szükséges technikai feltételek és igények csak a jövőben fognak (ha fognak) jelentkezni. Éppen ezért jelenleg csak azok fogják igazán értékelni, akik elsősorban elméleti szempontból foglalkoznak vele. A jelenlegi változat nem végleges és valószínűleg nem is hibátlan, de mégis érdemes megismerni, mint az emberi agy egy kristálytiszta logikájú – és már önmagában ezért is – értékes termékét.

A szerzők a nyelvet arra szánták, hogy azon algoritmusokat lehessen közölni, számológéppel végrehajtani és tanítani. Ehhez a hármas célhoz egy negyedik kívánkozik – bár ezektől nem független – az ember és a számológép kapcsolatának új jelleget adni. Ezen utóbbi szempont nem lényegtelen, hiszen mióta gépeket és algoritmikus nyelveket használunk, ezt a célt minden programozási nyelv szükségképpen kitűzte magának. Ha a feladatot úgy fogalmazzuk meg, hogy a számológép minél több emberi (esetleg „ember feletti") tevékenységet végezzen minél egyszerűbb és kényelmesebb emberi beavatkozás segítségével, akkor az ALGOL 68 jelentőségét nagyra kell értékelnünk, mert olyan lehetőséget mutat meg, amellyel egy számológépből értelmes eszközt lehet létrehozni.

Irodalom

1. *A. van Wijngaarden* Draft Report on the Algorithmic Language ALGOL 68 (Matematisch Centrum, MR 93 Amsterdam 1968. január)
2. *A. van Wijngarden, B. J. Mailloux I. E. L. Peck, C. H. A. Koster:* Working Document on the Algorithmic Language ALGOL 68 Matematisch Centrum MR 95 Amsterdam 1968. július)

KALMÁR LÁSZLÓ

A programozási nyelvekkel kapcsolatos további teendők

Előadásom tárgya*: a programozási nyelvekkel – elsősorban az Esztergomi Nyári Egyetemen megismert modern programozási nyelvekkel – kapcsolatos további teendők. Nem beszélek itt azokról a nyilvánvaló teendőkről, mint az, hogy az itt hallottak és a kapott, valamint a jövőben még megküldendő írásos anyag alapján tanuljuk meg a PL/I, ALMO, ALGOL 68 nyelveket addig a szintig, hogy használni is tudjuk azokat, vagy mint az – és ez különösen az oktatásban dolgozók teendője, mint amilyen magam is vagyok – hogy másokat is megtanítsunk e nyelvekre. Nem beszélek részletesen azokról a teendőkről sem, amelyeket mások biztosan el fognak végezni helyettünk. Ilyen például az ALGOL 68 végleges (definiáló) anyagának elkészítése, amely az IFIP W. G. 2.1 (ALGOL) munkacsoportja négytagú albizottságának feladata, amely albizottság egyik tagját kivételes szerencsénk volt előadóként hallgatni a Nyári Egyetemen. Az ALGOL 68 – és a PL/I, esetleg az ALMO – szintaxisa és szematikája tanulási célra szolgáló, didaktikus megfogalmazásában már lehet szerepünk, hiszen Magyarország tudvalevőleg a matematika olyan kiváló pedagógusait adta a világnak, mint *Pólya György*, vagy *Dienes Zoltán*, hogy a fiatalabbakról ne is beszéljek.

Bizonyára meg fogják írni mások a PL/I és az ALGOL 68 nyelvekhez is azokat a kézikönyveket, amelyek a legfontosabb, elektronikus számológépen jól alkalmazható numerikus (és nem numerikus) algoritmusokat felírják e nyelveken, a publikáció előtt gépen kipróbált programok, illetve eljárás-deklarációk alakjában, mint amilyen több kötetes kézikönyv készül jelenleg mind a FORTRAN IV,[1] mind az ALGOL 60[2] nyelv számára. Az is lehet, hogy mire e sorozatok befejeződnek, a szerzők áttérnek modernebb programozási nyelvekre. A mi feladatunk ezzel kapcsolatban elsősorban az lesz, hogy e könyvekben található programokat – a nálunk használatos elektronikus számológépek, illetve fordítóprogramok által előírt korlátozások közé szorítva azokat – beiktassuk e gépek rutin-könyvtárába.

