü változók neve található karakter formában, negativ előjellel, illetve az alapnyelv konstanstipusainak megfelelő konstanstáblák memóriabeli kezdőcimei találhatók pozitiv előjellel.

Futás /végrehajtás/ alatt az MCIM tömb elemeiben az egyszerü változó nevek /tömbnevek/ helyére a változónak a memóriabeli kezdő cime kerül. Mivel ez nem az előforditás befejezésekor egy lépésben történik, hanem futás alatt a tipus deklaráló utasitások végrehajtásakor, ezért az egyszerü változó neve és az egyszerü változó memóriabeli cime között különbséget kell tenni.

Például:

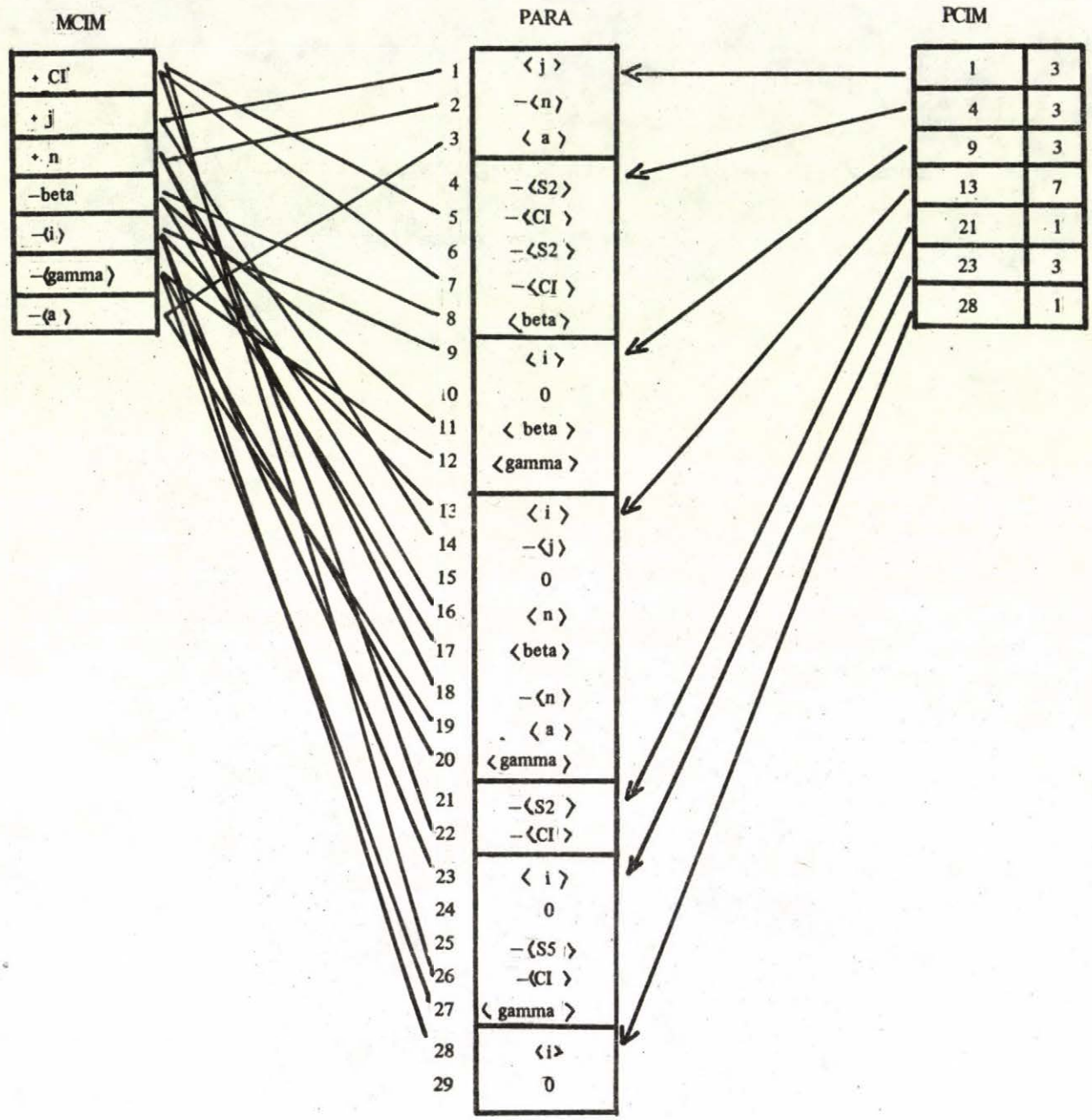
$P_{1,1}$ = a^n/j/ $P_{1,2}$ = beta^2/2/

$P_{2,1}$ = gamma /beta /*i// $P_{2,2}$ = gamma /a^n/beta/n^*j/i///

$P_{3,2}$ = 2 $P_{4,1}$ = gamma^5/*i/

$P_{4,2}$ = *i

általános változók futás alatt



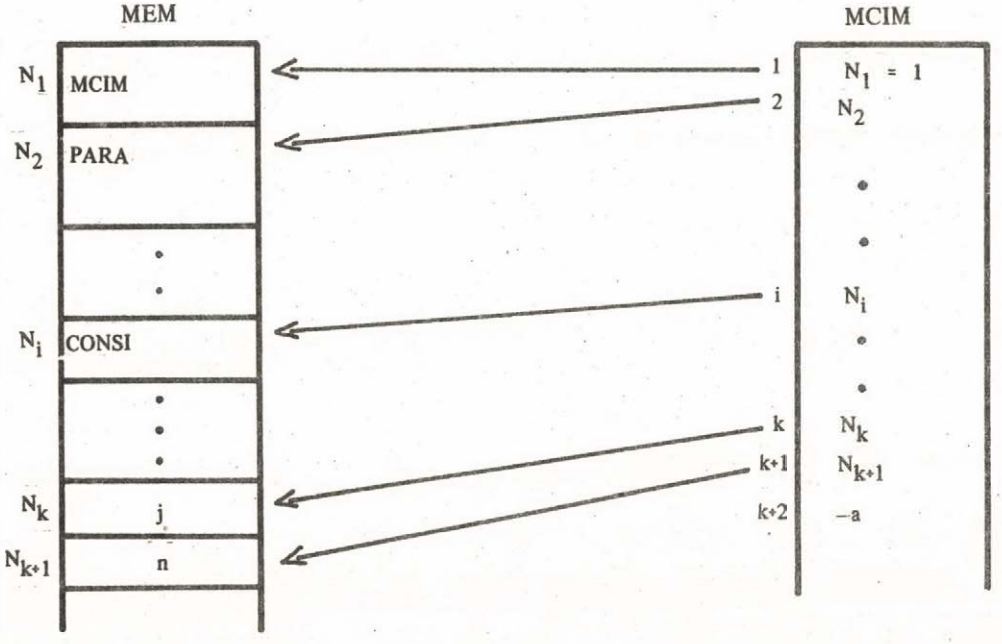
3. ábra

ahol S/k/ a k konstansnak a konstanstáblabeli cime /sorszáma/, ⟨v⟩ a v egyszerü változó /konstanstábla/ MCIM tömbbeli sorszáma.

A belső ábrázolás tömbjei a CHANGE programokban használt tömbök és a munkatömbök egyetlen rögzitett kezdőcimü MEM nevü tömbként kerülnek tárolásra a memóriában. Ez a megoldás lehetővé teszi a tömbméretek dinamikus kezelését.

A tömbökhöz való hozzáférést az MCIM tömb biztositja, amelynek kezdőcime egybeesik a MEM tömb kezdőcimével. Az MCIM tömb az összes MEM tömbön tárolt tömb kezdőcimét tartalmazza, igy saját kezdőcimét is /ezért rögzitett az MCIM tömb kezdőcime, hiszen ellenkező esetben a tömbökhöz fordulás nem egyértelmü/.

Az MCIM tömb futás alatt



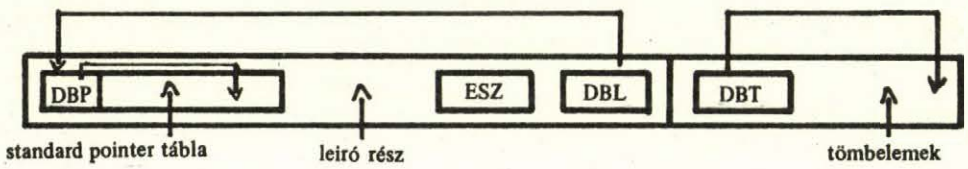
4. ábra

ahol N_1, N_2,.....N_i,.....N_k, N_{k+1} az MCIM, PARA ..... stb. rendszer tömböknek, a CONSI, CONST..... stb. konstanstábláknak és a j,n változótömböknek a MEM tömb kezdőciméhez viszonyitott relativ kezdőcimei.

Az egy tömbhöz /táblához/ tartozó memória terület egy leiró részből, és egy a tömbelemeket tartalmazó részből áll. A tömb /tábla/ MCIM tömbbeli azonositója a tömbelemeket tartalmazó rész memóriabeli kezdőcimére mutat.

A tömbelemeket tartalmazó rész első elemében a tömbelemeket tartalmazó memóriaterület elemszáma /DBT/ kerül tárolásra.

Az egy tömbhöz /táblához/ tartozó memóriaterület felosztása:



5. ábra

A leiró rész utolsó elemében a leiró rész elemszáma /DBL/ kerül tárolásra, utolsó előtti eleme /ESZ/ pedig arra utal, hogy egy-egy tömbelem hány memóriaelemben kerül tárolásra. ESZ = 0 esetén a tömbelemek változó számu memóriaelemet foglalnak el. Ebben az esetben a leiró rész mindig egy standard pointer táblával kezdődik. A standard

pointer tábla első eleme ezen tábla elemszámát /DBP/ tartalmazza, i-dik eleme az i-dik tömbelemnek a tömbelemeket tartalmazó részen belüli helyére utal /pld. relativ kezdőcimére mutat/.

A leiró rész tartalmazhat kiegészitő információkat is, amelyeket egyes szubrutinok felhasználhatnak. Ennek helye a standard pointer tábla és ESZ között van.

ESZ = 1,2, ... esetén minden egyes tömbelem 1,2..... számu memóriaelemben kerül tárolásra. ESZ≠0 esetén a leiró rész nem feltétlenül tartalmaz standard pointer táblát. ESZ≠0 esetén az i-ik tömbelem memóriabeli kezdőcime

MCIM/TÖMB/ + /ixESZ/

Egy leiró rész mindig legalább két elemet /DBL-et és ESZ-t/ tartalmaz.

A MINSZK-32-es elektronikus számitógépen az egész, a valós, a logikai és a karakter tipusu értékek egy-egy memória rekeszben kerülnek tárolásra. Ennek megfelelően ESZ=1 és a tömbökhöz /táblákhoz/ tartozó leiró rész standard pointer táblát nem tartalmaz.

A szövegtipusu értékek a szöveg karaktereinek számától függően egy, vagy több memória rekeszben kerülnek tárolásra, igy ESZ=0, és a szövegtipusu tömbökhöz /táblákhoz/ tartozó leiró rész standard pointer táblával kezdődik, és az i-dik tömbelem memóriabeli kezdőcime.

MCIM /TÖMB/ + SPT /i/ ahol SPT a standard pointer

táblát jelöli.

Az egész tipusu számok tárolása fixpontos egész számként történik.

A valós tipusu számokat lebegőpontos számokként tároljuk.

Egy logikai érték tárolása egy memóriarekeszben történik.

Egy karakter tipusu érték egy memória rekeszben a legalacsonyabb helyértékü biteken /"jobbra adjusztálva"/ kerül tárolásra.

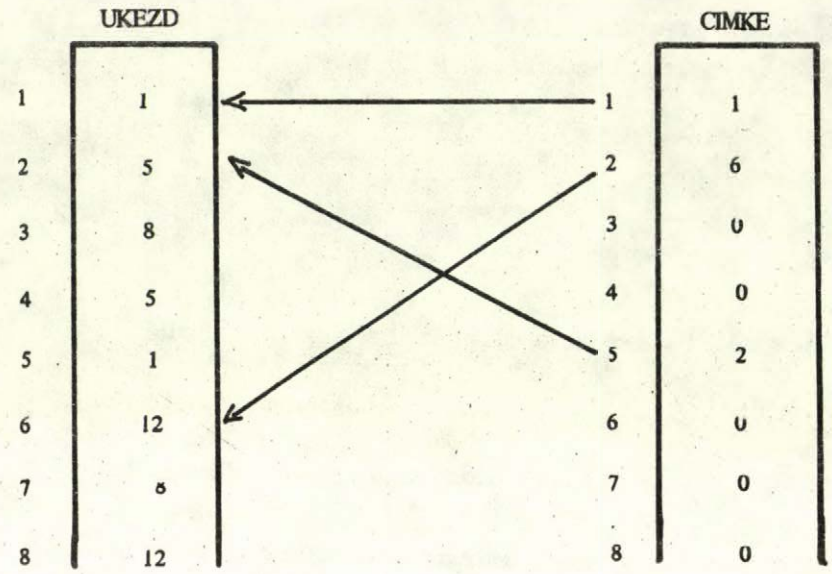
Egy szöveg tipusu érték a szöveg karaktereinek számértékétől függően egy vagy több egymást követő memória rekeszben kerül tárolásra folyamatosan.

Egy rekesz a benne elhelyezhető maximális számu karaktert tartalmazza /kivéve esetleg a szövegtipusu tömb utolsó memóriarekeszét/.

Ismétlődő konstansok csak egyszer kerülnek tárolásra.

Végül cimkével ellátott utasitások esetén /cimke tetszőleges, legfeljebb 4 számjegyből álló tizes számrendszerbeli pozitiv egész szám lehet, a 0-át kivéve/ az utasitás belső sorszáma a CIMKE tömbnek a cime által meghatározott elemében kerül tárolásra.

Például az 1. utasitás cimkéje 1
2. utasitás cimkéje 5
6. utasitás cimkéje 2



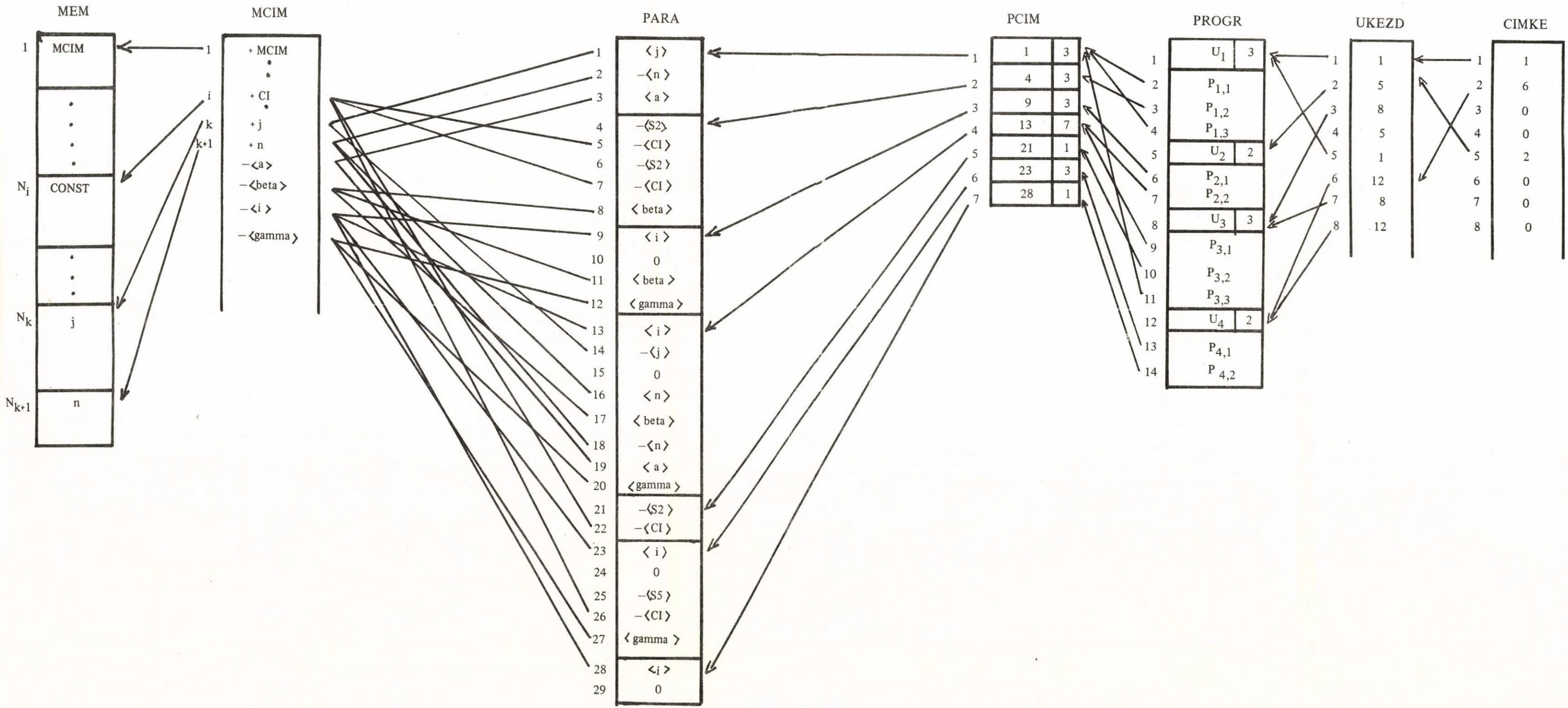
6. ábra

A CIMKE tömb azon elemeinek értéke, amelyeknek megfelelő cimke a programban nem fordul elő, 0-val egyenlő.

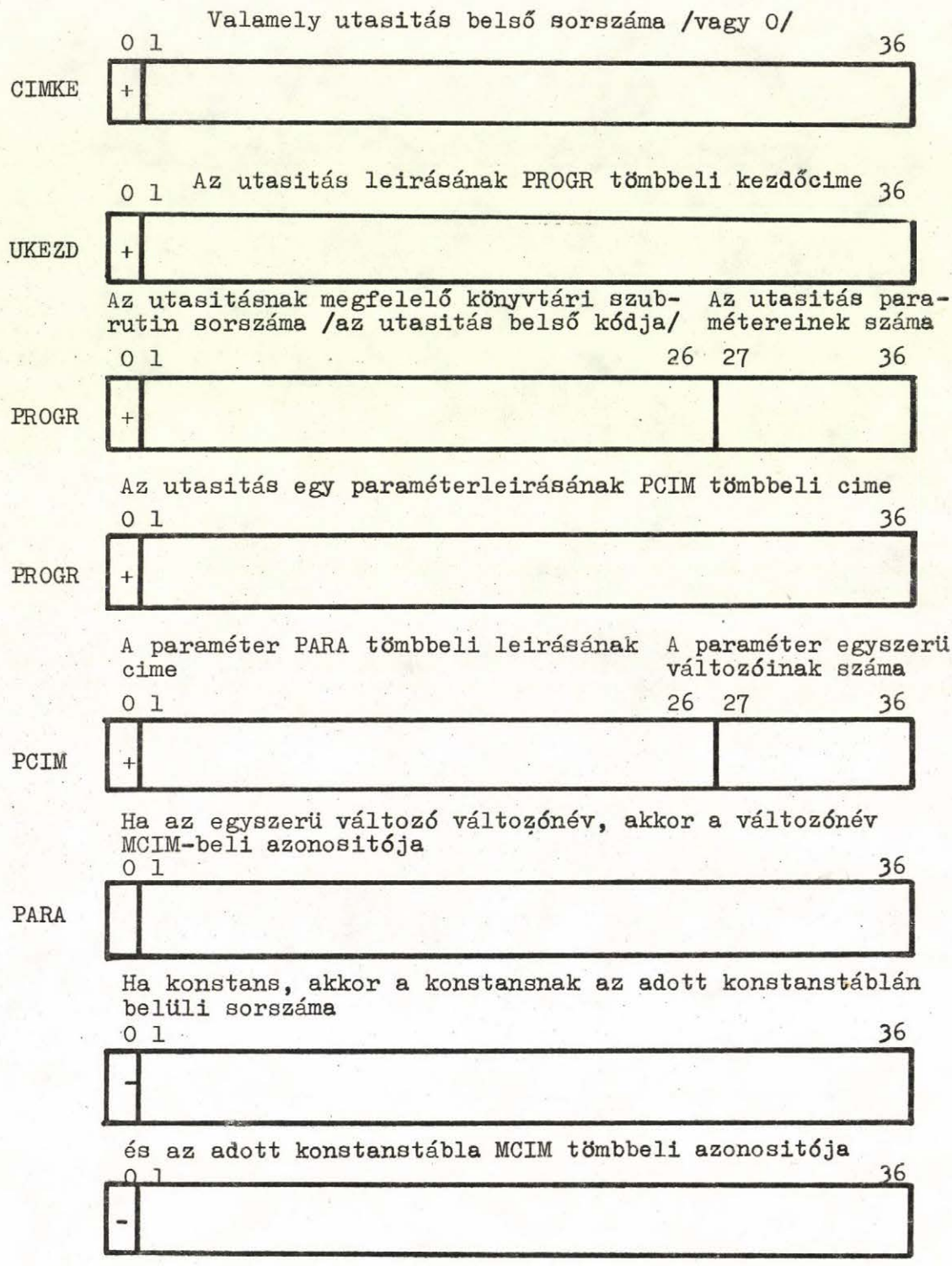
A programnak megfelelő belső tömbök közötti teljes strukturális kapcsolat a 7. ábrán, ezen tömbök információs szavainak felépitése és a nyelv adattipusainak ábrázolása a MINSZK-32-es tipusu elektronikus számitógépen a 8. ábrán látható.

RENDSZERTÖMBÖK

7. ábra

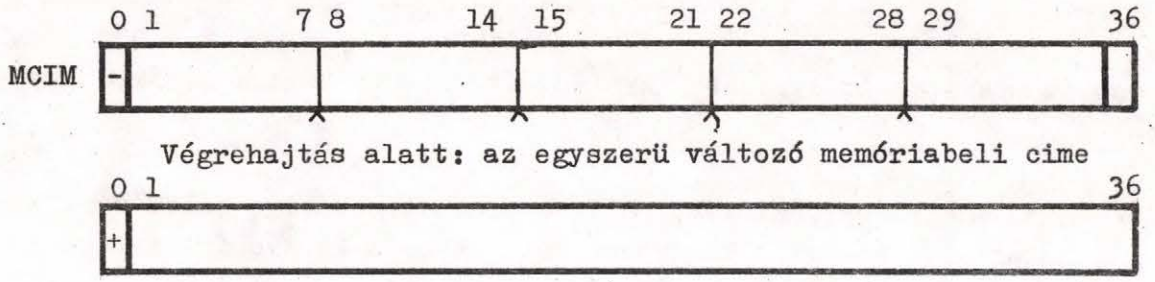


A rendszertömbök információs szavainak felépitése



8. ábra

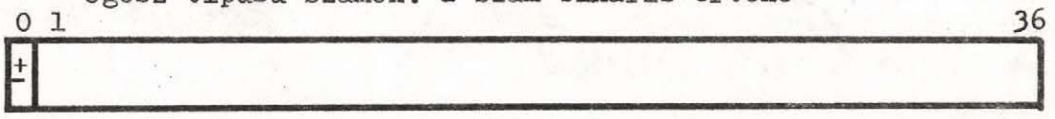
Előforditás alatt az egyszerü változó neve karakter formában



Végrehajtás alatt: az egyszerü változó memóriabeli cime

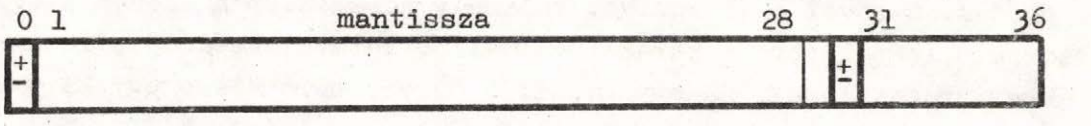
A CHANGE nyelv adattipusainak ábrázolása

egész tipusu számok: a szám bináris értéke

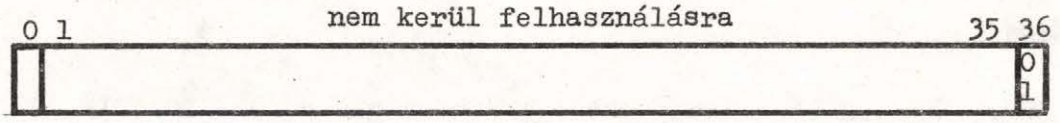


valós tipusu számok

karakterisztika

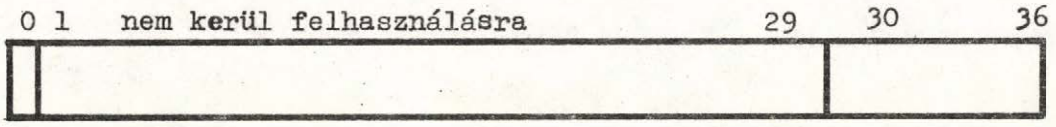


logikai érték



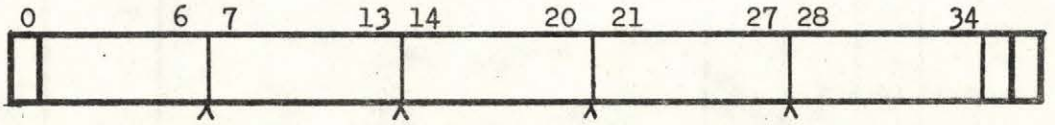
karakter tipusu érték

a karakter



szöveg tipusu érték

az utolsó két bit nem kerül felhasználásra



Megjegyzés: azokban a rekeszekben, vagy a rekeszek azon bitjein, amelyekre más feltételt nem szabtunk, egész tipusu bináris számok kerülnek tárolásra.

