Programozás-módszertani vizsgálatok a SZÁMKI-ban

Havass Miklós

A tanulmány összefoglalja a számítógép segítségével történő feladatmegoldás különböző közelítési módjait. Felvázolja a közelítések közös kutatási területeit, és ezekre hivatkozva bemutatja a SZÁMKI programozáselméleti, programozás-módszertani munkájának eredményeit. Ezután felvázolja a kutatómunka további céljait, amelyek összekapcsolódnak az SZKCP-nek a nagyüzemi software-gyártás megteremtésével kapcsolatos törekvéseivel.

(Érkezett: 1978. január 3.)

Bevezetés

A programozás – általánosan értelmezve – a számítógéppel megoldandó feladat megfogalmazásától a feladat megoldását szolgáltató „kód" lefuttatásáig terjed. E tevékenységsorozat lényegében nem más, mint sajátos problémamegoldó folyamat, ahol szerepet kap az intuíció, a tervezés, a heurisztika stb., egyszóval a problémamegoldás alapkategóriái. A számítógépek történetének elmúlt évtizedei során a programozók rájöttek arra, hogy e folyamat egyes tevékenységei már elég jól megfogalmazottakká váltak ahhoz, hogy automatizálni, gépesíteni lehessen azokat, azaz e tevékenységeket programokkal lehessen helyettesíteni. E téren a legjelentősebb eredményeknek az assemblerek, fordítóprogramok, operációs rendszerek megjelenését tekinthetjük. Az automatizálás eredményeként azt nyertük, hogy bonyolultabb, nagyobb feladatokat tudunk megoldani kevesebb hibával és nagyobb hatékonysággal (tehát pl. rövidebb idő alatt).

A nagyobb lehetőségek azonban még bonyolultabb feladatok megoldásának igényét vetették fel. Továbbá bebizonyosodott az is, hogy az egyszer már megoldott feladatok minőségileg magasabb szinten történő újra- és újrafogalmazása és megoldása elkerülhetetlen a számítógépek eredményes alkalmazásához, vagyis a megoldandó feladatok nagyságuknál vagy időben változó, megújuló voltuknál fogva bonyolult, összefüggő feladatrendszereknek tekinthetők. Ilyen feladatrendszerek programozásához az automatizálás során eddig elért eredmények korántsem tekinthetők elegendőnek, s ma a számítógépek „jó" alkalmazásának legfőbb gátja feladatmegoldó képességünk elégtelen volta, szemben a korábbi idők hardware-korlátaival (ebben az értelemben software-krízisről is szokás beszélni).

A kérdés megoldása, a feladatmegoldási, programozási módszerek javítása ma a számítástechnika legsürgősebb teendői közé tartozik. A próbálkozások egyik ága a jelenlegi programkészítési folyamat egyes lépéseinek tisztázására, javítására (pl. strukturált programozás, defenzív programozás, programozói csoportok munkaszervezése stb.), esetleg automatizálására (pl. tesztkiértékelő rendszerek) irányul, a másik ága pedig a feladatmegoldás más útjait próbálja bevonni a programozás körébe (pl. a mesterséges intelligencia módszereinek alkalmazása).

A SZÁMKI, mint a számítógépek alkalmazási problémáit kutató intézet, fennállása óta kiemelt figyelmet fordít a feladatmegoldás automatizálásának kérdéseire, hiszen a jövő gépei kielégítő alkalmazásának alapvető fontosságú kérdése a feladatok adekvát megoldását lehetővé tevő programrendszerek, ill. számítórendszerek kidolgozása. Jelen tanulmányunkban az utóbbi évek kutatómunkájának főbb eredményeit mutatjuk be, és a jövendő legfontosabb céljait fogalmazzuk meg.

A számítógépes feladatmegoldás modellje

A számítógépeket feladatok megoldására szoktuk felhasználni. A feladat megoldását részben, vagy egészben végzi a számítógép, de a megoldásban részt vesz az ember is. *Számítórendszeren* valamilyen feladat megoldásán együtt dolgozó számítógép, programok és emberek összességét értjük.

Problémamegoldó rendszeren olyan számítórendszert értünk, amely egy inputként kapott feladatra eredményként a feladat megoldását adja interaktív vagy automatikus üzemmódban. A problémamegoldó rendszer előállíthatja közbülső lépésként a feladat megoldását nyújtó programot, de adhatja közvetlenül is a megoldást.

A problémamegoldó rendszer intputja a feladatnak a problémamegoldó rendszer által ismert fogalmakkal (objektumokkal, valamint közöttük értelmezhető relációkkal és műveletekkel) történő leírása, *specifikációja.* Egy bonyolult feladat specifikálása nehéz szellemi munkát igényel, ugyanakkor a problémamegoldás kulcsfon-

tosságú részének tekinthetjük. A feladat specifikálása könnyebben elvégezhető akkor, ha nem egy lépésben kell végrehajtani, hanem mód van a feladat fogalmainak szukcesszív közelítésére, lebontására – ami Sokratesig nyomon követhető gondolkodási forma [4].