Ennek persze – és az említett algoritmusok publikáció előtti gépi kipróbálásának is – feltétele az említett nyelvek implementálása a megfelelő elektronikus számológépeken. A nagy teljesítményű gépeken más országokban bizonyára elvégezik ezt a munkát. De nálunk, legalábbis egyelőre még, sok kis- és középteljesítményű gép üzemel (és bizonyos célokra továbbra is üzemben lesz külföldön is). Ezekkel kapcsolatban főleg az a teendő vár ránk, hogy az e nyári egyetemen megtanult magas bonyolultsági fokú nyelvek olyan résznyelveit válasszuk ki, illetve definiáljuk, amelyek e kisebb gépeken is előnyösen implementálhatók, és amelyeken lehetőleg kevéssé nehezül meg a gyakorlatban előforduló típusú programok írása a teljes nyelvhez képest. Mind a PL/I, mind az ALGOL 60 esetében hivatalosan is létezik ilyen résznyelv, bár lehet, hogy egyes, nálunk használatos gépekhez még az is túlságosan bonyolult, és csak egy további résznyelvének implementálása látszik reálisnak. Természetesen e rész-

* Az esztergomi „Modern programozási nyelvek" című Nyári Egyetem záróelőadása

nyelvek implementálása is teendőink közé tartozik, lehetőleg jól megszervezett, országokon belüli és nemzetközi együttműködés előnyeinek felhasználásával. Az ALMO nyelvvel kapcsolatos megfelelő teendőket – az egyes modern programozási nyelvekről ALMO ra és az ALMO-ról az egyes elektronikus számológépek belső nyelvére fordítóprogramok megírása ALMO nyelven – *Ljubimszkij* professzor már említette előadásában.

A fordítóprogramokkal kapcsolatban még volna néhány megjegyzésem. Jelenleg olyan fordítóprogramokat szokás készíteni, amelyek gondos szintaktikus analízist végeznek fordítás előtt, és az ennek során kiderülő szintaktikus hibákról diagnosztikai üzenetet adnak. Ezt a programozók által elkövetett hibák nagy gyakorisága teszi szükségessé, ami az „errare humanum est" speciális esete; másrészt azonban nagyon megnyújtja a program első gépre vitele és tényleges, hasznos futtatása közben eltelt időt. A PL/I az alapértelmezésekkel és az ALGOL 68 az automatikus mód-konverziós (coercion) szabályokkal új utakat nyitott e tekintetben. Az utóbbiak józan mértéktartással hivatalosan is megengedik a programozó olyan eltéréseit a túlságosan is precíz szintaxistól, amelyek félreértés veszélye nélkül alkalmazhatók, viszont, főleg az előbbiek, felidézik azt a veszélyt, hogy tényleges hibák ellenére mást csinál a gép, mint amit a programozó vár tőle. Úgy gondolom, a jövő az olyan fordítóprogramoké, amelyek gyakorlatilag a legtöbb esetben automatikusan grammatikálissá teszik a szintaktikusan hibás programot is, de arról adnak üzenetet, hogy a program mely részén milyen korrekciókat végeztek, hogy ellenőrizni lehessen, e korrekciók megfelelnek-e a programozó intencióinak; csak ha ez nem áll, akkor kelljen a programozónak – e munkát megkönnyítő, a modern programozási nyelvek számára elkészítendő, hibajavító programok felhasználásával – kijavítania és újra gépre vinnie a programját. *Riesz Frigyes* professzor, mint az *Acta Scientiarum Mathematicarum* című szegedi folyóirat szerkesztője, annak idején úgy rendelkezett, hogy a szedőnek ki szabad javítania a kéziratban általa észrevett hibákat – nagyon jól képzett, a matematikai képletekben előforduló szintaktikus hibák iránt kifinomult érzékkel rendelkező szedője volt akkor a nyomdának, ahol a folyóirat készült – csak jeleznie kellett a kefelevonaton, hol, mit javított. Nos, hasonlót meg lehet engedni az elektronikus számológépnek is. Persze itt is mértéket kell tartani aközött, hogy mit engedünk meg hivatalosan is a programozónak, és mi számít szintaktikus hibának. Az ALGOL 68 mód-konverziói esetén például nem kell a gépnek korrekciós üzenetet adnia, ami túlságosan sok és felesleges ellenőrzési kötelezettséget róna a programozóra: vajon helyes-e a korrekció. A szintaktikus hibák viszont a gép által erről szóló üzenet adása kötelezettségével korrigálhatók.