AZ ASSEMBLER FORMÁTUMU CHANGE UTASITÁSOK SZINTAKTIKÁJA

Egy program utasitásait lyukkártyákra rögzitjük. Egy kártya mezőkre van osztva.

Az első mező, amely egy pozicióból áll, a folytatás kártya jelölésére szolgál, ha az első pozición nem space van, akkor a kártya folytatás kártya.

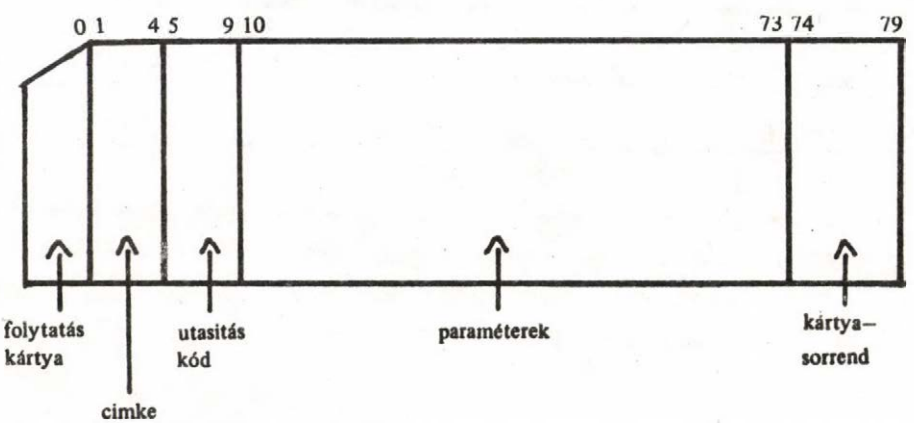
A második mező /2-5. pozició/ az utasitás cimkéjét tartalmazza, amely a második pozición kezdődik. Ha az utasitás nem cimkézett, akkor a mező space-ket tartalmaz.

A harmadik mezőn /6-10. pozició/ az utasitás kódot tároljuk. /Az utasitás kódokat lásd az 1. sz. mellékleten/.

Ezt követően az utasitás paramétereit /általános változóit/ irjuk, egymástól pontosvesszővel elválasztva. Valamely paraméterben szereplő egyszerü változónév legfeljebb öt karakterből álló, betüvel kezdődő alfanumerikus karaktersorozat lehet. Ha egy utasitás összes paramétere egy kártyán nem fér el, akkor a felsorolást folytatás kártyán folytatjuk.

Technikai jellegü meggondolásokból a kártya utolsó hat poziciójá kártya sorrend jelölésére tartjuk fenn /amelyet nem kötelező kiirni/.

A kártyasorrendet pozitiv egész számokból álló növekvő számsorozattal jelöljük.



9. ábra

Egy utasitásban szereplő paraméter tetszőleges általános változó /speciálisan konstans/ lehet.

Az általános változók szerkezete:

Egy általános változó lehet indexes vagy index nélküli. Az index lehet pozitiv egész szám, vagy pozitiv egész értékeket felvevő egész tipusu változó. Az indexet minden esetben kerek zárójelek közé kell tenni.

Például a (j (b(3))) stb.
a(b(5)) , c(b(a alfa)))
alfa (beta (gamma (3)))
a (*j), b(gamma *c)) , *i,

Egy indexes változónak lehet kitevője, amely nem-negativ egész szám, vagy nem-negativértékeket felvevő egész tipusu egyszerü változó lehet. A kitevőt az eléje tett kettősponttal különböztetjük meg.

Például: a:3(j) , b:c(i) , d:k(5) ,
k:2(2) , a:xc(i) ; beta:3(xgamma)
ahol $a{:}n(j) = a^{n}(j) = \overset{1}{a}aa\ldots\overset{n}{a}\ (j)$
$j > 0 \quad n \geqq 0$

A konstansok szerkezete:

Az egész tipusu konstansok tizes számrendszerbeli számjegyekből állnak. A legnagyobb egész tipusu szám, amely a MINSZK-32-es elektronikus számitógépen ábrázolható 2^{36}-1. Pl. 362 vagy 981.

A valós konstansok szintén tizes számrendszerbeli számjegyekből állnak. A tizedes pontot minden esetben fel kell tüntetni. A gépen ábrázolható valós tipusu számok /x/ a

$$0{,}5421010 \cdot 10^{-19} \leqq x \leqq 0{,}9223372 \cdot 10^{19}$$

tartományba esnek. Pl. 0.032 vagy 31.

A logikai konstansok: .TRUE.
.FALSE.

A karakter konstansok: A MINSZK-32-es tipusu elektronikus számitógép által használt GOSZT-10659-64 szabványban lévő első hetvennyolc karakter. Ehhez tartoznak

- az orosz ABC nagybetüi /a Ë és ъ kivételével/
- a latin ABC nagybetüi /azok kivételével, melyek megegyeznek a cirill ABC nagybetüivel/
- számok 0-tól 9-ig
- jelek + - x : , /10 ↑

  = ( ) ; [ ] *

  ' " ≠ > < ␣

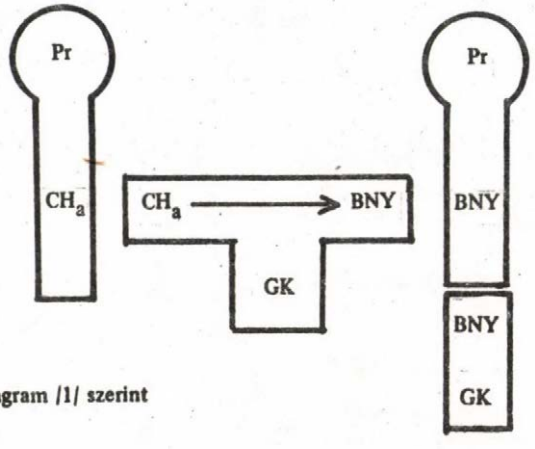
Az egyes karaktereket kezdő és záró aposztrof közé kell tenni.
/pl. 'A' '+' 'B' ':' stb./

A szövegkonstansok szögletes kezdő és végzárójel közötti tetszőleges hosszuságu karaktersorozatok. /A szögletes végzárójelek kivételével/
Pl. [MEZO = A2+A 4]

A FORDITÓ-ÉRTELMEZŐ RENDSZER STRUKTURÁJA

A nyelv utasitásai célszerüvé teszik a programok interpretativ végrehajtását. A végrehajtás nem teljesen interpretativ. Egy előforditási lépés során a programot a belső ábrázolásnak megfelelő formára /"belső nyelvre"/ forditjuk, és ennek a belső nyelvnek az utasitásai kerülnek interpretativ végrehajtásra.

A forditó-értelmező rendszer T diagramja



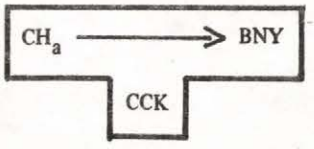
T diagram /1/ szerint

10. ábra

ahol CH_a az assembler formátumu CHANGE nyelvet, BNY a belső nyelvet, GK a belső gépi kódot jelöli.

Az előfordító program CCK /MINSZK-32 assembly / nyelvü.

Az előfordító program a CHANGE assembly formátumról /CH_a/ belső lista strukturára /belső nyelvre BNY/ fordit.



11. ábra

Az előfordító program felépiti a belső nyelvü programot, azaz a belső ábrázolás tömbjeit.

Ennek részeként a szintaktika analizátor balról jobbra haladva részekre bontja az utasitást, az egyes részeket külön-külön elemzi:

tárolja az utasitás cimkéjét,
meghatározza az utasitásoknak megfelelő
könyvtári szubrutin belső sorszámát,
felbontja az általános változókat egyszerü
változók sorozataira.

A szintetizáló programrész az igy kapott információkat beépiti a belső nyelvre forditott programba.

Szintaktikusan hibás utasitások esetén az előfordító program hibajelzést ad a hibatipus megjegyzésével.

Az előfordító program outputjai:

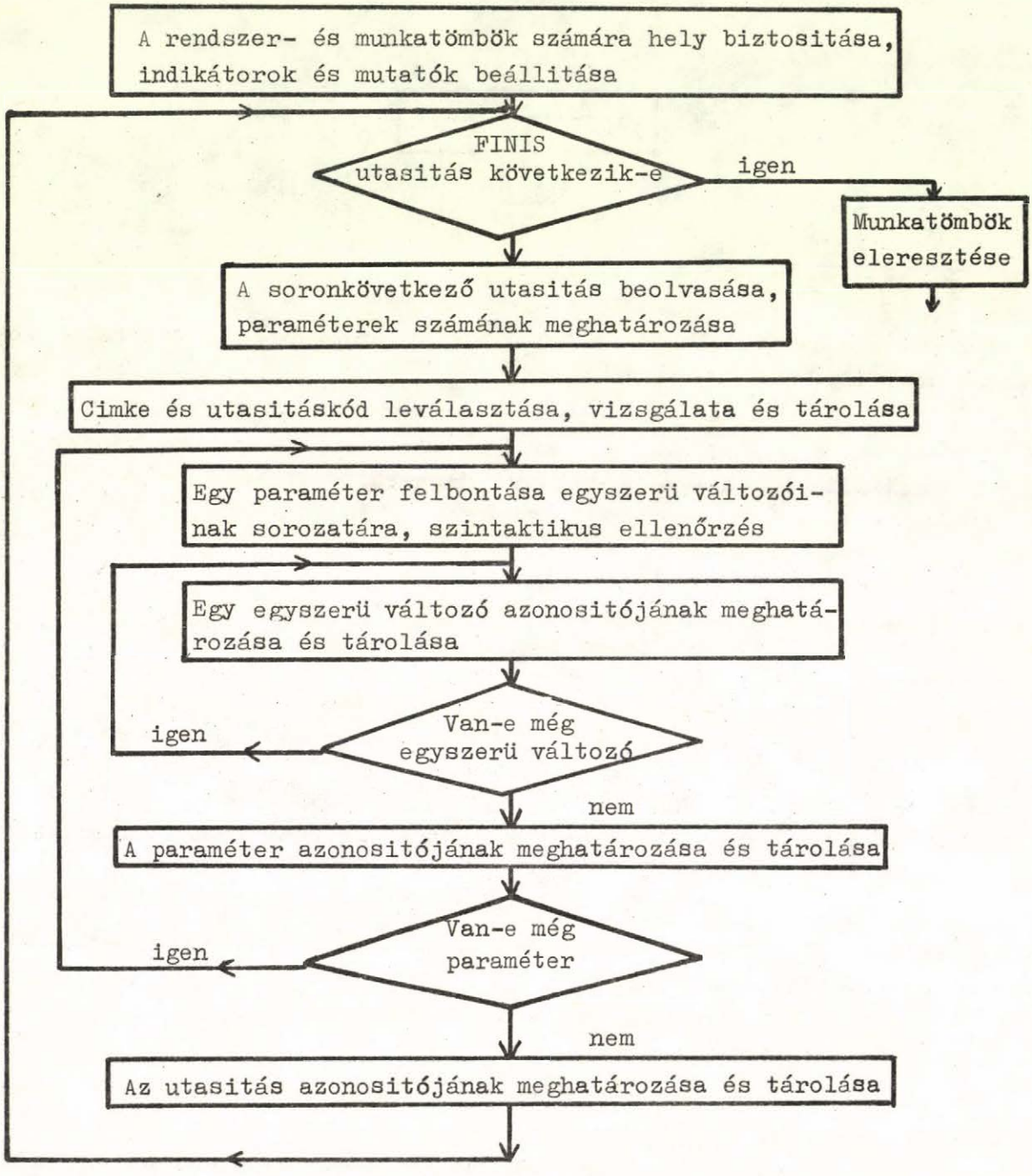
az assembly formátumu CHANGE nyelvü program belső nyelven /a felépitett rendszertömbök, konstanstáblák/

egy HIBA indikátor, amelynek értékét az előfordító program 1-re állitja, ha a leforditott programban szintaktikusan hibás utasitást talál.

A HIBA indikátor értéke szintaktikusan hibátlan program esetén 0.

A HIBA = 1 esetén az értelmező és végrehajtó program /a programozó külön tevékenysége nélkül/ nem aktivizálódik.

Az előfordító program funkcionális blokkvázlata



12. ábra

Az értelmező és végrehajtó program a belső nyelvre forditott program utasitásainak értelmezését és végrehajtását biztositja. Az értelmező és végrehajtó program CCK nyelvü

13. ábra

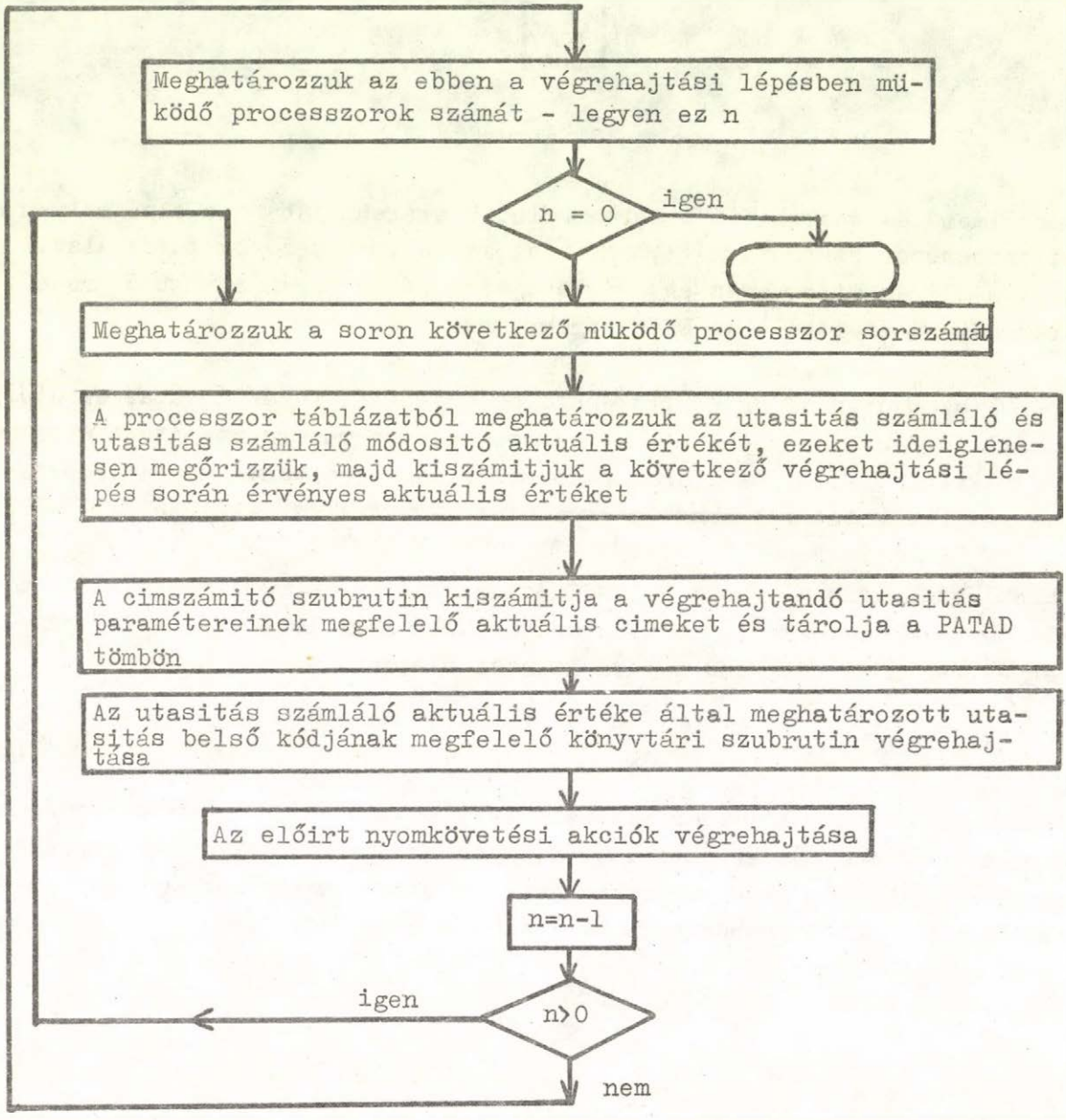
Az értelmező és végrehajtó program belső hierarchiájában az első helyen a multiprocesszor szubrutin áll, amely az egyes processzorok futás alatti vezérlését irányitja, azaz egy végrehajtási lépés során a müködő processzorok sorszámát átadja az EXEC1 programnak.

Az EXEC1 szubrutin az egyes processzorok müködését vezérli, azaz az utasitásszámláló tartalma által meghatározott belső sorszámu utasitást értelmezi és végrehajtja, meghatározza a következő végrehajtandó utasitás belső sorszámát /valamint végrehajtja az előirt nyomkövetési akciókat/.

Az EXEC1 szubrutin elsőként a CIMSZ /cimszámitó/ szubrutint aktivizálja, amely meghatározza az utasitás paramétereinek megfelelő /aktuális/ cimeket az egyes paraméterek belső nyelvi leirása alapján.

Az EXEC1 szubrutin ciklusban hivja a CIMSZ szubrutint a végrehajtandó utasitás paraméter számának megfelelően, majd az utasitás kódjának megfelelő könyvtári szubrutin kerül végrehajtásra. A paramétereknek megfelelő /aktuális/ cimek a PATAD tömbön kerülnek tárolásra és átadásra.

Az értelmező és végrehajtó rendszer funkcionális blokkvázlata:

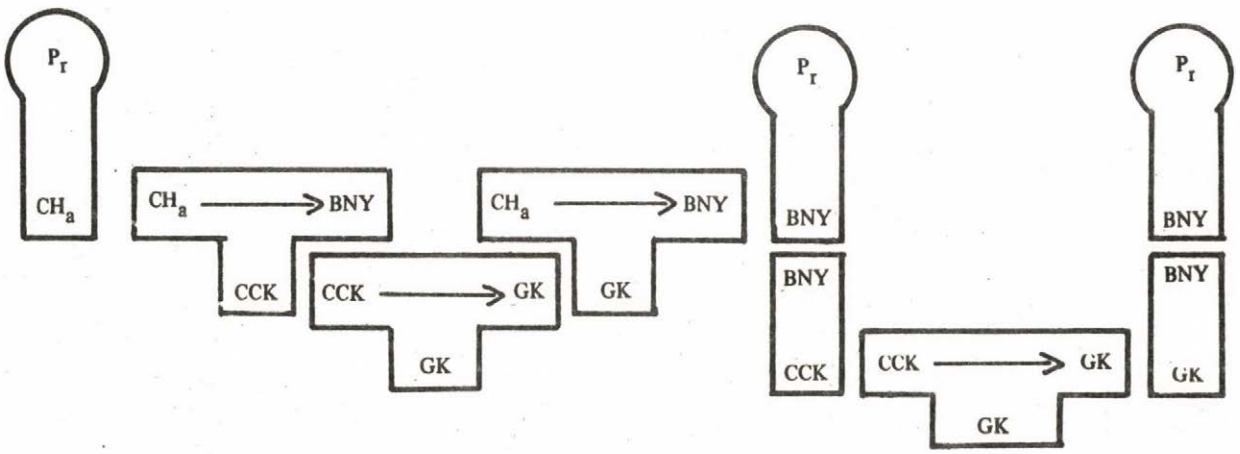


14. ábra

Az assembly formátumu CHANGE utasitásoknak megfelelő alapkönyvtárban a nyelv adattipusainak és utasitásainak szintaktikus és szemantikus leirása található.

A könyvtárban /L18/ az utasitások sorszámozottak. Ez a sorszám egyben az utasitás belső kódja is. /Lásd: Függelék/

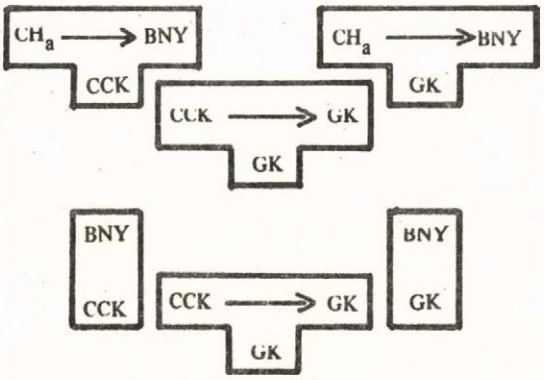
Végül a fordító-értelmező rendszer teljes T diagramja.



15. ábra

Ebből a teljes diagramból az alábbi két /a rendszer müködtetéséhez csak egyszer elvégzendő/ forditási lépés végrehajtásával kaphatjuk az 10. ábrán feltüntetett

T diagramot /ennek komponensei minden CHANGE program forditásakor és futásakor aktivizálásra kerülnek/.