Egy problémamegoldó rendszer *strukturált*, ha módot nyújt az általa megoldandó feladat több szintű, hierarchikus specifikálására. Ez azt jelenti, hogy a feladat specifikációját minden egyes szinten olyan *primitív fogalmak* segítségével adjuk meg, amelyek definiálására más szinteken kerül sor. A primitív fogalmak „adat" vagy „művelet" típusúak lehetnek.

Az egyes szinteken definiálásra kerülő fogalmak meghatározása kétféle módon történhet:

a) *algoritmikus-megadással* amikor azt adjuk meg, hogy az újonnan definiált fogalom milyen, más szinteken meghatározandó fogalmak által meghatározott lépésekből építhető fel;

b) *követelmény-megadással* amikor azt adjuk meg, hogy az újonnan definiált fogalom milyen feltételeknek kell, hogy eleget tegyen, amely feltételek megfogalmazásánál felhasználhatunk más szinteken specifikálandó fogalmakat is.

A problémamegoldó rendszer módot ad

- *hagyományos* (strukturált) *programozásra*, ha az egyes szinteken csak az algoritmikus megadással specifikálhatjuk a fogalmakat;
- *verifikált programozásra*, ha az egyes szinteken mindkét specifikációt megadjuk, s valamilyen módon (manuálisan vagy automatikusan) bebizonyítjuk ezek ekvivalenciáját;
- *automatikus (vagy félautomatikus) programozásra*, ha az egyes szinteken a követelmények megadásából automatikusan (vagy interaktív módon) állítjuk elő az algoritmikus definíciókat;
- *autonóm problémamegoldásra*, ha a követelmények interpretálása segítségével, az algoritmikus definíciók explicit előállítása nélkül oldjuk meg a feladatot.

A programozás fent felvázolt rétegei nagyjából feladatmegoldási képességeink színvonalát is jelentik. Természetesen a magasabb szintű problémamegoldás sokkal igényesebb számítórendszert tételez fel, mint az alacsonyabb szintű. Emiatt – bár perspektívikusan feltétlenül domináns szerepet kell játszania az automatikus, ill. autonóm programozásnak – a közeljövőben még számolnunk kell a hagyományos programozással, ill. annak továbbfejlesztett, *racionalizált*, azaz egyes lépéseiben „gépesített" variánsaival. Így a programozás módszertani kutatásoknak egyszerre kell foglalkozniok valamennyi réteggel. Ezt szerencsére megkönnyíti az a tény, hogy a több fajta közelítésnek, mint problémamegoldási folyamatnak, tekintélyes átfedése van.

Vegyük számba, melyek a leglényegesebb (többségükben közös) kérdések.

1. A feladatok specifikálásához (akár az algoritmusok leírására, akár a követelmények definiálására) *nyelvekre* van szükség. Miután a strukturált problémamegoldó rendszerekben a specifikáció több szinten történik, ahol az egyes szinteken megjelenő fogalmakat az alacsonyabb szintű elemi fogalmakból kell felépíteni, nem is egyes nyelvekre van szükség, hanem nyelvek hierarchiájára. Az ilyen hierarchiának alsó fokain álló nyelvek (alacsony szintű, gépikód nyelvek) fogalmai a számítógép fogalmaihoz állnak közel, míg a magasabb szinten álló nyelvek (magas szintű, igen magas szintű nyelvek) a megoldandó feladat sajátos fogalmaihoz közelednek. A legmagasabb szintű nyelvek a természetes nyelvek családjába tartoznak.

   Minél általánosabb, magasabb szintű nyelven fogalmazzuk meg problémánkat, annál kevésbé tudjuk kihasználni a számítógép sajátos struktúráinak előnyeit, így annál inkább szükség van a feladat közvetlen megoldását szolgáltató specifikáció vagy program *optimalizálására*.

   Amikor azonban a nyelvek magasabb szintjei egyre inkább megadják a felhasználó számára a feladat leírására szolgáló nyelv géptől való függetlenségét, a problémamegoldó rendszereket előállítók számára előtérbe kerül a gépek közötti *áthelyezhetőség* problémája.

   Az optimalizálás és az áthelyezhetőség egymásnak ellentmondó követelményeit a *kódgenerálás* közben kell (illetőleg lehet) megoldani.

   Miután a hierarchikus feladat specifikációit szolgáló nyelvek lehetőség szerint a feladat fogalmi rendszeréhez közel álló fogalmakkal dolgoznak, a nagyszámú „célnyelv"' gyors implementációjához szükség van a *fordítóprogramok készítésének gyorsítására*, módszereinek fejlesztésére (fordító-fordító programok, ill. rendszerek).