Hadd említsek e tekintetben egy gyakran hallott mondást, amelyet akár a dialektikának a tagadás tagadására vonatkozó törvénye illusztrációjául is lehet használni. A kezdő programozó sokféle hibát követ el. A haladó programozó tudja, hogy e hibákat nem szabad elkövetnie és elkerüli azokat. A mesterprogramozó tudja, egy-egy ilyen „hiba" elkövetésekor mit csinál a gép, és szándékosan elköveti, ha éppen annak a „hibának" a következményére van szüksége. E mondás még a gépi kódban való programozás idejéből származik. Azóta nem azt kell néznie a mester-szintű programozónak, hogy a gép mit csinál, hanem azt, hogy a fordítóprogram hogyan működik adott esetben. Ha a gép diagnosztikai üzenet adása mellett mindig leáll, akkor megakadályozza a haladó programozót abban, hogy mester-szintre emelkedjék. Jobb hát, ha nem mindig áll le és „korrekciós üzenetét" csak a futtatás után kell ellenőrizni. Hogy mely esetben járjon el így, és mely esetben jelezzen szintaktikus hibát, az attól is függ, hogy milyen felkészültségű programozóra bízzuk a munkát.

Ezzel kapcsolatban jogos felvetni azt a kérdést, hogy miért csak az információátvitel területén használunk hibajavító kódokat, miért nem szerkesztünk automatikus hibajavítást lehetővé tevő programozási nyelveket is. A teendő itt az, hogy olyan szintaxissal lássuk el a nyelvet, hogy annyi és olyan hiba esetén, amelyeket az átlagos programozó óhatatlanul elkövet, — hacsak nincs szó durva hibákról — a fordítóprogram automatikusan ki tudja azokat javítani. Ehhez természetesen a programozási nyelvekben rejlő redundanciát minél nagyobb mértékben fel kell használni, de csak minimális mértékben szabad növelni ott, ahol ez nélkülözhetetlen, például numerikusan adott konstansok esetén.