16. ábra

FÜGGELÉK

Néhány példa arra, hogy a CHANGE utasitásoknak milyen mnemonikus assembly formátumu CHANGE utasitás kódokat feleltettünk meg:

BELSŐ SORSZÁM	CHANGE UTASITÁS	ASSEMBLY FORMÁTUMU UTASITÁS KÓD
Aritmetikai müveleti utasitások		
	P1 = P2 + P3	PLUS P1;P2;P3;
	P1 = P2 - P3	MINUS P1;P2;P3;
	P1 = P2 x P3	SZOR P1;P2;P3;
	P1 = P2 / P3	PER P1;P2;P3;
Logikai müveleti utasitások		
	L1 = L2 NOT L3	NOT L1;L2;L3;
	L1 = L2 AND L3	AND L1;L2;L3;
	L1 = L2 CR L3	CR L1;L2;L3;
	L1 = L2 NAND L3	NAND L1;L2;L3;
	L1 = L2 NOR L3	NOR L1;L2;L3;
Belső függvények		
37	V1=ABS V2	ABS V1;V2;
38	V1=ALOG V2	ALOG V1;V2;
39	V1 ALOG10=V2	ALOGT V1;V2;
41	V1=MAX V2,V2	MAX V1;V2;V3;
40	V1=MIN V2,V2	MIN V1;V2;V3;
42	V1=ATAN V2	ATAN V1;V2;
43	V1=COS V2	COS V1;V2;
44	V1=SIN V2	SIN V1;V2;
Vezérlésátadó utasitások		
22	IF/I/ I1,I2,I3	IFI I1;I2;I3;I4;
24	IF /I1.real.I2/ I1,I2	IFIEQ I1;I2;I3;I4;
25		IFINE I1;I2;I3;I4;
26		IFIGT I1;I2;I3;I4;
27		IFIGE I1;I2;I3;I4;
Ciklus utasitások		
69	DO P1 I1=I2,I3,I4	DO P1;I1;I2;I3;I4;
70	/P1/ REPEAT	REPET P1;

BELSŐ SORSZÁM	CHANGE UTASITÁS	ASSEMBLY FORMÁTUMU UTASITÁS KÓ
Szubrutin hivás		
76	SUBR P1	SUBR P1;
77	EXIT	EXIT
75	.SDI./I1/	SDI I1;
CHANGE utasitások		
78	CHANGE I1=I2,I3	CHANG I1;I2;I3;
79	CHANGE V1=V2	CHANG V1;V2;
1	NULL	NULL
A végrehajtási módot előiró utasitások		
91	UTSZ=I1	UTSZI I1;
95	COPROCESSZOR I1;I2	CPROC I1;I2;
96	WAITE	WAIT
88	DCONTROL I1;I2	DCON I1;I2
Stop utasitások		
56	STOP	STOP
58	GENERAL STOP	GSTOP
Adatátviteli /INPUT-OUTPUT/ utasitások		
60	FILE F IS ARRAY T	FILEA F;T;
61	FILE F IS DEVICE I	FILED F;I;
62	FILE F IS ON SECONDARY STORAGE	FILES F;
63	LOCATE F TO I	LOCAT F;I;
64	FROM F1 TO F2	FROMF F1;F2;

IRODALOMJEGYZÉK

/1/ Jay Earley -
Howard Sturgis: A formalism for translator interactions CACM 1970. 10

/2/ Legendi Tamás: A CHANGE nyelv /multiprocesszor MTA SZTAKI Tanulmányok 1973.

FORTRAN-TRACER FORRÁSNYELVI NYOMKÖVETŐ PROGRAM

Nagy Zsigmond - Nagy Judit

A TRACER forrásnyelvi nyomkövető program a CDC 3300-as gépre készült, COMPASS nyelven. Feleslegessé teszi a FORTRAN nyelven irt programok belövése és kipróbálása során a próbaprintek alkalmazását, megkönnyiti a kiirt eredmények azonositását, lehetővé teszi a hibák megkeresését. A TRACER nyomkövető a program futása folyamán, a program előre megadott helyein a sornyomtatóra kinyomtat minden értékadó utasitás baloldalán szereplő értéket. Ez lehet valós, komplex, dupla pontosságu, egész vagy logikai tipusu. Az adott érték mellé kinyomtatásra kerül a szubrutin nevének első négy karaktere és a forrásnyelvi listán szereplő sorszám, ahol az értékadó utasitás előfordul. Amennyiben a nyomkövetni kivánt szakaszban ciklusutasitás fordul elő, kinyomtatásra kerül a ciklusba lépést jelző "KEZDI" szó, és minden ciklusban a ciklusváltozó aktuális értéke.

A FORTRAN-TRACER HASZNÁLATA

A FORTRAN-TRACER az 5-ös disken található. A file nyitókártyája:

```
$ * DEF (O,W, DSI, 222041, FORTRAN-TRACER, UT, *X*, O)
```

A forrásprogram leforditását az FTNU task helyett az FTTR végzi.

A task name kártya:

```
$FTTR,DSI (paraméterek), ahol DSI az FTTR-t tartalmazó megnyitott
                         file dsi-je.
```

Paraméterek lehetnek:

FTNU paraméterek: I,X,L,F,S,A, *

Nyomkövető paraméterek:

T A program minden értékadó utasitást követ.

C Ciklusokban nem történik nyomkövetés, csak ha erre külön utasitás van (ld. utasitások). Cikluson kivül a program mindent nyomkövet.

CMI (I egyjegyü egész szám)
I mélységig egymásbaskatulyázott cikluson kivül előforduló értékadó utasitásokat követ a program. Igy I-nél többszörösen egymásbaskatulyázott ciklusokban csak akkor történik nyomkövetés, ha erre külön utasitás van.

Az FTTR kártya nyomkövetést biztositó paraméterei a forrásprogramban módosithatók. Ha az FTTR kártyán a T,C,CMI paraméterek közül egyik sem szerepel, a nyomkövetés a forrásprogramban megadott utasitásokkal történik.

A programban megadhatók nyomkövető és nyomkövetést módositó utasitások, amelyek a forrásnyelvi programban a szegmensnyitó (programnév, FUNCTION vagy SUBROUTINE) utasitást követik.

Egy szegmensnyitó utasitást több TRACE utasitás is követhet. Az utasitások első csoportja (nyomkövető utasitások) csak az adott szegmensre érvényes.

TRACE LINE N1,N2-N3,N4

ahol N1,N2, stb. a program listáján megadott sorszámok. (A TRACE utasitások nem növelik a sorszámot!) Az adott sorokban, illetve az adott sorok közötti programszakaszokban történik nyomkövetés.

TRACE LABEL L1,L2-L3

ahol L1,L2 stb. az adott programszegmensben szereplő cimkék. A cimkék által meghatározott sorokban, illetve sorok közötti programszakaszban történik nyomkövetés.

TRACE NAME X1,...,XN

ahol X1,...,XN változónevek, melyek az adott szegmensben szerepelnek. Az adott változók értékadásait a program nyomköveti.

TRACE NO

az adott szegmensben nem történik nyomkövetés.

A TRACE utasitások másik csoportja a nyomkövetés módositására vonatkozik, és hatása kiterjed azokra a szegmensekre is, amelyek a forrásnyelvi programban helyileg az utasitás után következnek.

TRACE CANCEL $p_1 \cdots p_\ell$

ahol p_i az L,T,C,CMI paraméterek lehetnek. A megadott paramétereket az adott szegmenstől kezdve az FTTR nem veszi figyelembe.

TRACE SELECT $p_1 \cdots p_\ell$

ahol p_i szintén az L,T,C,CMI paraméterek lehetnek. A megadott paramétereket az FTTR az adott szegmenstől kezdve figyelembe veszi, akkor is, ha eredetileg nem szerepeltek az FTTR paraméterei között.

(A TRACE SELECT és TRACE CANCEL kártyákon a paraméterek közé nem szabad vesszőt tenni.)
A TRACE utasitások a kártya 7. oszlopától kezdődhetnek, folytatósor nélkül. Ha a nyomkövetni kivánt cimkék vagy sorok egy kártyára nem férnek el, ujabb TRACE kártyán folytathatók. A T,C,CMI nyomkövető paraméterek közül egyidejüleg csak egynek van értelme. Együttes előfordulás esetén a program a paramétereket T,CMI,C sorrendbe állitva a magasabb prioritásut veszi figyelembe.

A TRACER a forrásnyelvi programról L paraméter előfordulása esetén listát készit, amelyet kétfajta sorszámmal lát el. Az első, a sor elején lévő sorszám alapján történik az eredmények azonositása, a FORTRAN fordító pedig a második, a sor végén elhelyezkedő sorszám szerint jelzi ki a szintaktikus hibákat.

A következő példa a FORTRAN-TRACER használatát illusztrálja:

```
$ JOB,...
$ SCHED, CORE=46, CLASS=B, SCR=5, 854=1
$ * DEF (0,,TR, 222041, FORTRAN-TRACER, UT, *X*, 0 )
$ FTTR, TR(X,L,T)
```

```
          PROGRAM MAIN
          .
          .
          .
          .
          END

          SUBROUTINE ELSOE
          TRACE CANCEL T
          TRACE LINE 5-10, 15, 20-22
          TRACE LABEL 3-4
          .
          .
          .
          .
          END

          SUBROUTINE MASODIK
          .
          .
          .
          .
          END

          SUBROUTINE HARMADIK
          TRACE SELECT CM2
          TRACE NAME X1,X2
          .
          .
          .
          END
               FINIS
ØX,LGO
```

A példában szereplő program fő szegmensében minden értékadás kinyomtatásra kerül. Az ELSOE szubrutinban csak a megadott sorokban, illetve szakaszokon, a MASODIK szubrutinban pedig egyáltalán nem történik nyomkövetés.
A HARMADIK szubrutinban az értékadó utasitások közül, amelyek kettőnél mélyebben egymásbaskatulyázott ciklusban szerepelnek a TRACER csak azokat követi, amelyekben az X1 vagy X2 változók kapnak értéket. A ciklusokon kivül, és kétszeres mélységig egymásbaskatulyázott ciklusokban előforduló összes értékadó utasitást követi a program.

A FORTRAN-TRACER számára (a forditással együtt) 46 CORE szükséges, mely a forditás után a program számára felszabadul. A // SCHED kártyán fel kell tüntetni a disc igényt is. A kinyomtatásra kerülő változóktól és a program hosszától függően a program futási ideje 1,2,-2,5-szeresére nő.

PROGRAMOZÁSI NYELVEK SZINTAXISA ÉS SZEMANTIKÁJA, AZOK DEFINICIÓJA ÉS FORMALIZÁLÁSA

Farkas Ernő

Mint az közismert, a programozási nyelvek esetében szokás a nyelv szintaxisáról és szemantikájáról beszélni. Intuitive azt mondjuk, hogy egy program szintaktikusan hibás, ha a leírásában olyan formai hibát vétettünk, amitől a nyelv fordítóprogramja nem fogadja el /hibajelzést ad/; és azt mondjuk szemantikusan hibás, ha mást hajt végre, mint amit várunk tőle. Mivel a program igy vagy ugy, de mindkét esetben rossz, ez a körülmény alkalmas arra, hogy a szintaxis és a szemantika közötti alapvető különbségeket elfedje.

Kicsit pontosabban fogalmazva a fogalmakat igy definiálhatjuk:

Szintaxisnak nevezzük a nyelv azon szabályait, amelyek alapján az adott nyelven helyes programokat irhatunk, vagy egy programról eldönthetjük, hogy helyes-e.

Szemantikának nevezzük a nyelv azon szabályait, amelyek megmondják, hogy egy szintaktikusan helyes program milyen akciók végrehajtását jelenti.

A fentiekben egy nyelv szintaxisáról és szemantikájáról beszéltünk, de beszélhetünk egyetlen program szintaxisáról és szemantikájáról is, ez esetben arra gondolunk, hogy abban a konkrét esetben, hogyan érvényesülnek a fenti szabályok.

A definiciónak ezen a szintjén a szintaxis és szemantika között a következő különbségeket láthatjuk:

1. A szintaxisról lehet beszélnünk a szemantika ismerete nélkül, de fordítva lehetetlen.

2. A szintaktikus hibákról beszélhetünk önmagukban is, a szemantikus hibákról nem. Ez részletesebben a következőket jelenti. Ha ismerjük a szintaxis összes szabályát és egy konkrét programot, akkor megállapithatjuk, hogy a program szintaktikusan, helyes vagy nem. Ha a szemantika összes szabályát ismerjük, akkor ebből még nem tudjuk eldönteni, hogy a program szemantikusan helyes-e vagy nem, csak akkor, ha összehasonlitjuk azzal a feladattal, amit a programnak realizálni kellene. Ez viszont egy olyan külső fogalom, aminek a nyelvhez semmi köze.

3. A szintaxis statikus fogalom, a szemantika viszont dinamikus. Ezt ugy kell érteni, hogy egy programon önmagában ellenőrizni lehet a szintatikus szabályok teljesülését, a program szemantikáját azt, hogy milyen akciók, milyen sorrendben mennek végbe, általában nem határozhatjuk meg csak a programból, hanem bizonyos bemenő adatokra is szükség van és ez a sorrend más és más bemenő adatoknál más és más lehet. Ezt az utolsó különbséget ugy lehetne kiküszöbölni, hogy a bemenő adatokat is a program részének tekintjük, azaz a szemantikát egy programból és egy adatból álló párokon értelmeznénk, mi azonban a továbbiakban nem ezt a megoldást választjuk.

Mielőtt tovább mennénk, foglalkozzunk egy kicsit azzal a kérdéssel, hogy mi a szintaxis és szemantika definiálásával a célunk. Ez a cél többrétü:

1. Össze akarjuk foglalni a szabályokat, amelyek alapján a célul kitüzött feladatot az adott nyelven megfogalmazhatjuk.
2. Azt akarjuk, hogy az igy megfogalmazott feladatokat mindenki /emberek és számológépek/ megértse és ugyanugy értse.
3. Ezeket a szabályokat akarjuk felhasználni arra, hogy segitségével a nyelvet ujabb és ujabb gépekre implentáljuk.

Mint látjuk a probléma két irányban vetődik fel, egyrészt az emberek, másrészt a gépek irányában. Ezek közül egyelőre a gépek közötti kompatibilitás megteremtése látszik könnyebbnek, ugyanis az ember a felvetett problémákat az ő szokásos matematikai modelljébe képezi le, a gépek matematikai modellje viszont ettől különböző, "a gép más aritmetikával számol". Ez a különbség egyelőre csak a felhasználók egy részénél tudatosul és csak igen kevesen képesek arra, hogy a szokásos matematikai modellről a gépi modellre áttérjenek vagy közvetlenül arra fogalmazzanak.

Ennek az okai közé tartozik az is, hogy a gépi aritmetika fogalmai nincsenek teljesen tisztázva.

A másik kérdés amivel foglalkoznunk kell mielőtt tovább megyünk, egy terminológiai kérdés: mit nevezünk definiciónak és formalizálásnak. Mi a továbbiakban definiciónak nevezzük a következőt: feltételezzük, hogy vannak olyan egyszerü, mindenki számára magától értetődő és mindenki számára ugyanazt jelentő alapfogalmak, amelyek segitségével minden további fogalmat és állitást megmagyarázunk.

Formálisnak nevezünk egy definiciót akkor, ha a következő módon végeztük a definiálást:

1. Megadunk egy jelölésrendszert és megmondjuk, hogy a problémát az adott jelölésrendszerrel hogyan lehet leirni.
2. Az adott leiráson milyen formai változtatásokat kell vagy lehet elvégezni.
3. A kapott eredmény mit jelent.

Nézzük meg például, hogyan alakul a szintaxis formális definiciója a mondatszerkezetü grammatikák segitségével.

1. Megadjuk a nyelv karakterkészletét és bizonyos segédszimbólumokat, továbbá olyan szabályokat, amelyek alapján, ha egy jelsorozat tartalmaz bizonyos réssorozatot, akkor ezt egy másik jelsorozattal helyettesithetjük.
2. Képezzünk olyan jelsorozatokat, amelyek egy adott jelből levezethetők a fenti helyettesitésekkel, belőlük további jelsorozat már nem vezethető le, és nem tartalmaznak segédszimbólumot.
3. Az igy definiált jelsorozatok lesznek a szintaktikusan helyes programok.

Hasonlóan definiálható VDL-ben a nyelv szemantikája. Mindkét definició kitünően szemlélteti a formalizálás lényegét, de a későbbiekben láthatjuk, hogy a valóságot viszont nem adja vissza ilyen hiven.

A formális definiciónak az a jelentősége, hogy pontos és egyértelmü, ez azonban lényegében csak a 2. pontban foglaltakra vonatkozik, hiszen egy formális leiráshoz és az abból kapott eredményekhez többféle jelentést is kapcsolhatunk.

Ezek után következzék a nyelv szintaxisának és szemantikájának pontos definiciója. Ez a definició igen általános lesz és éppen ezért az egyes konkrét esetekre nagyon keveset mond.

Legyen A a felhasználható karakterek halmaza.

$X = F(A)$ Az A-ból képzett véges jelsorozatok halmaza.

Legyen I a lehetséges input adatok halmaza, O pedig a lehetséges output adatok halmaza.

Legyen Fio azon függvények halmaza, amelyek az inputadatokhoz az output adatokat rendelik.

L nyelvnek nevezzük X egy <u>rekurziv</u> részhalmazát.
Ekkor L szintaxisát X karakterisztikus függvénye adja meg.

$$g(X) = \begin{cases} \text{igaz} & \text{ha } x \in L \\ \text{hamis} & \text{ha } x \notin L \end{cases}$$

Minden $x \in X$-re.

L szemantikáját pedig egy $\varsigma \in [X \rightarrow \text{Fio}]$ leképezéssel adhatjuk meg.

$$\varsigma(X) = \text{fio} \qquad X \in L \quad \text{és} \quad \text{fio} \in \text{Fio}.$$

Ha a fenti definiciót elfogadjuk, rögtön láthatjuk mennyivel egyszerübb a szintaxis megadása a szemantika megadásánál. A szemantika megadásához ugyanis 3 dolgot kell ismernünk.

1. Azt, hogy milyen jelsorozat tekinthető egyáltalán programnak, azaz pontos szintaxist.
2. Azt, hogy milyen függvényeket kaphatunk értékül.
3. Végül, magát az összefüggést, ahogy a programhoz a függvényt hozzárendeljük.

Vizsgáljuk meg most azt, hogy mit mondhatunk az egyes pontokról külön-külön.

A szintaxis formalizálásával kapcsolatban sok szakembernek az a véleménye, hogy az egy elintézett ügy: "hiszen minden programozási nyelv bizonyos értelemben CF nyelv, tehát a metanyelv jól leirja". Mások viszont legalább ilyen joggal mondják, hogy egyetlen programozási nyelv sem lehet CF. A helyzet ugyanis a következő a forditó programok a szintaxis vizsgálatát két részre bontják, az egyik részben amit szintaxis analizisnek neveznek és amely a metanyelvvel jól leirható, azt vizsgálják, hogy a program milyen egységekre bomlik fel, ezek az egységek milyen további alegységekre és igy tovább, lényegében nem tesznek mást mint a program egyes részeit össze zárójelezik többszörös mélységben. Ezt a zárójelezést implicite beleértjük a programba amikor megirjuk. Ezután a szintézis fázisában a programot a végrehajtás sorrendjében rendezzük át. Azaz először a legmélyebb szinten összezárójelezett részt kell kiszámitanunk, ha ez rendelkezésre áll akkor a felette levő szinteket és igy tovább. Igenám, de nem elég megállapitani a programban szereplő elemek összetartozásának hierarhiáját, hanem meg kell állapitani, hogy a kifejezésben szereplő bizonyos alapvető elemekről, hogy a megfelelő helyen egy megfelelő elem áll-e, és a megfelelő formában áll-e. Általában elmondhatjuk, hogy a nyelvnek ezek az épitőkövei bizonyos tulajdonságokkal kell, hogy rendelkezzenek, vagy nem szabad rendelkezniük vagy rendelkezhetnek. A forditóprogramban a munka jelentős részét teszi ki annak a vizsgálata, hogy az adott kontextusban az adott elem milyen tulajdonságu, illetve hogy a kontextus hogyan változtatja meg a tulajdonságait. Például igen sok nyelvben lehet többdimenziós tömböket deklarálni a deklaráció hatására feljegyzésre kerül, hogy hány dimenziós a tömb; és a tömb minden felhasználásánál ellenőriznünk kell, hogy ugyanolyan dimenziószámmal használtuk-e fel. Ezt metanyelvvel képtelenség leirni, bár maga ez a leirás is utmutatást nyujt arra, hogy hogyan lehetne az ilyen nyelvek szintaxisát jobban formalizálni. Elképzelhető persze olyan nyelv is, amelynél az egész zárójelezési eljárásnak, azaz a metanyelvnek nincs sok értelme. Képzeljük el, hogy a számológéppel bizonyos utasitássorozatot akarunk végrehajtatni és ezt ugy irjuk le, hogy először leirjuk a müveleteket egymás után egy bizonyos szempontból, majd ugyanilyen sorrendben valamilyen más szempontból, és a forditás során ezt a két listát elemenként összepárositva dolgozzuk fel.

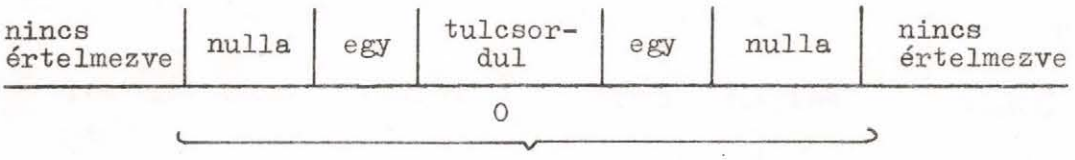
Egy másirányu bonyolult problémát vet föl az ALGOL 68 ahol az elemek tulajdonságai szabják meg a zárójelezés módját például olyan módon, hogy az operátorok prioritását az ujradeklarálás segitségével megváltoztathatjuk.

Ez az egész felsorolás azt bizonyitja, hogy nem várhatjuk azt, hogy egy egységes minden nyelvre egyformán jó formális szintaxist definiáló módszert találjunk, de azt várhatjuk, hogy minden egyes nyelvre vagy szerencsésebb esetben egész nyelvcsaládokra megtaláljuk a szintaxis formalizálásának módját.

Mivel a nyelvek bizonyos alapelemekből épülnek fel bizonyos összerakási szabályok segitségével a szintaxis definiálásának is ezt az utat kellene követni. Egy lehetséges ut lenne például a metanyelvi leirás továbbfejlesztése olyan módon, hogy a nyelvet nem a karakterekig vezetnénk vissza, hanem bizonyos alapelemekre és formalizálnák az, hogy ezeknek milyen tulajdonságaik vannak és azokat hogyan kell megvizsgálni.

A másik komoly feladat a szemantika definiálásának utján az, hogy megismerjük azokat a függvényeket, amelyeket egy-egy adott nyelven leirhatunk. A magasszintü nyelvek formalizmusa sok emberben kelti azt a téves hitet, hogy egy matematikailag jól leirt függvény számológépes kiszámitásához nem kell mást tennie csupán a megfelelő képleteket, vagy algoritmusokat a nyelv szabályainak megfelelő formában leirni. Elfelejtik ugyanis, hogy a matematika valós és egész számokkal dolgozik, a gép viszont fixpontos és lebegőpontos számokkal. Néhány apró probléma ebből a témakörből: a müveletek nem asszociativok a + (b + c) lehet, hogy tulcsordul és ugyanakkor (a + b) + c pedig nem, tehát a képletekben a zárójeleket más és más helyre helyezve a függvény értelmezési tartománya megváltozik. Ha egy szám létezik nem biztos, hogy a negativja is létezik, vagy a reciprokértéke is létezik, hiszen például az n jegyü bináris ábrázolás esetén 2^n darab számot tudunk ábrázolni és ha ezt mind fel is használjuk, akkor világos, hogy nem lehet ugyanannyi negativ mint pozitiv szám. Lebegőpontos számok esetén a szorzat akkor is nulla lehet, ha egyik tényezője sem nulla. A dolgot még sulyosbitja, hogy lebegőpontos számok esetében maga a lebegőpontos szám fogalma sem egyértelmü, hiszen például van kerekitéssel dolgozó aritmetika és letöréssel dolgozó aritmetika. Ez a különbség nemcsak azt okozhatja, hogy ugyanaz az algoritmus enyhén különböző eredményeket ad a két különböző üzemmódban, de szerencsétlen esetben ez a különbség szignifikáns is lehet sőt az is előfordul, hogy az egyik üzemmódban az algoritmus müködik a másikban pedig nem. Hasonló problémát vet fel az a kérdés, hogy mit csináljunk a legkisebb karakterisztikával még ábrázolható, de nem normalizálható számokkal.