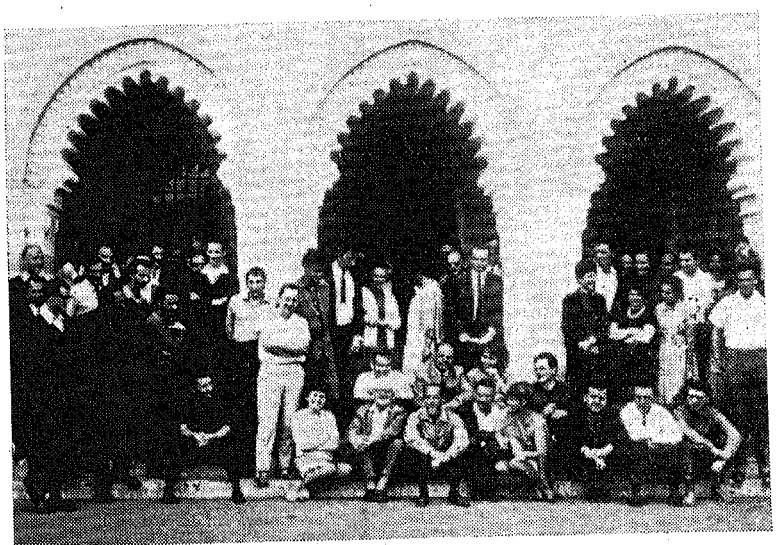
Ez a teendő már a programozási nyelvek továbbfejlesztéséhez tartozik. Minthogy a továbbfejlesztés lehetséges irányai szinte beláthatatlanok, és úgyis majd a programozási nyelvek alkalmazásának gyakorlata fogja megmutatni, hogy ezek közül melyek válnak aktuális teendőkké, nincs sok értelme tovább beszélni róluk. Csak azt a megdöbbentő tényt említem még meg, hogy a szaktudósok közül szinte a matematikus használ a legkevésbé elektronikus számológépet a maga elméleti kutatásaihoz, hacsak nem numerikus analízissel foglalkozik. Ez egyrészt azért van így, mert elterjedt tévhiedelem, hogy az elektronikus számológépek csak numerikus számításra valók, másrészt azért, mert nincs még jó, elméleti matematikai problémák gépi segítséggel való megoldására irányított, még a matematikus által is könnyen megtanulható programozási nyelv. Így a PL/I és az ALGOL 68 szerkesztői nem is vehették figyelembe az ilyen nyelvek tanulságait, mint ahogy például a COBOL tanulságait figyelembe vették. Hogy milyen segítségre számíthat reálisan a matematikus, azt – Hu Shih-hua pekingi professzor gondolatai felhasználásával – röviden vázolom. A matematikus meg akar oldani egy problémát, és ehhez van néhány ötlete, amelyeket közölne a géppel, hogy az ezeket kipróbálja és a részleteredményeket kinyomtassa. A matematikus ezeket megvizsgálja, kiválogatja közülük azokat, amelyek közelebb viszik a probléma megoldásához; a használható részeredmények ismeretében újabb ötletei támadnak, amelyeket újból közölne a géppel, s az előbbi folyamat ismétlődnék, amíg a probléma megoldását együttesen meg nem találják. Csak az a kérdés, milyen nyelven lehet matematikai ötleteket géppel közölni, amikor még arra sem ismerünk egyelőre más módot, hogy matematikus matematikussal közöljön matematikai ötletet, mint azt, hogy példán mutassa be az alkalmazását. Jelen esetben viszont épp a lehetséges alkalmazások megtalálása volna a gép feladata. (Az eddigi, matematikai tételek gépi bizonyítá-

sára irányuló algoritmikus és heurisztikus módszerek,[3] akármennyire érdekesek is, csak azt eredményezték, hogy a gép már ismert tételeket bizonyított be újból, esetleg új módon, holott alkalmazásának igazi értelme e téren az volna, ha gépi segítséggel olyan matematikai problémát is meg tudna oldani a matematikus, amit gép nélkül nem sikerült.

Megemlítem még azt is, hogy az első algoritmikus programozási nyelvek azzal a célkitűzéssel jöttek létre, hogy az emberi nyelvhez a belső gépi kódnál közelebb álló nyelvet konstruáljunk. Az újabban keletkezett programozási nyelvek azonban mind nagyobb mértékben eltérnek nemcsak az emberi nyelvektől, de hovatovább már a józan emberi gondolkodástól is. Egyelőre még áthidalható az eltérés, hiszen az ember nagyon tanulékony, alkalmazkodásra képes. De ha így megy tovább, mind nehezebb lesz áthidalni. További teendő tehát a programozási nyelvek fejlesztése terén: ismét közelebb hozni azokat az emberi nyelvhez. Olyan nyelvre gondolok, amelyen ilyenféleképpen lehet fogalmazni egy programot: „Legyen a valós x és y változó kezdőértéke .5 illetve .75 és definiáljuk a *funct* nevű eljárást tetszőleges valós u és v, valamint egész n paraméterek esetén a következőképpen: ... Hajtsuk végre a következő eljárást ismételten, $m =$ 1-től kezdve, 2-es lépésekben, addig, míg ez, és ez a feltétel nem teljesül: ... stb." És a diagnosztikai üzenetek is így szóljanak: „Elfelejtett w-nek értéket adni; mi az értéke? Elfelejtette a *regr* nevű eljárást definiálni; hogyan definiálja?" És ha a programozó hasonló stílusban adott válasszal pótolja a hiányzó értékadást, deklarációt stb., akkor a gép lefuttatja a programot. A felhasznált emberi nyelv némi szabványosítása árán ez az egyelőre utópiának látszó terv megvalósíthatóvá válhat anélkül, hogy a fordítóprogramnak a gépi fordításhoz szükséges természetes nyelvi szintaktikus analízishez hasonló bonyolultságú feladatot kellene végeznie.

Az előadó *Ljubimszkij* professzortól megkérdezték, hogy készül-e az ALMO-val izomorf belső nyelvű számológép, mint ahogy például a B 5000 a FORTRAN-nal, a kijevi MIR és még inkább a készülő Ukraina lényegében az ALGOL 60-nal izomorf belső nyelvű gép, és a University of Massachusetts (Amherst, Mass., U.S.A.) Computer Science Programja keretében, *Dr. J. A. N. Lee* vezetésével készül egy FORTPRAN–WATFOR-ral izomorf belső nyelvű gép. (Ilyen „programozási nyelv által vezérelt" elektronikus számológép gondolata, tudtommal, először *F. L. Bauer* és *K. Samelson* angol, francia és olasz szabadalmában szerepel, amelyben a verem-memória (push down store) műszaki megvalósítását szabadalmaztatták évekkel az erről – mint a szekvenciális formulafordítás software-módszeréről – szóló publikációjuk[4] előtt; ilyen géppel foglalkozik *W. Kämmerer* jenai habilitációs dolgozata is.[5] Talán az első tényleges megvalósításig *Z. Pawlak* lengyel mérnök jutott el,[6] igaz, lassú, dobmemóriás gép alakjában, amelyben ráadásul egy ciklus során többször is kellett a dobhoz fordulni.) Itt nincs idő arra, hogy részletesen kifejtsem az ilyen gépek előnyeit és azt, hogy ezek persze sokkal rugalmasabbak, mint egy szokásos elektronikus számológép, software helyett hardwarejébe fixen behuzalozott fordítóprogrammal, mert ezeknek belső nyelvére – amely már eleve tud annyit, mint egy algoritmikus programozási nyelv – mint alapra sokkal többet tudó software-rendszert lehet építeni. E programozás – és fordítóprogram – nélkül is működő gépekkel kapcsolatban valószínűleg csak a jövő fogja megmutatni, mi mindent lehet velük csinálni, ha ellátják őket programokkal is. Véleményem szerint az ALMO-n kívül a PL/I-gyel és az ALGOL 68-cal izomorf belső nyelvű gép szerkesztése is a jövendő feladatai közé tartozik.

Szeretném még megemlíteni, hogy elméleti teendők is vannak e téren. Például az ALGOL 68 szintaxisának leírásmódja egy lényeges újítást tartalmaz: e leírás kétlépcsős, amennyiben egy metanyelv generálja a tulajdonképpeni (alap-) nyelv végtelen sok szintaktikus szabályát. Az ilyen kétlépcsős grammatikák elméleti, matematikai nyelvészeti vizsgálata feltétlenül érdekes teendő, amelynek végrehajtását *Péter Rózsa* professzor már elkezdte. Péter Rózsa a kétlépcsős (szövegösszefüggéstől független) grammatikát általánosan, mint öt olyan véges, közös elem nélküli,