Nagyon teoretikusan hangzanak, de gyakorlati szempontból fontosak a következő kérdések is: mit nevezzünk a lebegőpontos számok körében konvergenciának, mikor folytonos egy függvény, a legközönségesebb függvények polinom, törtfüggvények, iterativ uton előállitott függvények folytonosak-e. Könnyen látható például, hogy $f(x) = \frac{1}{x^2} \cdot x.x$ függvény a valós számok körében lényegében azonosan egy. Ugyanez a függvény a lebegőpontos számok körében valahogy ily néz ki:



az ábrázolható lebegőpontos számok tartománya

Azt mondhatjuk a függvények megadásáról, hogy tulajdonképpen két módszert képzelhetünk el. Az elsőt "matematikai megadás"-nak nevezhetjük, ilyenkor valamilyen matematikai módszer segitségével egész pontosan definiáljuk azt, hogy milyen objektumok a nyelv alapelemei, azokkal milyen tevékenységeket végezhetünk el, de nem foglalkozunk azzal a kérdéssel, hogy lehet-e az igy definiált függvényeket számológépen megvalósitani, hogyan lehet és hány féleképpen lehet. Ennek a módszernek az a lényege, hogy feltesszük azt, hogy az igy adott definició olyan pontos és olyan egyértelmü, hogy bármilyen realizációt választunk is ki az alap épitőelemekre, ennek a függvényekre semmilyen mellékhatása nem lesz. A másik módszert "gépi megadásnak" nevezhetjük, ilyenkor olyan adattipusokból és utasitásokból indulunk ki, amelyek általában minden gépen léteznek, és ezekből épitjük fel a függvényeinket. Bár ez a megoldás a gyakorlat szempontjából kézenfekvőbb az előzőnél, de egyértelmü függvény definiciókat csak ugy kaphatunk, ha az alapfogalmaknak pontos matematikai definicióját rögzitjük le.

Végül beszélnünk kell arról is, hogyan lehet a szemantikát definiáló ς leképzést megadni. A három közül eddig ezen a területen tették meg messze a legtöbb lépést. A ς leképzés megadása két részből áll, egyrészt meg kell adnunk a nyelv alaputasitásainak megfelelő függvényeket, másrészt meg kell adnunk, hogy az utasitások összekapcsolása a függvények milyen összekapcsolását jelenti.

Egy másik szempontból a szemantikát vagy interpreterrel vagy transzlátorral lehet megadni.

Interpreternek nevezünk egy olyan $\bar{\varsigma}$ függvényt, amely $I \times X$-en van értelmezve és értékkészlete O. Ahol(I a lehetséges input adatok halmaza O a lehetséges output adatok halmaza L pedig a nyelv. A $\bar{\varsigma}(i,x)$ függvény pedig($i \in I$, $x \in L$) pedig egy olyan leképzés, amely egy adott x programhoz és a hozzátartozó i adathoz a program végrehajtása során keletkező outputot rendeli.

A transzlátor definiciója lényegében azonos a ς leképzés definiciójával, azaz:

$$\varsigma(X) = \text{fio} \qquad X \in L, \qquad \text{fio} \in \text{Fio}.$$

Interpreter esetében egy konkrét X program szemantikáját kifejező fio függvényt ugy kapjuk meg, hogy a konkrét x-et behelyettesitjük a $\bar{\varsigma}$ interpreterbe a i inputot pedig végig futtatjuk az s halmazon.

Mind az interpreternek, mind a transzlátornak vannak előnyei a másikkal szemben. Az interpreter esetében nem kell definiálni az Fio függvényosztályt, a transzlátor esetében pedig nem kell foglalkozni az input adatokkal.

Intuitive ugy érezzük, hogy az interpreter jól definiálja a nyelv szemantikáját, de sokkal rosszabbul, kevésbé áttekinthetően az egyes konkrét programokét.

Mindkét esetben a szemantika megadása a ς , illetve $\bar{\varsigma}$ megadásával történik. Hogy adhatjuk meg ezt a függvényt? Ezt a függvényt valamilyen nyelven irjuk le. Ez a nyelv lehet valamilyen természetes nyelv pl. angol, valamilyen matematikai formalizmus, vagy valamilyen programozási nyelv, vagy ezek keveréke. A nyelv szemantikájának a formalizálásának esetében a definiáló nyelv mindig egy matematikai formalizmus vagy egy olyan programozási nyelv, amely a szemantikája nyilvánvalóbb mint a definiálandó nyelvé. A szemantika definiálásának esetében is érvényesül a kettősség itt is beszélünk "matematikai" - és "gépi definicióról".

Transzlátorok esetében az Fio függvényosztályt is le kell irnunk ennek a leirásnak a nyelve lehet azonos is a ς-t leiró nyelvvel de lehet különböző is.

A következőkben megpróbálom csoportositani a szemantika definiálására irányuló módszereket:

Matematikai definiciók:

1. Bitenként leiró módszerek

Ezek a módszerek abból indulnak ki mivel az egyes számológépek mindig más gépre nem jellemző utasitáskészlettel memóriakapacitással, cimzési módokkal stb. rendelkeznek először meg kell alkotni egy általános számológép modellt valamilyen turinggép, absztrakt automata vagy formális rendszer formájában és ezek után egy nyelv szemantikáját ugy adhatjuk meg, hogy a nyelv programjaihoz az absztrakt számológép programjait rendeljük hozzá. Mivel a leirás bitsorozatok vagy jelsorozatok transzformációjából áll az igy adott leirás tökéletesen egyértelmü és rendszerint alkalmas arra, hogy matematikai tételeket bizonyitsunk ilyen módon. Gyakorlati felhasználásukra nem igen van példa, mert a használt formalizmus tömeg emberi és gépi fogyasztásra egyaránt alkalmatlan. Megjegyzendő még, hogy igy definiált nyelvek, vagy nyelvi jelenségek inkább a matematikai példa illusztrálására szolgálnak, mint a gyakorlati programozás céljára.

2. Állapottranszformációs módszer

Az állapottranszformációs módszer a nyelvet utasitásokból épiti fel, az utasitás szemantikus tartalma egy absztrakt állapothalmazon végrehajtott transzformáció. Az utasitásokból bizonyos kifejezéseket képezhetünk, azokból meg ujabbakat, mig végül egy programot kapunk. Ez a módszer megmutatja, hogy hogyan épithetjük fel egy program állapottranszformációját az egyes utasitások állapottranszformációjából anélkül, hogy magukról az állapotokról vagy a transzformációkról valami konkrétet tudnánk. Ez a módszer inkább elméleti eredmények létrehozására alkalmas, ezek az eredmények azonban közeli kapcsolatban állnak a gyakorlattal.

Ezt a módszert alkalmazta Dana Scott és Christopher Strachey (2). Ez az irás azért is érdekes, mert bátran szakitottak sok olyan hagyománnyal, amely inkább akadályozta a munkát mint segitette.

A következő módszer már félig a gépi módszerek közé tartozik.

3. A Lambdakalkulus és az applikativ nyelvek

A lambdakalkulus egy olyan matematikai formalizmus, amely annak leirására szolgál, hogy hogyan képezhetünk kifejezéseket, a kifejezésekből függvényeket, hogyan helyettesithetünk ilyeneket egymásba, hogyan definiálhatjuk a kifejezésekben szereplő változók hatáskörét. Mint azt Church bebizonyitotta a lambdakalkulus algoritmikusan egyenértékü a rekurziv függvényekkel.

A fentiek alapján a lambdakalkulus rendkivül alkalmasnak látszik a szemantika definiálására. Az alkalmazás során azonban problémák merültek fel, melyek megértéséhez ujabb fogalmak bevezetésére van szükség.

Egy nyelv applikativ tulajdonságainak nevezzük azokat a lehetőségeket, hogy kifejezéseket képezhetünk, függvényeket deklarálhatunk, ezeket alkalmazhatjuk kifejezésekben és hogy kifejezéseket kiértékelhetünk.

A nyelv imperativ tulajdonságainak nevezzük az utasitások sorrendjét, az ugrásokat, értékadásokat és függvények side-effectjeit.

A gépi kód teljesen imperativ jellegü nyelv. A magasabb szintü nyelvek között vannak teljesen imperativak (IPL/v) és teljesen applikativak (LISP 1.5) általában azonban a kétféle jellemző egyszerre található meg.

Az imperativ jellemzők általában a programozási nyelv hatékonyságát, az applikativ tulajdonságok a kényelmes programozást segitik elő. A csak applikativ vonásokat tartalmazó nyelveket applikativ nyelveknek nevezzük.

Az applikativ nyelveket a lambdakalkulus"szintaktikusan cukrozott" változatának nevezik, ezt ugy kell érteni, hogy egy applikativ nyelv lényében azonos a lambdakalkulussal csak a leggyakoribb kifejezésekre egyszerüsitő jelöléseket vezetnek be.

Más a helyzet az imperativ jellemzőkkel. Egyrészt problematikus az utasitások sorrendjének, ugrásoknak a leirása, ezt azonban meg lehet oldani valahogy (11, 3, 4, 5, 8) applikativ nyelven is.

A nagyobb probléma, hogy a legtöbb imperativ nyelvben lehetőség van arra, hogy egy mennyiségre több különböző néven is hivatkozzunk. Ezért, ha egy applikativ nyelven irt interpreterrel akarjuk a szemantikát definiálni, akkor ebbe a modellbe egy memóriamodellt is bele kell épiteni, amely képes ezt a lehetőséget biztositani.

Ugyancsak az imperativ tulajdonságok vetik fel azt a problémát, hogy a nyelv tisztán applikativ részei is imperativ vonásokat vesznek fel. Pl. egy függvény értéke függ az argumentumainak kiszámitási sorrendjétől.

Ehhez hasonlit az applikativ nyelveknek a problémája, amely viszont már nincs összefüggésben az imperativ vonásokkal, hogy a függvények paramétereit érték szerint vagy név szerint hivjuk-e.

Az applikativ függvények körében beszélhetünk név szerint és érték szerinti paraméterhivásról és ezeket a következőképpen értjük. Adott egy függvény, amelyet egy bizonyos helyen ki akarunk számolni, a függvényt egy kifejezés adja meg, amelyben a függvény paraméterei változóként szerepelnek. Pl.

$$f(x,y,z) = a^2x + by + z$$

Ha most ezt a függvényt egy olyan helyen akarjuk kiszámitani amit aritmetikai kifejezésekkel adunk meg /amely lehet egyetlen változó is/, pl. f(u + v, v, uv) , akkor érték szerinti hivás esetén először kiszámitjuk a kifejezések értékeit, majd ezek segitségével a függvénynek megfelelő kifejezést olyan módon, hogy a megfelelő változó helyén a megfelelő értékkel számolunk. Név szerinti hivás esetén a kiértékelést a függvénynek megfelelő kifejezés kiértékelésével kezdjük és valahányszor egy paraméterre szükségünk van a kiértékelést felfüggesztjük, kiértékeljük a megfelelő kifejezést és az igy kapott értékkel folytatjuk a függvény kiértékelését. Mint azt Manna és McCharthy (13) bebizonyitotta, az érték és név szerinti hivás azonos eredményt ad ugyan, de a név szerinti hiváskor függvénynek nagyobb az értelmezési tartománya általában, ugyanis ebben az esetben lehetséges, hogy valamelyik paraméter az adott helyen nem értelmezhető és igy az összetett függvény sem érték szerinti hivás esetén, név szerinti hivás esetén viszont lehetséges hogy azt a paramétert nem használjuk fel a függvény kiértékeléséhez.

A másik probléma, amely ilyenkor felmerül szintén nem kapcsolódik szorosan az imperativ tulajdonságokhoz, de éppen az imperativ tulajdonságok applikativ leirásával kapcsolatban merül fel gyakran ez a probléma az önalkalmazás problémája.

Magasabbrendünek nevezünk egy programozási nyelvet, ha függvényeken értelmezett vagy függvény értékü függvények is leirhatók benne. A probléma ott lép fel, ha a függvényeken definiált függvény önmagával azonos tipusu függvényekre is alkalmazható. Ez esetben ugyanis a függvények értelmezési tartományára a

$$V = V \longrightarrow V$$

rekurziv egyenlettel irható, ami ha $\longrightarrow$ jelölést ugy értelmezzük, mint egy halmazból egy másikba történő összes lehetséges leképzést számossági okokból lehetetlen eredményt kapunk. Nyilvánvaló tehát, hogy itt nem az összes lehetséges leképzésről van szó. Az ilyen függvényeknek az elméletét Dana Scott (12) alapozta meg algoritmuselméleti, hálóelméleti és topológiai alapokon. Ezen alapuló ujabb eredményeket találunk Manna és Vuillemin 14 munkájában.

Összefoglalva megállapithatjuk, hogy igen sok kisérlet történt a szemantika applikativ nyelvü interpreterrel történő definiálására. Ezeknek nagyon jó összefoglalása található Reynolds (5) müvében. Mindezek elméleti szempontból sok érdekeset tartalmaznak. Gyakorlati szempontból természetesen a VDL kiemelkedik a többi közül. (3,4).

Mindezeknek a kisérleteknek közös jellemzője azonban az, hogy nem a valóságos nyelvet interpretálják, hanem egy ugynevezett absztrakt szintaxist. Az absztrakt szintaxis lényegében a programnak egy olyan leirása, amely a nyelvben szereplő összes implicit információt explicite leirja. Az absztrakt szintaxis például minden változónál megmutatja annak összes lehetséges tulajdonságát vagy annak a hiányát megmutatja a kifejezések részeinek egybekapcsolódását stb. A konkrét program és az absztrakt szintaxis kapcsolatát egy transzlátor hozza létre, de ezt egyik irás sem részletezi.

4. Az öndefiniáló módszer

A módszert először a LISP és más listakezelő nyelvek alkalmazták, a módszer lényege a következő.

Feltesszük, hogy egész pontosan és egyértelmüen tudjuk definiálni a nyelv alapelemeit és bizonyos képzési szabályokat, ahogy bonyolultabb kifejezéseket felirhatunk ezekből. Ezek után a bonyolultabb elemeket az alapelemekből képzett kifejezésekkel irunk le, és a mégbonyolultabbak az eddig definiáltak segitségével és igy tovább.

A módszernek hármas előnye is van:

1. A fokozatos felépités lehetővé teszi a nyelv jó megértését.
2. Lehetővé teszi a nyelv interpreterének a létrehozását bootstropping utján.
3. Mivel az igy készült interpreter általában terjedelmes és lassu, ezt ugy javithatjuk fokozatosan, hogy a bennük szereplő bonyolult függvényeket egyenként kicseréljük közvetlenül gépikódban irt változatukra.

5. A portabilitáson alapuló módszerek

Nagyon gyakori jelenség az, hogy egy nyelvnek a különböző gépeken alkalmazott változatai jelentős szintaktikai és szemantikai eltéréseket mutatnak. Egy nyelv két forditóprogramját kompatibilisnek nevezünk, ha a két forditóprogram ugyanazokat a programokat fogadja el helyesnek és a leforditott programok végrehajtása ugyanazt az eredményt szolgáltatja. A kompatibilitás nyilvánvalóan csak a nyelv szintaxisának és szemantikájának pontos definiciója alapján valósitható meg.

A nyelv egy lehetséges definiciója és egyben a kompatibilitás elérésének legbiztosabb utja az, ha irunk egy forditóprogramot a nyelvre és ezt alkalmazzuk minden lehetséges gépre.

Egy program portabilitása az a tulajdonsága, hogy könnyen alkalmazható minden gépen, illetve sok gépen. A portabilitást ugy érhetjük el legkönnyebben, ha egy portabilis /azaz sok gépre könnyen alkalmazható/ nyelven irjuk meg a programot.

Ha a nyelv forditóprogramját egy portabilis nyelven irjuk meg és ez a forditóprogram egy portabilis tárgy nyelvre fordit /a két nyelv lehet azonos, lehet különböző is/; akkor mind a forditóprogramot, mind a leforditott programokat nagyon könnyü átültetni egy-egy konkrét gépre /rendszerint egyetlen makróprocesszálási fázis/.

A portabilis forditóprogram nemcsak a gép szempontjából jelenti a szintaxis és szemantika definicióját. Az igy nyert forditóprogram ugyanis általában lassu és terjedelmes, ilyenkor a forditóprogram kritikus részeit ujraprogramozzák a gép nyelvén a program alapján.

Vizsgálatok folytak arra nézve, hogy milyen nyelv alkalmas leginkább portabilitásra. (15) A vizsgálat azt mutatta, hogy az assembler szintü nyelv a legalkalmasabb. A vizsgálat ugyanis azt mutatta, hogy a gépi kódok általában hasonlók egymáshoz, ha pedig nagyon eltérnek (ILLIAC IV, CDC STAR) akkor ez még a magasszintü nyelveken is érezhető.

A portabilis nyelveknek ez a szintje teszi lehetővé, hogy a portabilis nyelvek szemantikája majdnem triviális.

Mint a felsorolás is mutatja a szemantika definiálásának igen sok lehetséges utja van, és bár pillanatnyilag még egyik ut sem vezetett célhoz, valószinü, hogy a célhoz több uton is el fogunk jutni.

Összefoglalva tehát a következőket mondhatjuk a szintaxis és szemantika definiálásáról és formalizálásáról.

Mindaddig nem beszélhetünk komolyan a szemantika definiálásáról, amig a szintaxisnak egy a jelenleginél sokkal pontosabb definiciója meg nem születik.

A számolásra orientált nyelvek szemantikáját nem definiálhatjuk közérthetően, addig amig nem szerzünk tiszta képet a számológép aritmetikájának az alapvető tulajdonságairól.

A szintaxis és szemantika formalizálása több különböző uton lehetséges; egyes nyelveknek bizonyára az egyik, más nyelvnek a másik felel meg jobban. Elképzelhető, hogy a jelenlegi nyelvekkel kapcsolatban belső ellentmondások me-

rülnek fel és ezért uj nyelvek definiálása válik szükségessé.

IRODALOMJEGYZÉK

[1] C.Strachey: Varieties of programing Language. International Computin Symp. Venice 1972.

[2] Dana Scott and Christopher Statchey: Towards a Mathematical Semantics for Computer Languages. Oxford University Programing Research Group 1971.

[3] E. .Neuhold: The Formal Description of Programing Languages. IBM System Journal 1971. No2.

[4] P.Lucas, K.Walk: On the Formal Description of PL/I. - Annual Review in Automatic Programing. 1969. Part 3.

[5] John C.Reynolds: Definitional Interpreters for Higher-Order Programing Languages. ACM Annual Conference 1972. Boston.

[6] J.McCarthy, Recersive Functions of Symbolic Expressions and Their Computation by Machine, Part I. Com ACM 3 196o.

[7] W.M.Waite: Building a mobil programmig system. The Computer Journal 1970.

[8] An Axiomatic Basis for Computer Programing C.A.R. Hoare Com. ACM 1969/10.

[9] P.J.Landin: A Correspondence Between Algol 60 and Churcl's Lambda Notation. Part I, Part II. COM ACM 1965/2,3.

[10] D.Scott: Outline of a Mathematical theory of Computation. Oxford University Programing Reseach Groups 1970.

[11] Z.Manna, J.McCarthy: Properties of Programs and Partial Function logic Machine Intelligence vol 5.

[12] Z.Manna, J.Vuillemin: Fixpoint Approach to the Theory of Computation. Com. ACM. 1972/7.

[13] P.J.Brown: Levels of Language for Portable Software. Com ACM 1972/12.

EGY "TERMÉSZETES SZÁMITÁSI RENDSZER" SZIMULÁCIÓS MODELLEZÉSE ÉS SZÁMITÁSTUDOMÁNYI VONATKOZÁSAI

Gáspár András

Vajon képes lesz-e valamikoris egy gép gondolkodni? Létrehozható-e gondolkodásra képes, mesterséges hardware?

Véleményem szerint igen. A bizonyossághoz természetesen pontosabban kell kérdezni és alapos vizsgálat szükséges.

Elképzelhető-e "intelligensebb" hardware bizonyos problémák számára, mint amilyent jelenleg használunk?

Elképzelhető-e egy másfajta, de egyszerü számitási rendszer, melyben több millió müvelet ténylegesen egyszerre történik?

Az általános vélemény az, hogy "igen", de természetesen a meggyőződés kevés. Az ugynevezett sejtautomata viszont, melyet az agy az idegsejt néhány tulajdonságából absztraháltak, az utóbbira már bizonyosságot is ad.

A természettől sok ötletet kaphat a számitástudomány. Ezért szükséges vizsgálnunk az agyat a számitástudomány nézőpontjából, mint egy "természetes számitási rendszert", mint egy "számitási rendszer architekturáját".

A jelen számitógépe nem a lehetséges agymodellek legjobbika

Foglaljuk össze azon modellek /a következőkben M/ valószinü alapelveit, melyek véleményünk szerint már elég jól tükrözhetik a valóságos agy alaptulajdonságait:

1. M determinisztikus automata.
2. Irányitott gráfra épül. /Ez a neuronhálózatnak felel meg. A pontok a neuronok sejttestjeit, az élek a szinapszisokat jelképezik/.
3. A gráf minden pontjához hozzátartozik lehetséges állapotainak egy halmaza. /Például a később tárgyalandó modellben két állapot lehetséges minden pont számára: "aktiv", illetve "passziv"/.

A továbbiakban egy-egy ponton belül szaggatott vonal választja majd el a "természetbeli" és az annak megfelelő "modellbeli" objektumot.

4. Az élő rendszerben /a következőkben E/ a szinapszisok állapota változik.

M-ben a gráf minden éléhez változó információk tartoznak, melyeket "nyomoknak" nevezünk.

5. E-t külső és belső környezete ingerli. E-ben az inger nyomán a receptorokból kiinduló ingerület keletkezik, melynek egy része befut E agyába. E-ben a modellezés első fázisában az agy inputjának a teljes ingerületet tekinthetjük, mig outputjának olyan parancssorozatot, mely E külső és belső szerveit, mirigyeit stb. vezérli.

M inputja az ingerület, outputja "parancssorozat operációkra". Az input és az output az M tipusától függően más és más folyamat. /Például: a későbbiekben elemzendő modellben speciális diszkrét folyamat/.

6. Az ingerület az agyban bizonyos neuronpályákon folyik tovább.

M-ben az input a gráf bizonyos utjain folyó "áramlást" hoz létre.

Az agyban az ingerület változtatja /sőt a modellezés első fázisában feltehetjük, hogy csakis az ingerület változtatja/ a pályája menti neuronok állapotait és a pályája menti nyomokat.

M-ben az "áramlás" a gráf bizonyos utjai mentén történő változások sorozata, melyben nyomok és pont állapotok változnak. Egy-egy ut mentén a változások szekvenciálisan történnek.
Szemlélhetjük ugy, hogy M-ben az áramlás okozza a változásokat.

7. Az agyban az ingerület jellemezhető az általa pillanatnyilag "gerjesztett" neuronokkal.

M-ben az áramlás az általa pillanatnyilag bizonyos állapotuvá változtatott pontok halmazával jellemezhető. Nevezzük az áramlást egy-egy pillanatban /a továbbiakban t-ben/ jellemző ponthalmazokat pozíciónak /a továbbiakban P_t/. Például a vizsgálandó modellben t-ben "aktiv" pontok halmaza lesz P_t.

8. A modellezés első fázisában feltehetjük, hogy az agyban egy pillanatban az ingerület továbbfolyásának pályáit a következők és csakis ezek határozzák meg:

   a/ a gerjesztett neuronokban kezdődő szinapszisok állapotai
   b/ az inger /és igy az ingerület/ folytatódása
   c/ E korábbi tevékenységei, operációi által keletkezett öningerlés /és igy az általa kiváltott ingerület/.

M-ben, t-ben az áramlás aktuális pályáit együttesen a következők határozzák meg:

a/ a P_t pontjaiban kezdődő élekhez tartozó nyomok. A továbbiakban N/P_t/.

b/ Az input folytatódása /a t-beli input/

c/ Az M korábbi operációi által kiváltott input. /Itt egy visszacsatolás van!/ Szemlélhetjük ugy, hogy a P_t irja elő, hogy mely nyomok vesznek részt az áramlás pályája aktuális folytatásának meghatározásában.

9. Az agyban, egy-egy pillanatban a gerjesztett neuronokban "kezdődő" szinapszisok állapotai determinálják a pillanatnyi outputot, azt, hogy az agy milyen pályán küld parancsokat, ami egyértelmüen meghatározza, hogy milyen parancsokat küld. A gerjesztett neuronok determinálják a belőlük "kiinduló", bizonyos állapotu szinapszisok olyan állapotváltozásait, melyeknek mi "várakozási" jelentést tulajdonithatunk. A várakozás azt jelenti, hogy az ingerület azon szinapszison folytatódása valószinü.

M-ben N/P_t/ határozza meg mi lesz a t-beli vezérlés:

a/ Mi lesz az output, M mit fog "operálni".

b/ Az input milyen folytatódása várható, M mit fog "várni".

10. Az előzőek szerint az agyban, a t-ben ujonnan gerjesztetté váló neuronokat az agy állapota /a régi gerjesztett neuronok és a szinapszisok állapota/, továbbá az inger folytatódása határozzák meg, és csakis ezek. Ezért mondhatjuk, hogy a "szituáció" határoz meg mindent. Az előzőek szerint a gerjesztett neuronok határozzák meg az agybeli történések helyeit.

M-ben mint láttuk, a t-beli uj pozíciót M állapota /P_t és a nyomok/ továbbá az input folytatódása határozzák meg. Nevezzük M állapotát és az input folytatódását együtt "szituációnak". Igy a szituáció meghatározza az uj pozíciót. Korábban láttuk, hogy az áramlás, a pontok állapotváltozásai, a nyomváltozások, a vezérléskialakulások mind a pozíció pontjaiból kiinduló élek által meghatározottak. Ilyen értelemben mondhatjuk, hogy a pozíció cimzi az előzőekben felsorolt müveletek helyeit M-ben. Beszélhetünk tehát pozíció általi vagy szituáció általi automatikus cimzésről.

o O o

Tehát mig a jelenlegi számitógépek "programozható és programozandó cimzésüek", addig az agy modelljei automatikus cimzésü számitási rendszerek. Ugy gondoljuk, hogy általában az ilyen rendszereket szokás "cimnélkülinek" nevezni.

A matematikusok ma még nem az agy nyelvét beszélik

A következőkben valójában az agynak csak egy erősen egyszerüsitett modelljét fogjuk vizsgálni. Ez a modell gyakorlóterülete a vizsgálatnak, a bizonyitásnak és a modellezéshez megfelelő szemlélet kialakitásának. Ez utóbbit különösen hangsulyozni kell: Egy a g y s z e r ü számitási rendszer megértésének lényeges alapfeltétele mindent elfelejteni, de mégis mindent tudni. /Sajnos a valóság nem veszi figyelembe(!) sem jelenlegi jól bevált számitási rendszereinket, sem a szép matematikai objektumokat és nem hajlandó(!) igazodni hozzájuk. Nincs más megoldás nekünk kell igazodni. Modellezéskor el kell felejteni őket/.

Látható, hogy a matematikai leirás számára szükség van a nyomok- és az ingerek, illetve ingerületek mint folyamatok algebrai strukturájára, az irányitott gráf megadására, egy sereg jelölésre és a modell müködésének algoritmusára. Mivel definiciójuk eltakarná a lényeget az ezután következőkben az informális leirást választottuk.

Példák bemutatása során folyamatosan fogjuk megadni a jelöléseket, és definiálni az algoritmus egy-egy részletét.

Informális bevezető a modellbe

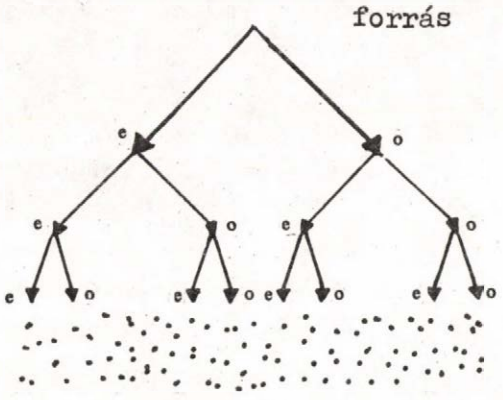
A modellben az inger és az ingerület /azaz az input/ egy diszkrét folyamat, melynek minden időpontjában /t/ az "eseményhalmaz" /E/ és az "operációhalmaz" /O/ egyesitésének néhány eleme következhet be. Ez a szám lehet 0 is.

Legyen például $E \doteq \langle e,f \rangle$ és $O \doteq \langle o \rangle$. Most, ha az /e,ef,o/ folyamat bekövetkezik, akkor:

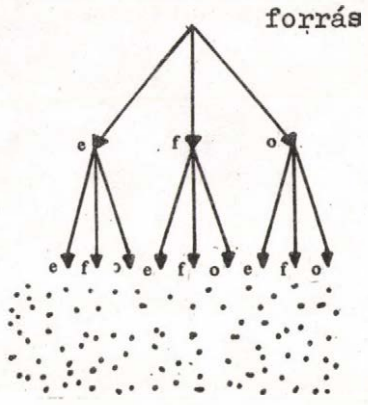
t = 1-ben e
t = 2-ben e és f
t = 3-ban o következik be, azaz a megfelelő időpontokban ez lesz az input. Az előző folyamat hossza 3 ütem volt. Egy h folyamat hosszának jele legyen $|h|$.

A modellt egy speciális, végtelen, irányitott gráfra /G/ épitjük az EUO halmaznak megfelelően, ami egy-egy értelmüen határozza meg a gráfot, az alábbi példák szerint:

Ha EUO ≙ ⟨e,o⟩ , akkor G:

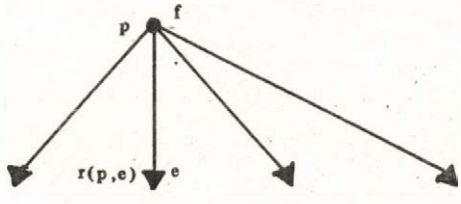


Ha EUO ≙ ⟨e,f,o⟩ , akkor G:



Legyen időtől független az EUO-beli elemek hozzárendelése a G csucsaihoz, és legyen a G csucsainak halmaza G_p.

Legyen $r : G_p \times /EUO/ \rightarrow G_p$, ahol $p \in G_p$ és $e,f \in EUO$ esetén:



Mint láttuk, az r/p,e/ csucs a p csucs e felé eső rákövetkezője.

Tartozzék a gráf minden éléhez egy időben változó számpár és nevezzük ezt nyomnak. Alakja /a,b/, ahol $a,b \in 0,1,2,...$

Ha $\overrightarrow{pr}$ egy él, akkor jelölje a hozzátartozó nyomot s/p,r/. Legyen s/p,r/ a kezdéskor /0,0/:

$\forall p \in G_p$ do $\forall e \in EUO$ do s/p,r/p,e//:=/0,0/;

Az előző sor a modell müködését definiáló algoritmus egy részlete volt.

Legyen t-ben a pozició P_t, a várható események és operációk halmaza V_t, a bekövetkezőké B_t. Ez egyelőre csak elnevezés. Definiciót az algoritmus megadásával kapnak majd.

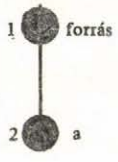
Nevezzük halmazváltozónak azt a halmazt, melynek elemeit egy algoritmus változtatja. Legyenek P_t, B_t, V_t halmazváltozók, melyekre $P_t \subseteq G_p$ és $V_t, B_t \subseteq$ $\subseteq$ EUO.

Tekintsünk egy példát. Legyen E $\doteq$ $\langle a,b,c \rangle$, O $\doteq \emptyset$ és /a,a,b,c,,,/ az inger. Ezen a példán fogjuk végigkövetni P_t, V_t, B_t és a nyomok változását ütemről ütemre.

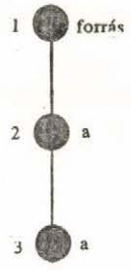
Az ábrákon mindig csak a gráf fontos részeit rajzoljuk fel. Először tekintsük át csak a pozició változásainak tulajdonságait a következő ábrákon. /Az algoritmus rögtön az ábrák után található/. Vegyük észre az inger és B_t kapcsolatát!

A pozició változásai az /a,a,b,c,,,/ folyamatban:

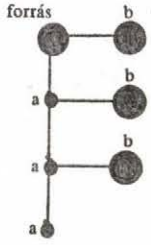
t=1
P_1
$B_1 = \langle a \rangle$



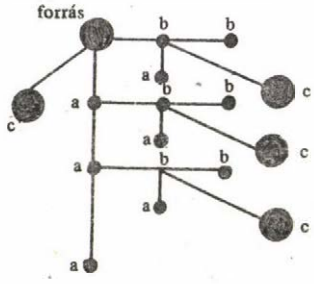
t=2
P_2létrejött t=1-ben
$B_2 = \langle a \rangle$



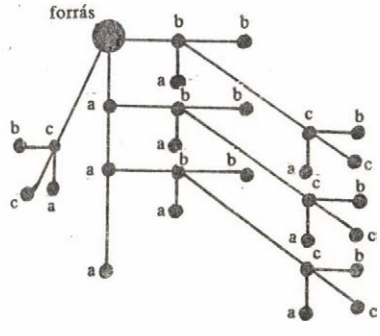
t=3
P_3létrejött t=2-ben
$B_3 = \langle b \rangle$



t=4
P_4létrejött t=3-ban
$B_4 = \langle c \rangle$



t=5
P_5létrejött t=4-ben
$B_5 = \emptyset$



t=6
P_6létrejött t=5-ben
$B_6 = \emptyset$

Az algoritmusban az "into/ , /;" a halmazbasorolás müveletét fogja jelenteni.

A pozicióváltozás algoritmusrészlete:

```
∀t∈T do into /forrás,P_t/;
t:=0;
...
again: ∀p∈P_t do
       ∀e∈B_t do
         begin
           ...
           into /r/p,e/,P_t+1/;
         end;
...
t:=t+1;
...
goto again;
```

/Tegyük fel, hogy az előző ábrasorozat egy mozgófilm néhány kockája volt. Képzeljük el a filmet, pontosabban az ábrasorozaton látható gráfot folytonos módosulásában. Mit látunk? Látjuk az aktiv pontok bizonyos szabályok szerinti áramlását és osztódását, látjuk, hogy mely pontok képesek egy-egy pillanatban aktivizálódni, látjuk a gráf rajzának növekedését. Erre a szemléletmódra a későbbiekben igen nagy szükségünk lesz./

Most megadjuk az algoritmus ujabb részleteit. Az előző algoritmusrészlet bele fog épülni a müködést definiáló algoritmusba. Az ujabb vagy módositott részeket a vastag csik fogja jelezni.

A megértéshez szükségesek az alábbiak:

Legyen $/a,b/ \oplus /c,d/ \stackrel{\text{def}}{=} /a+c,b+d/$.

Legyen az algoritmusbeli q függvény a következőképpen definiálva:

Ha $\sigma_e = /a,b/$ és $r \in (0,1)$, akkor

$$q/\sigma_e,r/ = \begin{cases} \underline{\text{true}} & \text{ha } a/(a+b) > r \\ \underline{\text{false}} & \text{ha } a/(a+b) \leq r \end{cases} \quad \text{ahol } 0/0 \doteq 0$$

Az algoritmusban r egy küszöb lesz.

```
comment                    1. Kezdő értékadások;
∀t ∈ T do
begin
  V_t := B_t := ∅;
  P_t := ⟨forrás⟩ ;
end:

∀p ∈ G_p do
∀e ∈ EUO do
    s/p,r/p,e//:=/0,0/;
t:=0;
comment Ezek voltak a kezdő értékadások;
. . .
comment                    2. Pozició- és nyomváltozás;
again: ∀p ∈ P_t do
       ∀e ∈ B_t do
        begin
          into /r/p,e/,P_{t+1}/;
          s/p,r/p,e//:=s/p,r/p,e// ⊕ /1,0/;
        end;
        comment  Ez volt a pozició- és a "pozitiv" nyomváltozás;
         ∀p ∈ P_t do
         ∀e ∉ B_t do
             if e ∈ V_t then s/p,r/p,e//:=s/p,r/p,e//⊕/0,1/;
        comment Ez volt a "csalódás" vagy másszóval a "negativ"
                nyomváltozás

comment                    3. Prognózis;
∀e ∈ EUO do Σ_e :=/0,0/;
∀p ∈ P_{t+1} do
∀e ∈ EUO do
     Σ_e := Σ_e ⊕ s/p,r/p,e//;
comment A "gyakoriságok" összegeződtek;
∀e ∈ EUO do if q/Σ_e, várakozási küszöb/ then into /e,V_{t+1}/;
comment Meghatározódott, hogy mi várható;
 . . .
t:=t+1;
 . . .
goto again;
```

Tekintsük át ismét az előző példát az algoritmus szerint és képzeljük el a változásokat a mozgófilmen is! Várakozási küszöb = 0.40

t=0 : /a,a,b,c,,,/

RÉGI:
$B_t = \emptyset$
$V_t = \emptyset$
$P_t = \langle 1 \rangle$

UJ:
$P_{t+1} = \langle 1 \rangle$
$\sum_a = /0,0/$
$\sum_b = /0,0/$
$\sum_c = /0,0/$
$V_{t+1} = \emptyset$

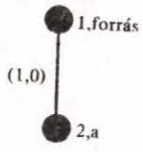
G: 1,forrás

t=1 : /a,a,b,c,,,/

RÉGI:
$B_t = \langle a \rangle$
$V_t = \emptyset$
$P_t = \langle 1 \rangle$

UJ:
$P_{t+1} = \langle 1,2 \rangle$
$\sum_a = /1,0/$
$\sum_b = /0,0/$
$\sum_c = /0,0/$
$V_{t+1} = \langle a \rangle$

G:

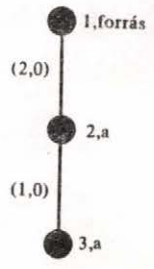


t=2 : /a,a,b,c,,,/

RÉGI:
$B_t = \langle a \rangle$
$V_t = \langle a \rangle$
$P_t = \langle 1,2 \rangle$

UJ:
$P_{t+1} = \langle 1,2,3 \rangle$
$\sum_a = /3,0/$
$\sum_b = /0,0/$
$\sum_c = /0,0/$
$V_{t+1} = \langle a \rangle$

G:

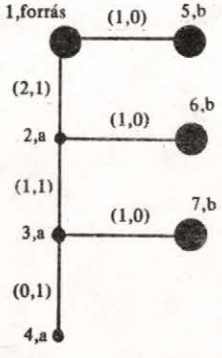


t=3 : /a,a,b,c,,,/

RÉGI:
$B_t = \langle b \rangle$
$V_t = \langle a \rangle$
$P_t = \langle 1,2,3 \rangle$

UJ:
$P_{t+1} = \langle 1,5,6,7 \rangle$
$\sum_a = /2,1/$
$\sum_b = /1,0/$
$\sum_c = /0,0/$
$V_{t+1} = \langle a,b \rangle$

G:

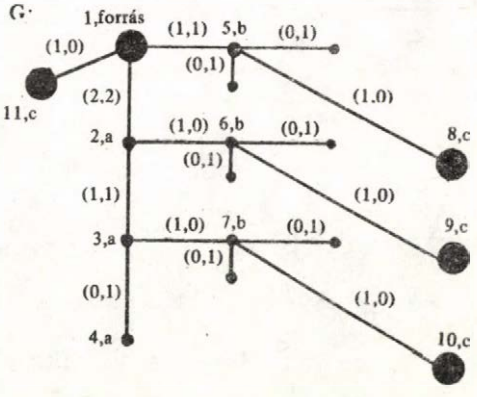


t=4 : /a,a,b,c,,,/

RÉGI:
$B_t = \langle c \rangle$
$V_t = \langle a \rangle$
$P_t = \langle 1,5,6,7 \rangle$

UJ:
$P_{t+1} = \langle 1,8,9,10,11 \rangle$
$\sum_a = /2,2/$
$\sum_b = /1,1/$
$\sum_c = /1,0/$
$V_{t+1} = \langle a,b,c \rangle$

G:

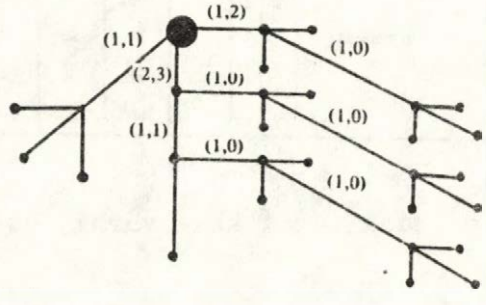


t=5 : /a,a,b,c,,,/
↑
Ez egy "szünet"

RÉGI:
$B_t = \emptyset$
$V_t = \langle a,b,c \rangle$
$P_t = \langle 1,8,9,10,11 \rangle$

UJ:
$P_{t+1} = \langle 1 \rangle$
$\sum_a = /2,3/$
$\sum_b = /1,2/$
$\sum_c = /1,1/$
$V_{t+1} = \langle c \rangle$

G:



RÉGI:

t=6 : /a,a,b,c,,,/

$B_t = \emptyset$

$V_t = \langle c \rangle$

$P_t = \langle 1 \rangle$ Ez a "nyugalmi pozició" /hasonlit a "dinamikus stop"-hoz/

UJ:

$P_{t+1} = \langle 1 \rangle$

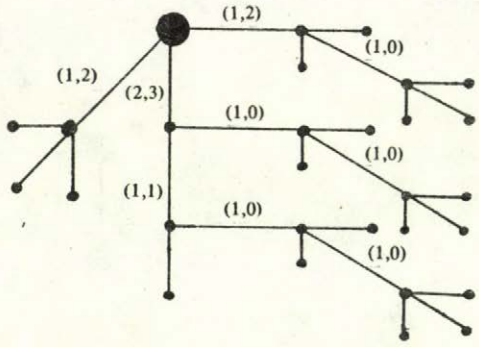
$\sum_a = /2,3/$

$\sum_b = /1,2/$

$\sum_c = /1,2/$

$V_{t+1} = \emptyset$

G:



t=7 : /a,a,b,c,,,/

RÉGI:

$B_t = \emptyset$

$V_t = \emptyset$ } Ez a "nyugalmi állapot".

$P_t = \langle 1 \rangle$

UJ:

$P_{t+1} = \langle 1 \rangle$

$\sum_a =$

$\sum_b =$ } Ezek maradnak változatlanul

$\sum_c =$

$V_{t+1} = \emptyset$

Láthatjuk, hogy a várakozás módosult. "Programok" keletkeztek a gráfon nyomok formájában, amelyek várakozást vezérelnek. Ha az ingerlést többször egymásután megismételnénk, akkor a következőket tapasztalnánk:

A folyamat			a	a	b	c				
A várakozás	Kisérlet szám	1.		a	a	ab	abc	c		
		2.		ab	ab	c		ac	a	
		3.		ab	ab	c		abc	ac	
		4.		ab	ab	c		ac	a	
		5.		ab	ab	c		abc	ac	

Láthatjuk, hogy a modell amit kell várni, azt várja, a kezdő ütem kivételével.

Képzeljük el a következőket a "mozgófilmen" is!

Láttuk, hogy ha egy időpontban a folyamatban "szünet" van, akkor létrejön a nyugalmi pozició. Nevezzük folytonos folyamatnak azt az f folyamatot, melyben nem létezik szünet. Jelben: $f \in C$

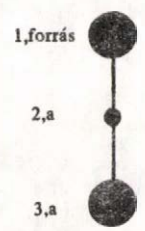
Nevezzük pozíciónak a $P \subseteq G_p$ halmazt, melyre igaz, hogy $1 \in P$. Jelben: $P \in \mathcal{P}$

Jelölje e/q/ a $q \in G_p$ ponthoz tartozó EUO-beli elemet. $e/q/ \in EUO$

Mint láttuk, van egy irányitás a gráfon. Ilyen értelemben a gráf félig rendezett. Legyen $p,q \in G_p$ esetén $p < q$, ha lehetséges p-ből q-ba elmenni. Legyen q megelőzője m/q/ a $<$ relációval meghatározott, és legyen $m/1/ \doteq 1$.

Legyen $M/P/ \doteq \langle m/p/ : p \in P \rangle$ ha $P \in \mathcal{P}$ és $P \neq \langle 1 \rangle$. Ez a P pozició megelőzője. Bizonyitható, hogy a megelőző, nemcsak gráf értelemben az, hanem a modell működési algoritmusa értelmében is az. Nem tudunk mondani semmit sem $\langle 1 \rangle$ megelőzőjéről a szünet már látott tulajdonsága miatt.

Nevezzük valós poziciónak /R/ azt a poziciót, mely egy nyugalmi pozicióból egy folyamat bekövetkezése végén létrejöhet. Jelben: $R \in \mathcal{R}$. Például, mint láttuk $P = \langle 1,2,3 \rangle \in \mathcal{R}$, mert $P = P_3$ az /a,a,b,c,,,/ folyamat /a,a/ részfolyamatában.



Tegyük fel, hogy $P = \langle 1,3 \rangle \in \mathcal{R}$. Ekkor $M/P/ = \langle 1,2 \rangle$ lenne. De vajon hogyan jött létre P az M/P/-ből? "a" bekövetkezett, mivel e/3/=a. De akkor $P = \langle 1,2,3 \rangle$ lenne. Ellentmondás, tehát $P \notin \mathcal{R}$.

<u>A poziciók és az algoritmus néhány tulajdonsága:</u> /ne feledjük a "mozgófilmet"/

1. Ha $q \in G_p$ és $q \neq 1$, akkor $q \in P_{t+1} \iff m/q/ \in P_t$ és $e/q/ \in B_t$.
   Azaz, ha q nem a forráspont, akkor q egy időpontban akkor és csak akkor lesz a pozició eleme, ha q megelőzője eleme volt az előző poziciónak és a q-hoz tartozó esemény bekövetkezett.
2. Az áramlás "sebessége" 1 él/ütem a gráfon.
3. Minden folyamat egyértelmüen meghatározza a végpoziciót, mely létrejön bekövetkezésekor. A folytonos folyamatok egy-egy értelmüen határozzák meg.
4. Az áramlás egy-egy értelmü kódja a folytonos folyamatnak. Például különböző folytonos folyamatok különböző pályák mentén módositják a nyomokat, azonosak azonos pályák mentén feltéve, hogy ugyanaz a kezdőpoziciójuk.

Legyen $p \in G_p$ esetén $\|p\|$ az 1 ponttól a p ponthoz vezető ut hossza. Nevezzük ezt a p pont szintjének. Poziciók szintjét hasonlóan definiálhatjuk:

$$\|P\| \doteq \max_{p \in P} \|p\| \quad \text{minden } P \in \mathcal{P} \text{-re}$$

5. Létezik minden $R \in \mathcal{R}$ valós poziciohoz egyetlen olyan $g \in C$ folytonos folyamat, melynek bekövetkezésekor a végpozició R, ha a kezdőpozició a nyugalmi pozició. Legyen $g \doteq f/R/$.

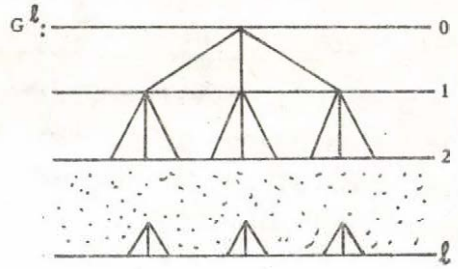
6. Egy $g \in C$ folytonos folyamat hossza egyenlő azon $R \in \mathcal{R}$ végpozició szintjével, mely létrejön, mikor a g folyamat bekövetkezik és a kezdőpozició a nyugalmi pozició. Azonban 5. miatt $g=f/R/$, igy $|f/R/| = \|R\|$ minden $R \in \mathcal{R}$-re.