Az esztergomi nyári egyetem résztvevői.
Az első sorban középen Kalmár László és I. E. L. Peck professzorok (Peck professzor felvétele)

a vesszőt és a kettőspontot nem tartalmazó Z, M, P, V és K adathalmaz által meghatározott absztrakt fogalmat (a matematikában elterjedt kifejezéssel: $\langle Z, M, P, V, K \rangle$ rendezett ötöst definiálja, ahol Z a jelek halmaza, M a metajelek halmaza, P a metaprodukciók, V a primitív előprodukciók és K a kategórianevek halmaza. Minden metaprodukciónak $m:l$ alakúnak kell lennie, ahol m metajel, l pedig jelek és/vagy metajelek egymás után írással jelölt lánca véges sorozata); minden primitív előprodukció $l:L$ alakú, ahol l olyan „vegyeslánc", mint előbb, és L ilyen vegyesláncok (vesszőkkel elválasztott) listája. Bármely metaprodukcióban a kettőspont utáni vegyesláncot a kettőspont előtt álló metajel közvetlen kifejtésének nevezzük. Egy metajel kifejtésein közvetlen kifejtéseit, továbbá azokat a vegyesláncokat értjük, amelyeket valamely már megkapott kifejtésében előforduló metajelnek valamely közvetlen kifejtésével való pótlásával kapunk. Valamely metajel értékeinek azokat a kifejtéseit nevezzük, amelyekben nem fordul elő egy metajel sem, csak jelek. A primitív előprodukciókból valamely bennük előforduló metajelet tetszőleges, de mindenütt, ahol előfordul, ugyanazon értékével pótolva további előprodukciókat kapunk; ezek közül azok, amelyekben már nem szerepel egy metajel sem, a produkciók. A kategórianeveknek (a K halmaz elemeinek) mind elő kell fordulniok valamely produkcióban a kettőspont előtt. Bármely produkcióban a kettőspont utáni listát, amely már „jelláncok", azaz kizárólag jelekből álló láncok listája, a kettőspont előtti jellánc közvetlen kifejtésének nevezzük. Ezekből kiindulva a többi kifejtéseit úgy kapjuk, hogy valamely már megkapott kifejtésében szereplő olyan jelláncot, amely előfordul valamely közvetlen produkcióban a kettőspont előtt, valamely közvetlen kifejtésével pótoljuk. Valamely kategórianév azon kifejtéseit nevezzük terminálisoknak, amelyek csupa olyan jellánc, ún. terminális fogalom listái, amely nem fordul elő egyetlen produkcióban sem a kettőspont előtt. Minden kategórianevet a terminális kifejtései halmaza (kategóriája) nevének tekintünk. Például ha „program" egy kategórianév, akkor a terminális kifejtései alkotják a programok kategóriáját. A kétlépcsős grammatika olyan értelemben generál egy nyelvet, hogy minden kategórianévhez hozzárendeli a megfelelő kategóriát.

Péter Rózsa eredményei közül megemlítem azt, hogy példát adott olyan kétlépcsős grammatikával generálható nyelvre, amelyben a terminális fogalmak halmaza primitív rekurzív, de amely nem generálható egylépcsős szövegösszefüggéstől független grammatikával. Viszont megmutatta, hogy ha csak véges számú terminális fogalom van egy kétlépcsős grammatikával generálható nyelvben, akkor az elvileg generálható egylépcsős grammatikával is (ha esetleg sokkal bonyolultabban is). Egyszerű ellenpéldával azonban azt is megmutatta, hogy az utóbbi tétel már véges számú terminális fogalom esetén sem áll, ha a metaprodukciók jobb oldalán – az ALGOL 68 szintaxisánál alkalmazott eljárástól eltérően – megengedünk vesszőket is, hasonló használattal, mint a produkciók jobb oldalán.