7. Ha $\|P_1 \cap P_2\| = k > 0$, akkor $\|M/P_1/ \cap M/P_2/\| = k-1$ minden $P_1, P_2 \in P$-re

8. Ha $R \in \mathcal{R}$ és $\|R\| \geq 1$, akkor $M/R/ \in \mathcal{R}$, valós pozició megelőzője is valós pozició.

9. Ha $R_1, R_2 \in \mathcal{R}$, akkor $R_1 \cap R_2 \in \mathcal{R}$. Valós poziciók közös része is valós pozició.

10. Legyen G^ℓ egy részgráf, mely a G felső ℓ szintjéből áll.
G^ℓ valós pozició, tehát ha $R \in \mathcal{R}$, akkor $R \cap G^\ell \in \mathcal{R}$. Egy folytonos folyamat utolsó ℓ üteme egy-egy értelmüen kódolható G^ℓ-n is /vesd össze 4-gyel/. Ez egy végesités lehetőségét mutatja.



11. Legyen:

$m \doteq |EUO|$ az események és operációk száma
$\ell \doteq$ a szintek száma
$n \doteq |G_p^\ell|$ a pontok száma
$r \doteq |\mathcal{R}^\ell|$ a valós poziciók száma G^ℓ-n

Ekkor: $$\lim_{\ell \to +\infty} \frac{\vartheta}{\vartheta \log r} = \frac{m + \frac{\vartheta}{\vartheta \log}\left(1 - \frac{1}{2^m}\right)}{\frac{\vartheta}{\vartheta \log} m} \approx \frac{m}{\frac{\vartheta}{\vartheta \log} m}$$

Ez segit nekünk eldönteni, hogy mikor milyen "gráfhardware"-t célszerü épiteni nagyszámu alkatrészből.

Eddig a példa csak a modell passziv adaptációját mutatta be, az algoritmus eddig nem definiálta az operációkat. Most az algoritmus egy ujabb bővitése következik.
Legyen az "operációs küszöb" nagyobb mint a várakozási küszöb, és nézzük a teljes algoritmust!
Az algoritmusban található "inner i ; " utasitás azt jelzi, hogy arra a helyre változó utasitássorozat kerül majd be.

```
comment                    1. Kezdő értékadások;
 ∀t∈T do
begin
  V_t:=B_t:=∅;
  P_t:= <forrás> ;
end;
 ∀p∈G_p do
 ∀e∈EUOdo s/p,r/p,e//:=/0,0/;
t:=0;
comment Ezek voltak a kezdő értékadások;
inner 1;
comment Itt történik a vizsgált folyamat és a kezdeti nyomok betöltése a
  B_t halmazokba és a gráfba;
comment                    2. Pozició- és nyomváltozás;
again: ∀ p∈P_t do
       ∀ e∈B_t do
         begin
           into /r/p,e/,P_t+1/;
           s/p,r/p,e//:=s/p,r/p,e//⊕/1,0/;
         end;
         comment Ez volt a pozició- és a "pozitiv"nyomváltozás;
          ∀p∈P_t do
          ∀e∉B_t do if e∈V_t then s/p,r/p,e//:=s/p,r/p,e// ⊕ /0,1/;
         comment Ez volt a "csalódás", a "negativ" nyomváltozás;
comment                    3. Prognózis;
 ∀e∈EUO do ∑_e :=/0,0/;
 ∀p∈P_t+1 do
 ∀e∈EUO do ∑_e :=∑_e ⊕ s/p,r/p,e// ;
comment A "gyakoriságok" összegeződtek;
 ∀e∈EUO do if  q/∑_e, várakozási küszöb/ then into /e,V_t+1/;
comment Meghatározódott, hogy milyen esemény és operáció várható;
 ∀o∈O  do if  q/∑_o, operációs küszöb/ then into /o,B_t+1/;
comment Meghatározódott, hogy a modell mit fog operálni t-ben, igy melyik
        operáció bekövetkezését fogja érzékelni t+1-ben;
```

```
t:=t+1;
inner 2;
comment A környezet itt válaszolhat az operációkra és a kisérlet itt fe-
        jeződhet be;
goto again;
```

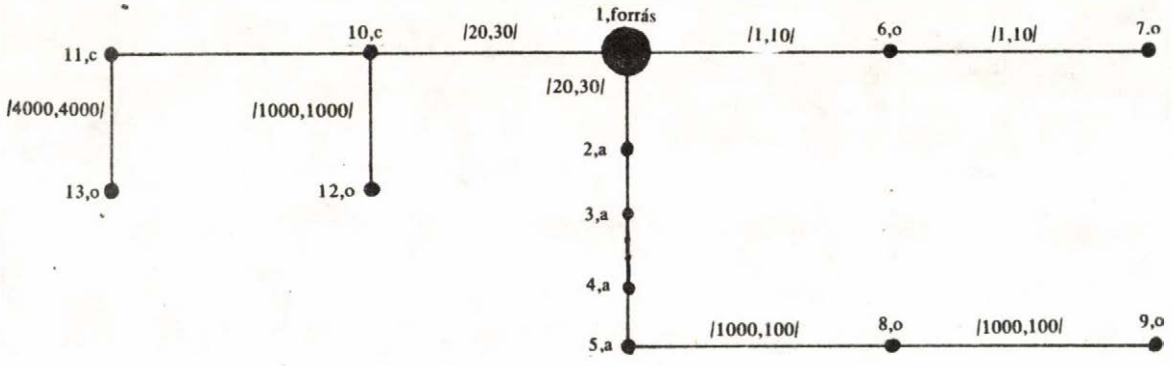
Tekintsünk egy uj példát! Ez egyben a modell egy alapvető tesztje is. Egy Pavlov féle kisérletet fogunk szimulálni. Egy feltétlen reflexet betöltünk a gráfba és vizsgáljuk a feltételes reflex létrejöttét.

Legyen $E \doteq \langle a,b,c \rangle$ és $O \doteq \langle o \rangle$, ahol

a = ételérzet
b = a csengő hangjának érzete
c = tul sok nyál érzete
o = nyálelválasztás /operáció/

Legyen a "belső környezet" törvényszerüsége a következő: Ha t-ben étel nincs /$\bar{a}$/, de nyálelválasztás/o/ van, akkor bekövetkezik a tul sok nyál /c/ érzete. Jelben: $\bar{a}o \Longrightarrow c$

Legyen a feltétlen reflex:



A be nem rajzolt nyomok értéke legyen /0,0/. Legyen a várakozási küszöb 0.40, az operációs küszöb 0.60.

Az /a,a,a,a/ folyamat bekövetkezése után végpozicióként a $P_5 = \langle 1,2,3,4,5 \rangle$ pozició jön létre. Mivel $5 \in P_5$ az s/5,8/=/1000,100/ nyom az o operáció bekövetkezését vezérli /$o \in B_5$/. Ekkor $8 \in P_6$ lesz és az s/8,9/ nyom ismét operációt vezérel. A feltétlen operáció tehát müködik: Ha a modell 4 ütem hosszan "érzi" az ételt, akkor "nyálat választ el" 2 ütem hosszan.

Ha t-ben operáció van, de "ételérzet" nincs, akkor bekövetkezik a "tul sok nyál érzete" /$c \in B_t$/. Ekkor $10 \in P_{t+1}$ és az s/10,12/=/1000,1000/ nyom egy speciális vezérlést "javasol".

q/ /1000,1000/, várakozási küszöb /=true, de
q/ /1000,1000/, operációs küszöb /=false

Esetünkben s/10,12/ "javaslata" érvényesül, más nyomok "javaslatával" szemben, tehát o várható lesz, de nem következik be. Nevezzük ezt a vezérlést "destruktivitás vezérlésnek". A "feltétlen destruktivitás" tehát működik.

Ismételjük most az /a,a,a,ab,,,,,,/ folyamatot egymásután sokszor /≈80/, mindig a nyugalmi pozicióból indulva. /Mikor a modell "ételt kap" a "csengő megszólal"/. Igy, ha t-ben kezdtünk, akkor $P_{t+4} = \langle 1,2,3,4,5,p,\ldots\rangle$, ahol p=r/1,b/. Mivel o bekövetkezik, ezért az s/p,r/p,o// nyom ennek megfelelően módosul: Első koordinátájának értéke mindig nagyobb és nagyobb lesz.

Most következzék be néhányszor /b,a,a,,,/. Ekkor, ha t_1-ben kezdtünk, akkor $p \in P_{t_1+1}$ és igy az s/p,r/p,o// nyom "szót kapott". o bekövetkezik, tehát a feltételes reflex kialakult. Ha csak a "csengő szól" is, van "nyálelválasztás".

Most következzék be sokszor /b,b/. Ekkor, ha t_2-ben kezdtünk, ugy $p \in P_{t_2+1}$ miatt csakugy, mint az előbb o bekövetkezik, de nem "talál" a-t, bekövetkezik tehát c is. Igy $10 \in P_{t_2+2}$ miatt az s/10,12/ nyom "szót kapott". Aktivizálódik a "feltétlen destruktivitás". $p \in P_{t_2+2}$ és $o \in V_{t_2+2}$, de $o \notin B_{t_2+2}$. Emiatt az s/p,r/p,o// nyom második koordinátája nő 1-gyel. Az s/p,r/p,o// nyom egyre "kevésbé" fog operációt vezérelni. A kialakult program megváltozik, megszünik. A "csengő" már nem vezérel "nyálelválasztást".

Láttuk, hogy az s/5,r/5,o// nyom akkor és csakis akkor "kap szót", mikor az áramlás végigfolyt a forráspontból azon út mentén, mely 5-be vezet. De ekkor /a,a,a,a/ bekövetkezett! Általánosan is igaz, egy pont akkor és csakis akkor aktivizálódik, mikor az ő ugynevezett "belépési folyamata" bekövetkezett. Legyen a p pont belépési folyamatának jele b/p/. Például: b/5/=/a,a,a,a/ illetve b/10/=/c/.

Amikor a környezet vagy a modell operációi által lejátszódik egy p pont b/p/ belépési folyamata, akkor mondhatjuk azt, hogy "a p pont vezérlést kapott".

Most már látjuk azt is, hogy mit célszerü programozásnak nevezni a modell esetében. "Forrásprogramozásnak" nevezzük a modellt érő ingerek előirását. "Tárgyprogramozásnak" nevezzük a modell egy állapotának megadását: a nyomok és az aktiv pontok előirását (például t = 0-ra).

/Vegyük észre, hogy a modellt "kétfelől" lehet programozni. Azonban ez egyáltalán nem meglepő, hiszen egy számitógép esetében is tulajdonképpen ugyanez a helyzet: Ott az "adatokat" tekinthetjük a második programnak/.

Egy program tervezésekor nagyon zavaró, hogy nem tudjuk jól elképzelni a modell létrejövő állapotait, hiszen a nyomok nagymértékben változnak a "tárgyprogram" "futása" alatt, például a "forrásprogram" által. Jó volna elhanyagolni a lényegtelen nyomváltozásokat és kiemelni a programban szereplő nyomok által előirt vezérlés lényegét.

Rendkivül nehéz megtalálni az egy-egy "tárgyprogramnak" megfelelő nyomokat, számpárokat. Jó volna a nyomok számszerü meghatározását gépesiteni.

Rendkivül nehéz megvizsgálni, hogy egy "tárgyprogram" vajon jó vagy hibás-e.

Többek között az előzők miatt célszerü egy szimbolikus programozás bevezetése. Mindenesetre ezzel elindulunk a modellt realizáló "gráf-hardware"-hez tartozó "assembly" nyelv felé.

Egy példa szimbolikus programozásra:

Legyen a szimbólumok halmaza S_k.

$$S_k = \langle g^1, g^2, \ldots, g^n, \ldots \rangle \cup \langle v^1, v^2, \ldots, v^n, \ldots \rangle \cup \langle o^1, o^2, \ldots, o^n, \ldots \rangle$$

/A "g" gátlást, a "v" várakozást, az "o" operációt vezérlő nyom lesz majd, ha definiáljuk./

Legyen ekvivalens minden S_k-beli szimbolum egy olyan számpárral, melyre igazak a következők:

1. Legyen $s \in S_k$-ra $q/s, \text{küszöb}/ = \begin{cases} \underline{\text{false}} \text{ ha } s=g^i \text{ vagy} \\ \qquad s=v^i \text{ és küszöb=operációs küszöb} \\ \underline{\text{true}} \text{ ha } s=v^i \text{ és küszöb=várakozási küszöb} \\ \qquad \text{vagy } s=o^i \end{cases}$

Jelöljük a q/a, küszöb/ = q/b,küszöb/ relációt "a∼b"-vel.

2. Legyen a $\oplus$ müvelet asszociativ és kommutativ és legyenek igazak a következő átalakitások: Az S_k halmaz elemeiből álló legfeljebb k tagu összegekben

$g^i \oplus v^i \sim g^i \oplus o^i \sim g^i$ és $v^i \oplus o^i \sim v^i$ minden i természetes számra.

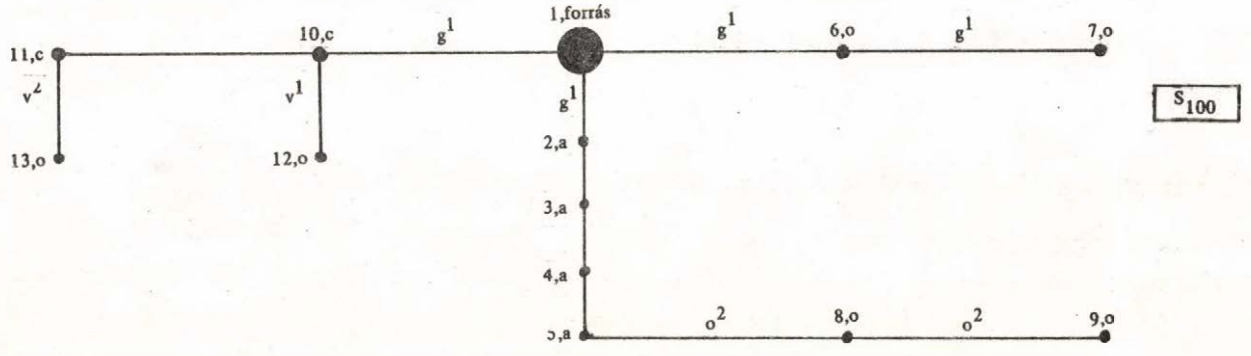
$s_1 \oplus s_2 \sim s_1$ ha az $s_1 \in S_k$ indexe nagyobb $s_2 \in S_k$ indexénél.

Például: $g^7 \oplus v^7 \oplus o^7 \oplus o^{13} \oplus o^{29} \oplus g^{29} \sim g^{29}$ ha a szimbolumok az S_{6+j} /j=0,1,.../ halmaz elemei.

Végtelen sok S_k létezik minden k természetes számra. Egy S_n is S_k tulajdonságu, ha $n \geqq k$.

Vegyük észre, hogy ha vezérlés becslésekor csak az S_k elemeivel kell számolnunk, akkor S_k tulajdonságai lehetővé teszik a fejben számolást a modell algoritmusával. 2. szerint egyszerübb lesz az $e \in EUO$-hoz tartozó $\sum_e$ meghatározása, 1. szerint pedig a $q/\sum_e$, küszöb/ értékét kaphatjuk meg gyorsabban. Azt, hogy becsléskor csak az S_k elemeivel kelljen számolnunk, elérhetjük ugy, hogy S_k elemeit elég nagy számokból álló nyomokból választjuk.

Most megadjuk az előző Pavlov féle programot egy S_{100} szimbolikus programozás segitségével:



Igaz, hogy S_∞ nem létezik, de mi elképzelhetünk ilyen tulajdonságu szimbólum halmazt. Vegyük az S_∞ nyomhalmazt is a modell axiómái közé. Jelöljük S_∞ elemet alsó indexszel.

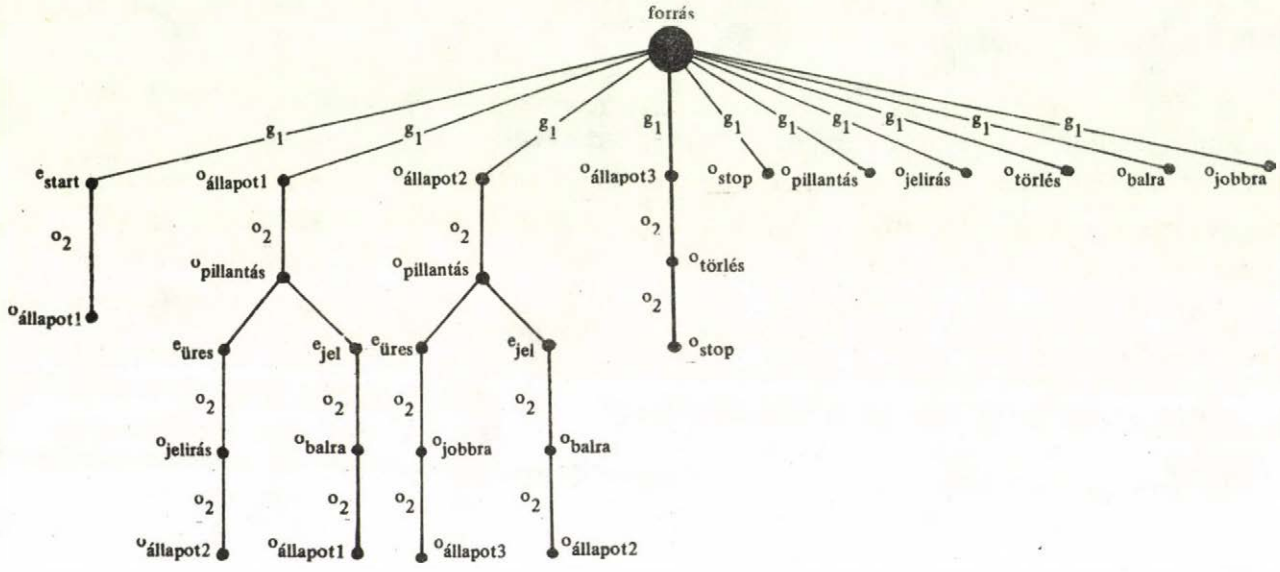
$$S_\infty = \langle g_1, g_2, \ldots, g_n, \ldots \rangle \cup \langle v_1, v_2, \ldots, v_n, \ldots \rangle \cup \langle o_1, o_2, \ldots, o_n, \ldots \rangle$$

/Lényegében az ilyen szimbólummal jelzett élekhez rendelt nyomokat egyrészt "maximálissá" tettük, másrészt változásukat kikapcsoltuk/.

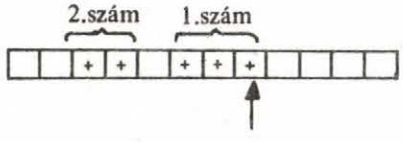
Térjünk át más példára. Itt az ideje, hogy megmutassuk, hogy a modell más szempontból is értékes számitási rendszer. Példaképpen nézzünk egy két számot összeadó Turing-gépet szimuláló "tárgyprogramot". Legyenek:

$E \doteq \langle e_{start}, e_{üres}, e_{jel} \rangle$ és $O \doteq \langle o_{pillantás}, o_{jelirás}, o_{törlés}, o_{balra}, o_{jobbra}, o_{állapot1}, o_{állapot2}, o_{állapot3}, o_{stop} \rangle$

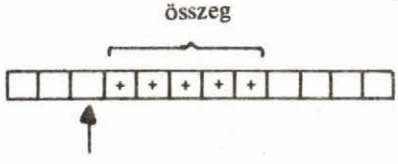
/Egy t-beli $o_{pillantás}$ operációt képzeljünk ugy el, hogy hatására bekövetkezik a "papirszalag" aktuális cellájának észlelése t+1-ben/



Tekintsük az ábrán látható "tárgyprogramot". Mi történik, ha e_{start} bekövetkezik egy nyugalmi pozicióban? Látható, hogy a beirt "tárgyprogram" végrehajtódik és a modell pontosan egy két számot összeadó Turing-gépet szimulál. Például, ha a "papirszalag" állapota e_{start} bekövetkezésekor



akkor az o_{stop} bekövetkezésekor a következő lesz:



Minden T Turing-géphez a bemutatott módon meg tudjuk adni az EUO halmazt, az ezen halmazhoz tartozó G gráfot és azt a Turing-tárgyprogramot, mely az e_{start} bekövetkezésekor megkezdi a T-Turing-gép szimulálását. De vajon létezik-e olyan rögzitett EUO halmaz és hozzátartozó G gráf, melyre tetszőleges T Turing-gépet szimuláló "tárgyprogram" ráirható?

A példa szerinti programozási módnál az EUO halmaz nem lehetett rögzitett, hiszen egy Turing-gépnek tetszőlegesen sok állapota lehet, igy az $o_{állapotx}$ operációk száma is tetszőlegesen nagy lehet.

Ez azonban az operációk kódolásával például a következő módon küszöbölhető ki:

Ha az $o_{állapotx}$ operációk száma n, akkor legyen $K \doteq \left[\log_2 n\right] + 1$ és tekintsük a következő megfeleltetéseket:

$$o_{állapot1} \rightarrow \overbrace{00\ldots001}^{K}{}_2 \rightarrow /o_{nulla}, o_{nulla}, \ldots, o_{nulla}, o_{nulla}, o_{egy}/$$

$$o_{állapot2} \rightarrow 00\ldots010_2 \rightarrow /o_{nulla}, o_{nulla}, \ldots, o_{nulla}, o_{egy}, o_{nulla}/$$

stb.

A példabeli gráf előzőek szerinti módositásával már "univerzális gráfot" kapunk, hiszen tetszőleges Turing-tárgyprogram az előbb bemutatott módon beirható rá.

A modell adaptivitásának vázlatos jellemzése

Ebben a részben különösen fontos a "mozgófilm" szemléletmód!

1. Láthattuk, hogy a modell lényege egy speciális kódolás és dekódolás. A bekövetkezett folyamatok kódolódnak a pozició és a nyomok változásává, mig később, az éppen aktuális pozició és a vele meghatározott nyomok dekódolódnak vezérléssé. /A modell müködésének ezen szintjén tehát az "adatok" átminősülése vezérléssé automatikus, nem pedig programvezérelt./ Láttuk, hogy a folytonos folyamatok egy-egy értelmüen kódolódnak a pozició változásává.

2. Láthattuk, hogyha egy nyom változik, akkor ugy változik, hogy "célszerübben" fog vezérelni egy ugyanolyan pozicióban később. Például, ha kialakul egy várakozás, de a várt esemény vagy operáció nem következik be, akkor a nyom módosulása olyan, hogy kevésbé fog várakozást vezérelni később. Ilyen értelemben mondhatjuk, hogy ekkor egy "destruktiv" feli-

déződés" történt.

Tudjuk, hogy az operációk bekövetkezése csak a modelltől magától függ. Igy talán elkerülhető lesz a destruktivitás egy operációkból álló kóddal. /Tudjuk, hogy az ember szintén ilyet "használ". Az ember beszél, és tudjuk a pszichológusoktól, hogy a gyerekek elsősorban beszéd által tanulnak meg gondolkodni. Gondoljuk meg, hogy vajon a beszéd tényleg egy operációkód-e!/. A közeljövőben meg akarjuk vizsgálni a modell tulajdonságait, ha létezik környezetének egy operációkódja/nyelve/és a környezet a modellt tanitani akarja.
Láttuk, hogy igaz, hogy a szituáció, tehát a pozició cimzi a történéseket a gráfon. A pozició határozza meg, hogy mi elevenedjék fel a multból, és hova tárolódjék a jelen. A pozició lényegében iró-olvasó pointerek halmaza.

3. Az aktuális pozicióban nemcsak egyetlen folyamat idéződik fel, hanem különböző folyamatok részei, melyek a multban különböző időben történtek. Igy a modell olyan eseményt vagy operációt is vár, mely "ilyenkor" sohasem történt vele. Tehát képes uj várakozás vezérlést "alkotni". Ez egy passziv-asszociáció.