Úgy gondolom, érdekes teendő e vizsgálatok folytatása, a kétlépcsős nyelvek elméletének további kifejtése, végtelen sok terminális fogalom esetén. Talán még a természetes nyelvek adekvát generálásának kérdését is közelebb hozza a gyakorlati megvalósításhoz a generálás ezen új módszere. Esetleg a három- és többlépcsős nyelvgenerálásban rejlő lehetőségeket is érdemes volna megvizsgálni. *Peck* professzor már említette, hogy egyes metafogalmak végtelen hosszú (valójában transzfinit, sőt, nem is a szokásos lineáris értelemben transzfinit) terminális produkciói – ha ilyeneket, a Péter Rózsa vizsgálataiban használt definíciókkal ellentétben megengedünk – szintén vetnek fel majd problémákat a logikusnak. Úgy gondolom, az ALGOL 68 szemantikájának a munkaokmányban alkalmazott definiálásmódja is érdemes elméleti vizsgálatokra, talán még a természetes nyelvek exakt szemantikai elméletéhez is adhat gondolatokat az ilyen vizsgálat.

Irodalom

[1] *Donald E. Knuth:* The Art of Computer Programming. Addison-Wesley. Eddig megjelent az 1. kötete: Fundamental Algorithms. Reading, Mass., 1968. 643 old.

[2] *F. L. Bauer, A. S. Householder, F. W. J. Olver, H. Rutishauser, K. Samelson, E. Stiefel:* Handbook of Automatic Computation. Springer-Verlag. Eddig két kötete jelent meg: Volume I, Part a, *H. Rutishauser:* Description of ALGOL 60. Berlin, Heidelberg, New York, 1967. 323 old. és Part b, *A. A. Grau, U. Hill, H. Langmaack:* Translation of ALGOL 60. Berlin, Heidelberg, New York, 1967. 397 old.

[3] Lásd például *A. Newell, H. A. Simon:* The logic theory machine. IRE Transactions on Information Theory. 1956. évi IT-2. sz. 61–79. old.; *H. Gelernter:* Realisation of a geometry theorem proving machine. Proceedings of the International Conference on Information Processing. Paris, 1959. 265–273. old.

[4] *K. Samelson, F. L. Bauer:* Sequentielle Formelübersetzung. Elektronische Rechenanlagen. 1959. évi 1. kötet. 176–182. old.

[5] *W. Kämmerer:* Ziffern-Rechenautomat mit Programmierung nach mathematischem Formelbild. Jenaer Jahrbuch. 1959. évi II. sz. 396–440. old.

[6] *Z. Pawlak:* Organisation of adress-free computer B–100. Bulletin de l'Académie Polonaise des Sciences, Série des Sciences techniques. 1961. évi 9. kötet. 229–234. old.

[7] *Péter Rózsa:* Zur zweistufigen Satzstruktur-Grammatik II. Studia Scientiarum Mathematicarum. Megjelenés alatt. (Az I. rész előzetes közlemény volt.)

Az ICL kutatásai a mágneses regisztrálás terén

A 68-as IFIP kiállítás alkalmával az ICL pavilonban bemutatják a Stevenage-i Kutató- és Fejlesztési Laboratóriumban kifejlesztett nagy sűrűségű mágneses adatrögzítési eljárásokat.

A Laboratóriumban kidolgozott program célja olyan új regisztrálási eljárások kifejlesztése, amelyekkel megközelítőleg tízszeresére növelhető a jelrögzítés sűrűsége, s egyidejűleg hasonló mértékben fokozható az adatátvitel sebessége is. A jelenleg alkalmazott módszerekkel 1000 bit/sec átviteli sebesség érhető el.

Előreláthatólag egy éven belül igazolódik az új technikai megoldások gyakorlati alkalmazhatósága, és akkor kezdetét veheti az első adattár prototípusának tervezése és kifejlesztése.

Bár a kutatómunka eredményei akármilyen számítógépes adathordozó közeg esetében hasznosíthatók, elsősorban a rögzített vagy cserélhető mágneslemezes tárolóegységeknél kerülnek majd felhasználásra. Az új eljárásokat az adatrögzítés sűrűségét meghatározó alábbi tényezők figyelembevételével alakították ki:

1. a regisztráló fej résmérete,
2. a mágneses regisztráló felület minősége,
3. a mágnesfej, és a mágneses közeg közötti távolság (koercitív erő).

ICL Sajtószolgálat
1968/július 29.