A modell igy nemcsak felidéz, reprodukál, de inkább "összeidéz", "koprodukál"./ Nem tudjuk, hogy ez utóbbi két szó vajon érthető-e. Ha nem, kérjük változtassák meg egy jobb szóra./

4. A modellben nem létezik ugynevezett memóriaegység. Az egész gráf a memória és a pozició cimzi a memóriatevékenységeket.
A poziciónál lévő nyomok csak együttesen határozzák meg az aktuális vezérlést. Külső szemléletből a tárolás eloszlott, mig belső szemléletből sokszoros.

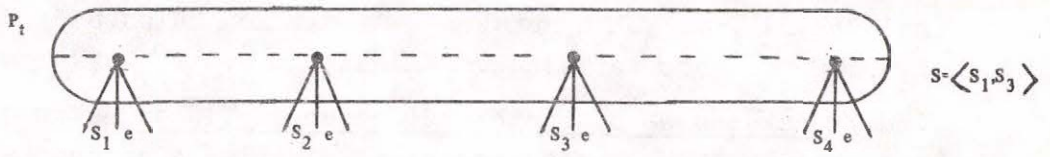
5. Mivel a tárolás sokszoros és eloszlott, ha egy gráfterület megsérül és a sérülés nem tul nagy, akkor a vezérlés lényege nem változik. Igy a modell ilyen értelemben védett kis sérüléstől.

6. Mivel részben a korábban kialakult nyomok határozzák meg a későbbi vezérlést, tekinthetjük a nyomokat "tárgyprogramokként", melyek a környezet hatásaiból mint "forrásprogramokból" keletkeznek - automatikusan! Ilyen értelemben tárgyprogramok keletkeznek, változnak és "elmulnak".

7. A pozició az aktiv pontok halmaza, igy az aktiv pontokban paralel gráfakciók történnek. Ez egy multiprocessing.

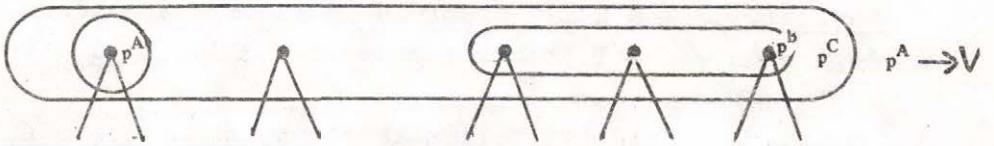
Az aktiv pontok együttesen határozzák meg a vezérlést. Elképzelhetjük ezt ugy, hogy a pontok "küzdenek" egymással. Mindegyik igyekszik érvényre juttatni azt a vezérlést, melyet a hozzá tartozó nyomok határoznak meg, de végülis a "szavazás" dönt. A szavazás eredménye egy vezérlés és lehet, hogy bizonyos nyomok /S/ szintén pontosan ugyanazt a vezérlést. "javasolták".

Például:



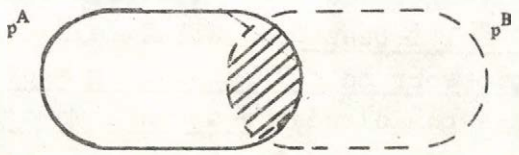
Tudjuk, hogy a P_t pozició pontjaiból kiinduló élekhez tartozó nyomok a bekövetkezett vezérlésnek megfelelően módosulnak. Ha a bekövetkezett vezérlés azonos S "javaslatával", vagyis, ha S prognózisa jó lett, akkor mondhatjuk, hogy ezek a nyomok S-nek megfelelően módosultak. Ilyen értelemben S vezérlése "részben" lemásolódik.

8. Az ugynevezett feltételes vezérlés egy másolás által jön létre.
Például:



Képzeljük el, hogy egy "erős" V vezérlés van P^A-ban. Ha P^A és P^B sokszor lesz együtt aktiv P^C részeként, akkor a vezérlés "átmásolódik" P^A-ból P^B-be. Igy jön létre a feltételes vezérlés. Most, ha P^B egyedül lesz aktiv, akkor a vezérlés már tartalmazni fogja V-t.

9. Az ugynevezett generalizálódás is másolás által jön létre.
Például:



Képzeljük el, hogy sokszor volt P^A aktiv, de csak kevésszer P^B. Igy a közös rész vezérlése a másolás miatt azonos P^A vezérlésével. Ha most P^B aktivizálódik, akkor a vezérlés hasonlitani fog a közös rész miatt. Ilyen értelemben P^A vezérlése általánossá válik, generalizálódik.

10. Később a P^A és P^B poziciók közös része valószinüleg "semlegessé" válik a gátlások és destruktivitások segitségével. Ez a diszkriminálódás.

Események esetében ezt általában a környezet törvényszerüségei hozzák létre. Operációk esetében azonban a gátlások.
A diszkriminálódás részletes vizsgálatát különösen fontos jövőbeli feladatunknak tartjuk.

11. Lehetséges az is, hogy $P_{t_1}=P^A$ és $P_{t_2}=P^B$ valamilyen folyamat t_1 és t_2 időpontjában. Ekkor olyan vezérlések generalizálódhatnak és diszkriminálódhatnak, melyek ugyanazon folyamat két különböző időpontjához, t_1-hez és t_2-höz tartoznak. Például, ha $t_1 > t_2$, akkor a t_1-beli vezérlés /várakozás és operáció/ "előrejön az időben és automatikusan optimalizálódik. Nevezhetjük ezt elemi elháritó adaptációnak, vagy reakcióidő optimalizálódásnak.

12. Ha a modell olyat vezérel, mely nem általános, akkor külső szemléletből a modell indukciót hajtott végre!!! Hiszen a modell egyedit "tekintett" általánosnak ekkor, és azért vezérelt ugy.
Hasonlóan magyarázhatnánk a dedukció , az analizis , a szintézis , stb. alapjait. Azonban ettől még a modell nem gondolkodik, csak jól vezérel. Csak a külső szemlélő hajlamos azt mondani, hogy "a modell logikusan vezérel, tehát a modellnek »logikája« van."

13. Ha egy pont belépési folyamata operációk sorozata és a pont vezérlést kap, akkor azt mondjuk, hogy elemi aktiv-asszociáció történt.
Az magyarázza az elnevezést, hogy a beszéd operációk sorozata és a gondolkodás /igy az aktiv asszociáció is/ a beszédből jön létre, pontosabban az ugynevezett belső beszédből, mely együttjár a külső beszéddel. Azonban a gondolkodás csak kialakulásának kezdetén olyan szekvenciális operációsorozat, mint a külső beszéd. Az előzőekben vázolt programmásolódás, feltételes vezérlés kialakulás, generalizálódás és diszkriminálódás segitségével a belső beszéd önállósul a külső beszédtől. Először belső multi-beszéddé válik, majd megkezdődik a beszédnek megfelelő operáció-sorrend felbomlása, az operációk sorrendje optimalizálódik. Ezzel a gondolkodás részben elveszti kezdeti beszéd jellegét, kezdeti szekvenciális, illetve multi-szekvenciális tulajdonságát. Az optimalizálódás során egyre nő az elemi aktiv-asszociációk száma. Ebben a pontban lényegében egy "interiorizációs" hipotézist, egy sejtést vázoltunk. Ezen az uton kivánjuk vizsgálni a gondolkodás kifejlődését a modellen, a korábban vázolt körülmények között.

Láthatjuk, hogy a nem pontos kifejezések és a hipotézisek száma egyre több és több. Tehát ennél a pontnál meg kell állnunk.

Ugy gondoljuk, hogy a modell alkalmas lesz később pszichofiziológiai és pedagógiai pszichológiai kutatásra.

Ha a modell hardwareként realizálódik, akkor mint egy számitógép modulja alkalmas lesz speciális folyamatfelismerésre és vezérlésre.

A hardwarenek sok értékes tulajdonsága van. Például bizonyos értelemben "tanulórendszer", "adaptiv rendszer", "alakfelismerő rendszer". Alkalmas lesz ilyen perifériának.

Egy modell nem jó "vagy" rossz ", csak "jobb " vagy "rosszabb " - mint egy másik

Az olvasó joggal kételkedik abban, hogy az agyban, vajon valóban végtelen és ilyen szépen hurokmentessé rendeződött neuronális hálózatot találhatnánk-e. Nem is állitunk ilyet. A bemutatott modell csak gyakorlóterülete a vizsgálatnak, a bizonyitásnak és a modellezéshez megfelelő szemlélet kialakitásának.

A második modell azonban már egy olyan tetszőleges irányitott gráfra fog épülni, melyben:

a. véges sok forrás lesz
b. az EUO halmaz elemei a gráf pontjaihoz tetszőlegesen lesznek hozzárendelve,

de az algoritmus csak az a. és b. pontoknak megfelelően lesz módositva.

Látható, hogy ez a modell már anatómiaibb megközelitést is megenged, hiszen a gráf és a hozzárendelés megadásával a neuronhálózatot szabadon választhatjuk meg.

Sejtjük, hogy az első modell segitségével kialakitott lényeges fogalmaink és bizonyitott tételeink analogonjait - bizonyos feltételek mellett - a második modellnél is megtaláljuk majd. Egyik feladatunk éppen ezen feltételek meghatározása lesz.

Azonban a második modell esetében is megmaradnak majd a modell olyan tulajdonságai, melyekben mint az agy tulajdonságaiban kételkednünk kell. Például ebben a modellben is igaz, hogy a vezérlést egymástól távoli nyomok együttesen határozzák meg, ami az agyban nehezen elképzelhető.

Tehát szükséges lesz egy olyan /harmadik/ modell vizsgálata is, melyben az egész gráfra, az egész modellre vonatkozó definiciókat, "lokális definiciók" fogják felváltani. Például a pozició fogalmát egy-egy pontra fogjuk értelmezni: A P_t/p/ pozició a p pont aktiv megelőzőinek halmaza lesz t-ben. Hasonlóan lokálisan fogjuk definiálni az algoritmust,továbbá a t-beli várakozás és vezérlés fogalmát is. A harmadik modell hasonlitani fog a McCulloch-Pitts féle neuronális hálózat modellhez. Éppen ezért érdemes megvizsgálni a leglényegesebb eltéréseket:

- Szembetünő különbség a nyomfogalom és a nyomok általi vezérlés. /A nyomhipotézis/

- Közismert, hogy az agy rendelkezik néhány hologramszerü tulajdonsággal.

Mint láttuk, ezeket már az első modell is részben tükrözi. Például egy kis gráfterület sérülése nem rontja lényegesen az adaptációt, illetve a nyomok módosulása részben azonos "gráfterületen" történik különböző folyamatok hatására, de ezek a folyamatok általában mégis felidézhetőek.

Vajon mi az oka a hologram, az agy és a modell közös tulajdonságainak? Feltehetően ugyanaz, nevezetesen: a holografikus kódolási mód és kód.

Bemutatásához és megértéséhez szükséges definiálni az első modellen néhány uj fogalmat a holográfiai szemléletmódnak megfelelően.

Mint láttuk, t-ben a P_t pozició és a pillanatnyi input határozzák meg az uj poziciót /P_{t+1} -t/. Nevezzük a P_t poziciót a t-beli szituáció referencia-kódjának és a pillanatnyi inputot a t-beli szituáció tárgykódjának. /Világos, hogy a t-beli referenciakód létrejöttében a t-1 ütembeli tárgykódnak is szerepe van. Igy a referencia kód időben változó /ellentétben a hologram referenciahullámával. A modellbeli referenciakód tehát általánosabb fogalom lesz./

Vizsgáljuk meg a valóságos hologram és az első modell közös tulajdonságait!

A valóságos hologram esetében a referencia- és a tárgyhullám "találkozik", interferál és változást idéz elő a fotolemezen. Egy későbbi időpontban az interferenciák nyomai és a referenciahullám rekonstruálja a tárgy képét. Mondhatjuk tehát, hogy a hologram kódja a tárgynak és a referenciahullám "cimzi" az interferencianyomokat, előidézve belőlük a tárgy képének rekonstruálását. Ha egyetlen lemezre különböző referenciahullámokkal több hologramot készitünk, akkor az előhivó referenciahullám határozza meg, hogy az interferencia nyomokból mely képek "idéződnek fel". A valóságos hologram

egy darabja is hologram, "minden" információt őriz.

Nevezzük a modell algoritmusa által előírt t-beli nyomváltozást és pontaktivizálást t-beli interferenciának. Igy a modell algoritmusa alapján mondhatjuk, hogy a referencia- és tárgykód interferenciát idéz elő. /Mindez csak játék a szavakkal, ha nem a mozgófilmet, hanem csak egyes kockáit látjuk magunk előtt!/ Mint láttuk, egy későbbi időpontban a nyomok és a referenciakód reprodukálják a tárgykódot, mondhatjuk tehát, hogy a nyomok "a tárgyak " /a szituációk/ "holografikus kódjai", és a referenciakód "cimzi" a nyomokat, előidézve azokból a tárgykódok reprodukálását. Mondhatjuk tehát, hogy a modellbeli automatikus cimzés "holografikus cimzés". STB.

Nem áll szándékunkban a holográfiai terminológia erőltetése. Az olvasóra bizzuk annak eldöntését, hogy vajon az előzőkben vázolt kódolási, dekódolási és cimzési módnak - az elnevezésén /"azonositóján"/ kivül - van-e valami köze a holográfiához.

Mindenesetre a "holografikus cimzés" jól magyarázza az agy és a modell közös hologramszerü tulajdonságait és megadja az ugynevezett "neuronális hologram" valószinü szerkezetét.

Ugy gondoljuk, hogy a "holografikus cimzés" érdemes részletesebb matematikai-számitástudományi vizsgálatra.

Felhasznált irodalom:

Arbib, M.A.: The Metaphorical Brain. Wiley-Interscience, 1972.

Neumann János: A számológép és az agy. Gondolat, 1964.

Pszichofiziológia. Szerk.: Ádám György: Gondolat, 1972.

Salamon Jenő: A Galperin-féle "értelmi cselekvés elmélete". Pedagógiai Szemle, 1966. 6. szám

Sebestyén Ferenc: Uj alapelv idegrendszeri modellekhez. Kandidátusi disszertáció, 1969.

Az ismertetett modell koncepciójának alapjait Pálvölgyi Lajos egyetemi hallgatóval közösen dolgoztuk ki.
Köszönettel tartozom Dávid Gábornak és Pálvölgyi Lajosnak a modell vizsgálatát lényegesen előrevivő számitástudományi, illetve pedagógiai-pszichológiai problémafelvetéseikért és Csáki Péternek a leirás pontatlanságait biráló értékes megjegyzéseiért.

FÜGGELÉK

A számitógépes tervezéssel kapcsolatos előadássorozat témaköréből csak két dolgozat érkezett be /azok is a kiadvány összeállitása után/. Ezek közül az átdolgozásra szükséges idő és a kötetben rendelkezésre álló hely hiánya miatt csak az egyik dolgozatot tudjuk közreadni. A Digitális Osztályon dolgozó szerzők, az Uzsoky Miklós vezetésével folyó több évi értékes munka eredményét jól áttekinthető tömör formában ismertetik.

NYOMTATOTT ÁRAMKÖRI KÁRTYÁK TERVEZÉSE SZÁMITÓGÉPPEL

Cseri Éva - Kerestély Domokos - Kovács Miklós
Máté Levente - Vincze Árpád

BEVEZETÉS

Az MTA-SZTAKI Digitális Osztályán kidolgozás alatt áll egy, a digitális berendezések számitógéppel segitett tervezésére szolgáló programrendszer [1, 2].

A tervezési feladat felosztásának megfelelően a programrendszer felbontása:

1. Logikai tervezést segitő programok
2. Konstrukciós tervezés: a/ kártyatervezés
   b/ keretkábelezés
3. Realizáló automaták vezérlő lyukszalagjait előállitó post-processzor programok.

Az egyes programrészek eredményei a következők adatainak képezik részét. Az adatkezelésre emiatt és az interaktiv tervezés lehetővé tétele érdekében is megfelelő adatstrukturát kellett választanunk. Az általunk kidolgozott adatstruktura egy listastruktura [3]. Ez az adatstruktura lehetővé teszi az adatközlés, a programok közötti adatátvitel és az interakciós pontokon való módositások egységes kezelését, valamint a különböző feladatokhoz tartozó adatok szimultán kezelését. Ebben az adatstrukturában lehetséges az adatok és az egyes információk közötti kapcsolatok ábrázolása is, oly módon, hogy az adatábrázolás alig redundáns és ugyanakkor az adatkezelés dinamikus.

Ez az ismertetés csak a nyomtatott áramköri kártyák tervezésére terjed ki [4].

NYOMTATOTT ÁRAMKÖRI KÁRTYÁKAT TERVEZŐ PROGRAM

A program célja, olyan nyomtatott áramköri lap konstrukciós tervezése, amely a kártyasikon mátrix elrendezésben tartalmaz dual-in-line integrált áramköri tokokat. A program által használt adatok három tipusba sorolhatók: logikai leirás, integrált áramkörök katalógusa és a konstrukciót és technológiát leiró adatok.

A program számos ponton eltér az eddig ismert kül- és belföldi hasonló célu erőfeszitések [5-10] elveitől és felhasznált algoritmusaitól. Legfontosabbnak tartjuk ezen eltérések közül az általunk használt késleltetett döntési stratégiát. Ezt a stratégiát, kissé tán komolytalanul, ugy fogalmazhatnánk meg, hogy "ne dönts el semmit csak, ha már a döntés nem halasztható". Az ismertebb rendszerek ezzel homlokegyenest ellentétes stratégiát követnek. A feladatokat mereven elkülönitik, és az aktuális feladatban a később jövő feladatok vélt egyszerüsitése érdekében elhamarkodott döntéseket hoznak. Az ilyen többfázisu programokra nézve az a jellemzés tipikus, hogy a program munkájának fele az előző fázis elhamarkodott döntéseinek kompenzálására irányuló, legtöbb esetben sikertelen erőfeszités.

Egy másik figyelemre méltó eltérés az, hogy a programot igyekeztünk maximálisan függetlenné tenni a konstrukciós és technológiai paraméterektől, szabványoktól. Ezeket, ha lehet, csak adatként kezeljük, s értéküket, sajátosságaikat nem épitettük be a programba.

A program felépitése lehetővé teszi különböző nagyságu mátrix elrendezésü kártyák tervezését, a feladat nagyságát csupán a rendelkezésre álló számitógép operativ memória terjedelme korlátozza. A megoldáshoz szükséges gépidő az elhelyezéstől eltekintve a realizálandó logikai rajz elemszámától lineárisan függ. Bár a program elve miatt nem igényli a tervező aktiv közremüködését a tervezés folyamatában, a programba beépitettünk néhány interakciós pontot: a logikai rajzot megvalósitó integrált áramköri készlet megváltoztatása, az elhelyezés módositása és elfajulások végrehajtása.

A felsorolt interakciós pontok a programot az alábbi részekre osztják: lefedés, elhelyezés, vezetékezés /1. ábra/.

A továbbiakban az egyes részfeladatokat tárgyaljuk, rámutatván a feladatok ujszerü felosztására s hagyományos rendszerekkel szembeni egyéb különbségekre.

I. A lefedés

a logikai rajz megvalósitásához szükséges minimális áru integrált áramköri modulkészletet határozza meg. A megoldás során a teljes IC katalógusból csak azokat az IC tipusokat tekinti elérhetőnek, amelyeket a tervező a feladat leirásában megnevezett. A megvalósitó készlet árát csökkentendő figyelembe veszi a logikai elemek elfajulási lehetőségeit, azaz azt a tulajdonságát, hogy bonyolultabb elem egyszerübb funkciót is képes ellátni.

Ellentétben a hagyományos módszerekkel, a lefedés csak egy - a logikai rajz megvalósitására alkalmas - IC-készletet határoz meg, de a logikai rajz elemei és ezen IC-készlet között semmiféle megfelelkezést sem hoz létre. A hozzárendelés bármely szintü eldöntése ezen korai fázisban elhamarkodott lenne, hiszen ezzel az elhelyezés és a vezetékezés eredményeit esetleg döntő módon meghatározó megszoritásokat vezetnénk be minden kézzel fogható előny elérése nélkül. A lefedés megoldásával a program eléri az első interakciós pontot, ahol a tervező

a/ jóváhagyja az eredményeket
b/ módositja a lefedő IC-készletet
c/ módositja a megvalósitandó kártya logikai rajzát.

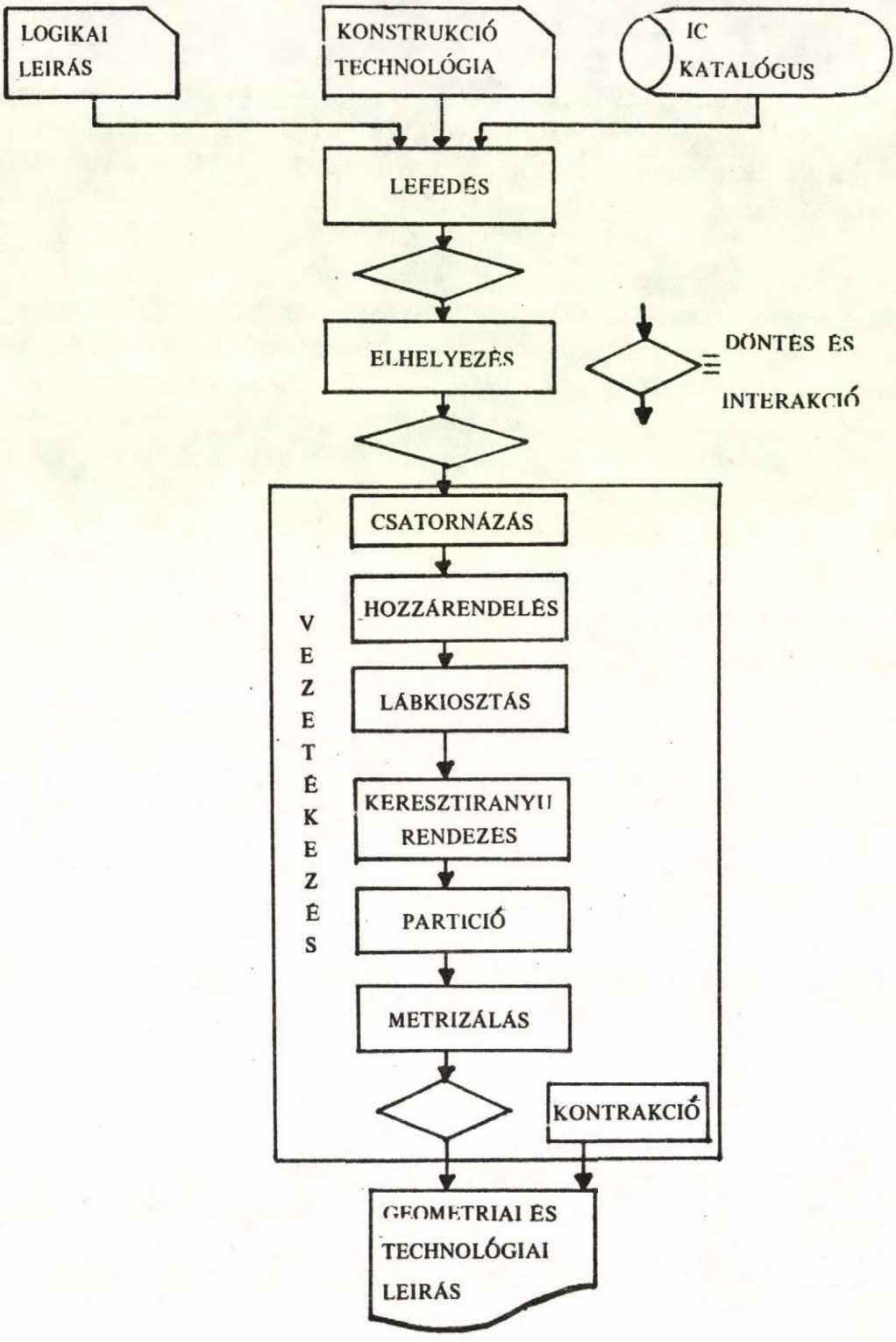
Ha a tervező a/ vagy b/-nek megfelelően dönt, a program futása az elhelyezésen folytatódik, mig c/ tipusu döntés esetén a lefedés ismételt futtatása szükséges.

II. Az elhelyezés [11-16]

feladata a logikai rajz elemeinek a tervezendő kártya modulhelyeire való leképzése. Ezt a feladatot - nem a végrehajtás, de a szemléletesség szempontjából - két részfeladatra oszthatjuk:

a/ a logikai elemek integrált áramkörökbe rendezése
b/ az integrált áramkörök helyének meghatározása.

Hangsulyozzuk azonban, hogy fenti két feladatot e program együttesen oldja meg, s munkája során mindvégig a logikai elemek /és nem az IC-k/ optimális helyének meghatározására törekszik. A megadott vagy pl. véletlenszerüen előállitott kezdeti elrendezést a program topológiai és geometriai összefüggéseket leiró célfüggvény szerint véletlen párcserékkel javitja.


LOGIKAI
LEIRÁS
KONSTRUKCIÓ
TECHNOLÓGIA
IC
KATALÓGUS
LEFEDÉS
ELHELYEZÉS
DÖNTÉS ÉS
INTERAKCIÓ
CSATORNÁZÁS
HOZZÁRENDELÉS
LÁBKIOSZTÁS
KERESZTIRANYU
RENDEZES
PARTICIÓ
METRIZÁLÁS
KONTRAKCIÓ
VEZETÉKEZÉS
GEOMETRIAI ÉS
TECHNOLÓGIAI
LEIRÁS

1. ábra

Az optimalizáló eljárás két logikai elemet jelöl ki, majd megvizsgálja, hogy az őket pillanatnyilag realizáló báziselemek kölcsönösen el tudják-e látni egymás funkcióját. Ha a két logikai elem az előbbi feltételnek eleget tesz, és a csere végrehajtása a célfüggvény értékét előnyösen változtatja, a két elem helyet cserél.

Ha az elempár nem helyettesithető egymással, akkor a program az aktuális modulok cseréjével próbálkozik, és hasonló elvek alapján dönt.

A sikertelen cserekisérleteket büntetjük, és ha a büntetések összege egy adott határt meghalad, akkor az eljárás befejeződik.

Az elhelyező program tehát meghatározza a felhasznált modulok helyét, az azokban realizált elemeket, de késleltetett döntési stratégiánknak megfelelően, a logikai elemek modulon belüli helyét még mindig nem határozza meg.

Az elhelyezés befejeztével a program kiirja, hogy az egyes logikai elemek a kártya mely helyén, milyen tipusu integrált áramkörben, milyen báziselemmel realizálódnak. Ha a realizáció elfajulás felhasználásával történt, a program közli, hogy az elfajuló báziselem mely lábai maradtak lekötetlenül. Ezen információk kiirásával a program eléri a második interakciós pontot. Itt a tervező

a/ az eredményeket jóváhagyja
b/ módositja az elhelyezést
c/ az elfajulások végrehajtásának módjára ad utasitást /a lekötetlen lábakat levegőben hagyni, tápfeszültségre, földre kötni, más lábbal összekötni/.

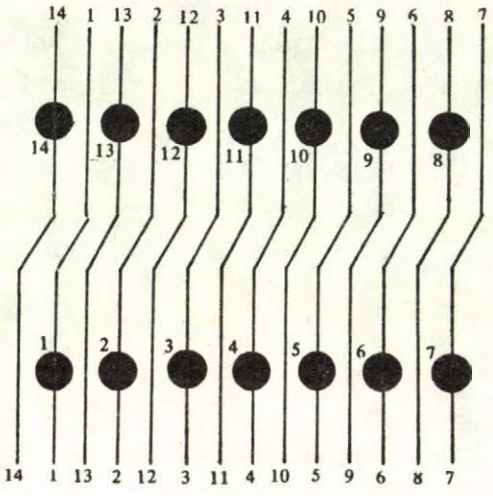
A tervező mindhárom döntése esetén a program először leadminisztrálja az interakciót, majd futása a vezetékezésen folytatódik.

III. A vezetékezés

határozza meg a tervezett kártya nyomtatási képét. Eredménye - egyben a kártyatervező program végeredménye - a nyomtatott kártya geometriai-technológiai leirása. A vezetékezést a hagyományostól gyökeresen eltérő módon oldja meg programrendszerünk, igy ezt a lépést mélyebben tagolva ismertetjük. A tervezés irányelvei e lépésre vonatkozóan a következők:

1. A tervezett kártyának két információs vezetékeket hordozó rétege van; az egyiken az x a másikon az y irányu vezetékszakaszok kerülnek realizálásra. Ugyanazon vezetéknek a két rétegen futó szakaszai között átfémezett furatok létesitenek elektromos kapcsolatot.

2. Az integrált áramkörök a kártyán sorokban és oszlopokban helyezkednek el. Az IC sorok és oszlopok közötti távolságok - "csatornaszélességek" - nem rögzitettek, hanem ezeket a vezetékezés helyszükséglete határozza meg.
3. Az integrált áramkörök lábai mindkét lábsor felől elérhetők - "in-line modell" - /2. ábra/.



2. ábra

A hagyományos eljárások általában a Lee-algoritmust [17-19] vagy a Müegér-algoritmust [20-21] használják a vezetékezés megoldására. Ezeknek alapvető hátránya, hogy

1. csak két pont összekötésére alkalmasak, holott az egy vezetékkel összekötendő pontok száma általában ennél több, igy a vezetékeket tervezés előtt valamely heurisztika szerint összekötendő pontpárok sorozatára kell bontani, s ez esetben fenti algoritmusok még saját célfüggvényük /minimális összhosszuság/ szerint sem képesek optimális megoldást adni az egyesitett vezetékképre.
2. Szekvenciális müködésüek, azaz a már megtervezett vezetékszakaszok alakja alapvetően meghatározza a tervezés alatt álló vezeték képét. Igy ugyanazon kártyát más vezeték sorrendben tervezve teljesen más eredményt nyerünk, s az optimális sorrend meghatározása gyakorlatilag lehetetlen.

A vezetékező program megirásánál célul tüztük ki, hogy leküzdjük a szekvenciális tervezés hibáit, feloldjuk a legrövidebb összhosszuságra való törekvésnek a lapközépen fellépő sürüsödéssel járó ellentmondásait és kihasználjuk a nem rögzitett csatornaszélességgel járó előnyöket.

a/ A csatornázás

meghatározza az ekvipotenciális pontokat összekötő vezetékek alakját a csatornák koordináta rendszerében. Az eljárás számára a kialakuló max. csatornakeresztmetszetek minimalizálása fontosabb szempont mint a "legrövidebb összhosszuságra" való törekvés. Bár a csatornázás nem a Lee-algoritmust használja a feladat megoldására, hanem tipusalakzatokból rakja össze a vezetékképet, mégsem teljesen mentes a szekvenciális tervezés hátrányaitól. Ezt azzal ellensulyozzuk, hogy az összes vezetékkép megtervezése után a leginkább tulterhelt csatorna keresztmetszeteket használó vezetékterveket eldobjuk, s ezeket a vezetékeket ujra tervezzük. Az igy kapott uj kártyakép általában egyenletesebb az eredetinél az éles maximumok kissé "szétkenődnek" és értékük csökken.

b/ A hozzárendelés

határozza meg, hogy az egyes integrált áramkörökön belül melyik báziselem melyik logikai elemet realizálja. Az egy IC-n belül realizálandó logikai elemeknek a csatornázás által meghatározott bekötésterve /hogy ti. mely lábat milyen irányból - balról, jobbról, alulról, felülről - érkező vezetékszakasszal kell bekötni/ ezen logikai elemek mindegyikére egy "orientációt" definiál. Az integrált áramkörök kivezetésrendszere a benne lévő báziselemekre* nézve hasonló tipusu orientáció értelmezését teszi lehetővé. E két orientációnak a fedésbe hozásával határozza meg az eljárás a vezetékezés szempontjából legkedvezőbb logikai elem - báziselem leképzést.

c/ A lábkiosztás

teszi teljessé a logikai elemek leképzését a realizáció elemeire azáltal, hogy a logikai elemlábak mindegyikének egy IC lábat feleltet meg. A megoldás triviális, ha az elem szerkezetéből következően ez a megfelelkezés eleve egyértelmü. Az u.n. ekvivalens lábcsoportok esetén /ahol tehát az elemnek több, funkcionálisan nem különböző lába van/ a lábkiosztás döntően befolyásolja a végső vezetékezési

*báziselemnek nevezzük - az ESzR terminológiával egybehangzóan - az IC legkisebb önálló funkciót megvalósitó elektronikusan elkülönithető részét.

terv jóságát. Körültekintő tervezés esetén ugyanis ezáltal lehetővé tudjuk tenni, hogy több vezetékszakasz ugyanazon vonalban haladjon, mig az erre nem ügyelő rendszerekben, ahol a lábkiosztást a tervezés első lépésében a hozzárendelésben eldöntik, ezt a lehetőséget csak sokkal ritkábban lehet kihasználni. A lábkiosztás megoldásához csatornánként előállitjuk az un. csatornagráfot, melynek csomópontjai a csatornázás által meghatározott vezetékszakaszokat, élei pedig egy általunk összeférhetőségnek nevezett hosszirányu rendezettséget reprezentálnak. Az i-edik csatornagráf p csucsából q csucsába akkor mutat él, ha p-től q növekvő koordináták irányában található, és p és q egy vonalban realizálható anélkül, hogy összeérnének. Ekkor p-t q-val összeférhetőnek mondjuk.

Az összeférhetőségek egy része a lábkiosztás előtt még nem eldöntött, s a lábkiosztás éppen annak érdekében dolgozik, hogy maximális számu összeférhetőséget hozzon létre. Az eljárás a csatornán belül vezetékszakasz-párokat vizsgál, s ha egy összeférhetőség létrehozásához csak egy lehetőség kinálkozik, az ebben szereplő lábakat leköti. Mivel akár egyetlen láblekötés elvégzése is egyértelmüvé tehet eddig el nem dönthető helyzeteket, az eljárást addig ismételjük, amig a fennmaradó el nem döntött helyzetek az összeférhetőség szempontjából közömbössé nem válnak. Ez után a még le nem kötött lábakat kézbekerülésük sorrendjében osztjuk ki. Az eljárás hatékonyságát illusztrálja, hogy gyakorlati példákon az egyenértékü lábakat mintegy 3/4-ének kiosztását az összeférhetőségek határozzák meg.

d/ A keresztirányu rendezés

az elemek fizikai összeköthetőségének megvalósitásához szükséges lépés. A csatornában lévő vezetékszakaszok között az összeférhetőség mellett még egy relációt definiálunk, amely az elemek keresztirányu rendezését irja le. Ez a keresztirányu rendezés tranzitiv irreflexiv és aszimmetrikus tulajdonságu.

A keresztirányu rendezések egyik csoportja a moduliránya csatornában keletkezik. Az egymással fizikailag szemben lévő lábak lekötései a csatornában lévő elemek között keresztirányu rendezést igényelnek. A rendezettség másik csoportját az egymásra merőleges csatornákban lévő összekötendő vezetékszakaszok összeférhetőségei hozzák létre.

A vezetékszakaszok közötti keresztirányu rendezést ismét a csatornagráffal ábrázoljuk. Ebben az irányitott gráfban azonban ciklusok keletkezhetnek. A ciklusok fizikailag meg nem valósitható rendezettséget jelentenek, ezért ezeket meg kell szüntetni. Bár találtunk egy

jó minimális index csomópont-kereső algoritmust, a gyakorlatban mégis azt az eljárást használjuk, hogy a ciklus megjelenése pillanatában felvágjuk a ciklust okozó elemet. Ez az elemfelvágás a valóságban azt jelenti, hogy uj vezetékszakaszt helyezünk el a csatornában, bizonyos modullábakat az eredetiről átkötünk ehhez, és gondoskodunk a két elem fizikai összeköttetéséről. Az uj helyzetben uj összeférhetőséget, uj sorrendet állapitunk meg, ha ismét ciklus lép fel, ujra elvégezzük az előző feladatokat.

A gyakorlatban ritkán keletkeznek ciklusok, többszörös ciklusok még ritkábban léphetnek fel, ezért jól bevált a fenti egyszerü ciklusfelvágási eljárás.

e/ A partició

a csatornában lévő vezetékszakaszokat rendezi el ugy, hogy azok minimális szélességü csatornát igényeljenek. A csatorna azon részét, amelyen egy vezeték helyezhető el utnak nevezzük, igy tehát az a feladat, hogy a csatorna minimális számu utat tartalmazzon. A feladatokat az előbb definiált gráfban ugy fogalmazhatjuk meg, hogy a csatornagráfot minimális számu olyan részgráfra /particióra/ kell bontani, amelynek elemei összeférhetőségi relációt tekintve teljes gráfot alkotnak, és a részgráfot egy pontnak tekintve fennáll az eredeti gráfon definiált keresztirányu rendezettség. Ez azt is jelenti, hogy egy keresztirányu rendezettségből legfeljebb egy elem kerülhet egy részgráfba. A feladatot egy olyan heurisztikus eljárással oldottuk meg, amely keresztirányu rendezettség hiánya esetén optimális, egyébként pedig ehhez közeli megoldást ad. Az eddigi példákon nem sikerült emberi beavatkozással javitani, a program által adott megoldást.

f/ A metrizálás

a nyomtatási terv elvi, még mindig csak topológiailag meghatározott képét geometriailag teszi a technológiai paraméterek figyelembevételével határozottá azáltal, hogy megállapitja az egyes rajzelemek koordinátáit. A kártya metrizálásával párhuzamosan hajtjuk végre a csatlakozók bekötését, és az utak csatornán belüli keresztirányu - a partició által még szabadon kapott - sorrendjében meghatározását. A feladat elvégzéséhez szükséges adat a kártya mérete, a külső csatlakozók száma /max. négy, minden oldalra legfeljebb egy/ és geometriai jellemzői.

Az utak sorrendjét a köztük lévő sorrendi reláció figyelembevételével ugy határozzuk meg, hogy a külső, csatlakozóval szomszédos csatornákban az un. rendezősávokban lévő vezetékszakaszok hosszának értéke várhatóan minimum legyen. Ezt azért hajtjuk végre, hogy a rendezősávokban a még tervezendő csatlakozó bekötő szakaszok összeférési esélyeit növeljük. Bekötjük a csatlakozókat, majd generáljuk a rendezősáv csatornagráfját oly módon, hogy a gráfban a lehető legtöbb összeférhetőséget hozzuk létre. Ezután elvégezzük a gráf particióját, és meghatározzuk az utak sorrendjét. Ezeket a müveleteket minden rendezősávra elvégezzük. Az eredmények birtokában elkészül a teljes nyomtatott kártya geometriai-technológiai leirása. Bizonyos esetekben előfordulhat, hogy a tervezés eredményeként kapott kártya fizikai mérete nagyobb mint a felhasználni kivánt alaplemez. Ebben az esetben a tervező az alábbi lehetőségek közül választhat:

a/ módositja a kapott nyomtatási képet
b/ uj tervezési paraméterekkel ujra inditja a programot
c/ hivja a kontrakció programot, amely megpróbálja a nyomtatási rajz méretét lecsökkenteni.

Mivel a program csak integrált áramköri elemeket vagy integrált áramköri modulban egyesitett más áramköri elemeket kezel, a tervező a metrizálási fázis után definiálhat különböző áramköri elemeket kötéseivel együtt a kártya szabad helyeire. A következőben ismertetésre kerülő kontrakció program már az igy megadott adatokat is kezeli.

g/ A kontrakció

a nyomtatott kártya geometriai-technológiai leirása alapján dolgozik. Egy lépésben egy csatornához tartozó vezetékszakaszokat kezel. A csatornában lévő vezetékszakaszokat a csatorna kis koordinátáju szélére tömöriti. A tömörités közben szükség van bizonyos vezetékszakaszok feldarabolására. Ha a kivánt tömöritést elértük a csatorna további vezetékszakaszokat és a csatornánál nagyobb koordinátaértéken lévő csatornákat a nyert távolsággal párhuzamosan eltolja.

A geometriai-technológiai leirás a kártyatervező program végeredménye. Ezt a realizációt végző NC gép vezérlő nyelvére a hozzátartozó post-processor program forditja le. Intézetünkben rendelkezésre áll egy olyan post-processor program, amely az ADMAP géphez készit vezérlőszalagot.

VÉGKÖVETKEZTETÉSEK

Nyomtatott áramköri kártyák számitógéppel segitett tervezésére történelmileg alakult ki a feladat elhelyezésére és huzalozásra való felosztása. Ezen felosztás okai részben a felhasználható számitógépek méreteiből, részben a két részfeladat matematikai megfogásának egyszerüségéből adódtak. Diszkrét, 2-3 kivezetésü alkatrészekből felépitett kártyák esetén a feladatok független kezelése - szakaszonkénti optimalizálás - elfogadható eredményeket adott.

Közismert az a tény, hogy a 60-as években a hardware fejlődése mellett a software messze lemaradt. Ennek egyik vetületeként a III. generáció nyomtatott áramköreit a II. generáció tervező eszközeivel tervezik. Legjobb esetben "megfejelték" fenti felosztást a hozzárendeléssel, ami visszavezette a feladatot az előzőre. Az igy kialakult rendszerek azonban igen mély ellentmondásokat tartalmaznak. Ezen ellentmondások alapja az, hogy a szakaszonkénti optimizálással nyert végeredmények távolról sem optimálisak az egész feladat szempontjából.

Jelen dolgozat - egy olyan rendszert ir le, amely a fenti merev feladat szegmentálást elveti, s egy a III. generáció eszközeit jobban figyelembe vevő, kevésbé merev tagolást alkalmaz. Az alkalmazott késleltetett döntési stratégia lehetővé teszi a részfeladatok kölcsönhatásainak széles körü figyelembevételét, s ezen keresztül az egész feladatra vonatkozó optimumkritériumok elérését.

A nyomtatott áramköri kártyatervező rendszert az MTA-SZTAKI Digitális Osztályán FORTRAN-nyelven realizáltuk, s jelenleg nagy számu kisérleti feladat futtatásával ellenőrizzük müködését.

IRODALOMJEGYZÉK

[1] M.Uzsoky, K.Pásztor, G.L.Kovács, L.L. Máté, M.Tarján: "Computer Aided Design of Digital Systems in Hungary", International Conference on Computer Aided Design, Southampton, 1972. IEE Conference Publication No. 86. pp. 359-365.

[2] Uzsoky M.: "A digitális berendezések tervezésének, konstrukciójának és gyártásellenőrzésének kérdései" előadás az MTA 1973. évi közgyülése keretében tartott Tudományos Tanácskozáson, Budapest 1973. május 8.

[3] Fidrich I.: "LIDI-72" előadás az MTA 1973. évi közgyülése keretében tartott Tudományos Tanácskozáson, Budapest, 1973. május 8.

[4] Kovács M., Vincze Á.: "Computer Aided Design of Printed Circuit Boards" Tudományos CAD szeminárium, Budapest 1973. április

[5] Shoichi Ninomiya et.al.: "Computer Aided Design for IC Computers" FUJITSU Scientific and Technical Journal, Vol. 5. No. 1. pp. 137-155.

[6] J.F.Jamison: "Programs Manual: UCARDS" Union Carbide Corporation, Nuclear Division Report No. CTC-4, 1969.

[7] J.F.Jamison: "User's Manual: UCARDS" Union Carbide Corporation, Nuclear Division Report No. K-1736 Revised, 1968.

[8] M.Breuer: "Recent Developments in the Automated Design of Digital Systems, Proc. IEEE Vol. 60. No. 1. January 1972.

[9] W.K.Orr: "Computer Aided Design for Custom Integrated Systems" FJCC, 1969, pp. 599-611.

[10] Állló G.: "Digitális áramkörök konstrukciós tervezése diszkrét elemek segitségével" előadás az MTA 1973. évi közgyülése keretében tartott Tudományos Tanácskozáson, Budapest 1973. május 8.

[11] L.Steinberg: "The Backboard Wiring Problem: A Placement Algorithm. SIAM Review Vol. 3, No. 1, Jan. 1961.

[12] J.Houghton: "A System for the Placement of Circuit Modules." International Conference on Computer. Aided Design 15-18, April, 1969, at the Univ. of Southampton.

[13] J.S.Mamelak: "The Placement of Computer Logic Modules" JACM Vol. 13, No. 4, pp. 615-629, October 1966.

[14] T.A.J.Nicholson: "Permutation Procedure for Minimising the Number of Crossings in a Network", Proc. IEE, Vol. 115. No.1, January 1968.

[15] P.C.Gilmore: "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem", J. Soc. Indust. App. Math. Vol. 10, No. 2, June 1962.

[16] A.Tiutin: "An Improved Algorithm for Component Disposition on a Board", Acta Frequenza, Vol. XXXIX, No. 5, Maggio 1970. pp. 417-421.

[17] C.Lee: "An Algorithm for Path Connections and its Applications", IEEE Transactions on Electronic Computer, Vol. EC-10, September 1961, pp. 346-365.

[18] S.Akers: "A Modification of Lee's Path Connection Algorithm", IEEE Transactions on Electronic Computers /Short Notes/, Vol. EC-16, February 1967, pp. 97-98.

[19] E.Majorani: "Simplification of Lee's Algorithm for Special Problems", Calcolo, Vol. 1. pp. 247-256, July 1964.

[20] E.Moore: "Shortest Path Through a Maze", Annals of the Computation Lab of Harvard Univ. Vol. 30, Cambridge: Harvard Univ. Press, 1959. pp. 285-292.

[21] I.Sutherland: "A Method for Solving Arbitrary-Wall Maze by Computer", IEEE Transactions on Computers, Vol. C-18, No. 12, December 1969, pp. 1092-1097.

A TANULMÁNYOK sorozatban eddig megjelentek:

1/1973 Pásztor Katalin: Módszerek Boole-függvények minimális vagy nem redundáns, { ∧ ∨ ¬ } vagy {NOR} vagy {NAND} bázisbeli, zárójeles vagy zárójel nélküli formuláinak előállitására

2/1973 **Вашкеви Иштван: Расчленение многосвязных промышленных процессов с помощью вычислительной машины**

3/1973 Ádám György: A számitógépipar helyzete 1972 második felében

4/1973 Bányász Csilla: Identification in the presence of drift

5/1973* Gyürki J.-Laufer J.-Girnt M.-Somló J.: Optimalizáló adaptiv szerszámgépirányitási rendszerek

6/1973 Szelke Erzsébet-Tóth Károly: Felhasználói Kézikönyv /USER MANUAL/ a Folytonos Rendszerek Szimulációjára készült ANDISIM programnyelvhez

7/1973 Legendi Tamás: A CHANGE nyelv/multiprocesszor

8/1973 Klafszky Emil: Geometriai programozás és néhány alkalmazása

9/1973 R.Narasimhan: Picture Processing Using Pax

10/1973 Dibuz Ágoston-Gáspár János-Várszegi Sándor: MANU-WRAP hátlaphuzalozó, MSI-TESTER integrált áramköröket mérő, TESTOMAT-C logikai hálózatokat vizsgáló berendezések ismertetése

11/1973 Matolcsi Tamás: Az optimum-számitás egy uj módszeréről

---

*-gal jelölt kivételével a TANULMÁNYOK megrendelhetők az Intézet Könyvtáránál /Budapest, I. Uri u. 49./