2. A hierarchikus feladat-specifikálás fő célja a feladat leírásához használt fogalmak, döntések időbeli szétlagolása, elkülönítése. A feladat megoldása céljából azonban szükséges a különböző szinteken megjelenő, de összetartozó (esetleg nagyon sok) fogalmak egyidőben történő kezelése. A problémamegoldó rendszerek lényeges kérdése tehát a problémamegoldás során használt fogalmi rendszerek reprezentálása, tárolása, előkeresése és változtathatósága: a *tudásreprezentáció*. A tudásreprezentáció kérdése az automatikus és autonóm programozásnál egészen új, bonyolult problémákat vet fel.

3. Az egyes szinteken megjelenő fogalmak leírásához, levezetéséhez olyan *dedukciós eljárásokra* van szükség, amelyek
   - a hagyományos programozás esetén előírják az egyes szinteken megjelenő, algoritmikusan specifikált fogalmak dekomponálásának strukturális, funkcionális szempontjait, a dekomponálás módszereit, valamint a dekomponálást végző emberi kiscsoportok viselkedési módját;
   - a vertifikált programozás esetén eljuttatnak a kétféle feladat-specifikáció ekvivalenciájának bizonyításához, továbbá
   - a feladatnak a követelményekkel együtt történő leírása esetén – a specifikáció által leírt állítások követelményeinek generálásán keresztül – a feladatmegoldás előállításához.

   Az első esetben ezek az eljárások informálisak, az utóbbiakban formálisak.

4. A feladatmegoldás során az eredmények előállításán kívül meg kell győződni az eredmények helyességéről. A helyesség vizsgálatának különböző szintjei lehetnek.

A *validitás-vizsgálat* a specifikáció által leírt feladat és a valóságos, megoldani kívánt feladat szembesítését végzi el. Bár a legsúlyosabb (legnehezebben kimutatható, leghosszabb következmény-sorozattal járó) hibákat a specifikáció megadása közben követjük el, sajnos a validitás-vizsgálat nehezen formalizálható eljárás, s ma még elég kevés eredményt tudunk e téren felmutatni.

A *program-verifikálás* a követelményekkel és algoritmikusan megadott feladat-specifikációk ekvivalenciájának bizonyításával az eredmény specifikációknak való megfeleltetését igazolja.

A *programtesztelés* a feladat-specifikációt statisztikusan veti egybe a feladat megoldásaival.

A programhelyesség vizsgálatának két utóbbi módja strukturált problémamegoldó rendszerek esetén szintenként is elvégezhető, ami nagymértékben egyszerűsíti az amúgy igen bonyolult eljárásokat.

5. A számítógépes feladatmegoldás minden szinten feltételezi a számítógép használatát. E kapcsolatban várhatóan sokáig fennmarad az interaktivitás, az ember-gép párbeszéd. A hagyományos programozásnál például ez a feladatmegoldás jellegéből következik (a specifikációnak algoritmikus leírássá való transzformálása mindig az ember munkáját tételezi fel).

   Az automatikus programozásnál a felhasznált módszerek terjedelmessége (kombinatórikai robbanás!) teszi szükségessé a próbálkozásoknak irányt mutató, intuitív programozói beavatkozást. Ebből a tényből következik, hogy mindenféle rendszernél szükség van a programozási munka *adatfeldolgozási* tevékenységének ellátásához szükséges komponensekre (pl. programverzió-kezelés, dokumentáció-előállítás stb.).

Eredmények

Intézetünk programozás-módszertani kutató tevékenységének irányait, aktuális feladatait döntő mértékben az Intézet általános feladatai, ezen belül az Intézet software fejlesztő aktivitása határozta meg. Ez elsősorban a hazai gyártású kisszámítógépek alap- és alkalmazási softwarerel való ellátására, vállalati irányítási rendszerek kidolgozására és újabban államigazgatási információs rendszerek létrehozására irányult. E munkáink közben merült fel számos olyan kérdés, amelynek megoldása módszertani kutatásokat igényelt és indított meg.

A programozási nyelvek területén kezdetben a fő kérdés ismert programozási nyelvek *fordítóprogramjainak* kisgépekre való implementálása volt. A fordítóprogramok kidolgozásának középpontjában a szintaktikus elemzés áll. E területen végzett munkánk eredményeképpen dolgoztunk ki egy olyan algoritmust, amely környezetfüggetlen nyelvtanokat határozott levezető elemzésre alkalmas formába hoz [19].

A fordítóprogramokkal kapcsolatosan vizsgáltuk a tárkezelés problémáit is az ALGOL–68 adattípusainak figyelembevételével [36].

Miután a fordítóprogramokat sok gépre kellett kidolgozni (csak a hazai gyártású gépeket említve: EMG 830, VT 1010/B, R–10, R–5, PRACTICOMP 4000), kerestük a fordítóprogramok írásának olyan általános lehetőségeit, amelyek függetlenítik a fenti programozási munkát a konkrét gépektől. Ezen erőfeszítések eredményeként dolgoztuk ki az UTRA (Universal Translator) rendszert, amelynek alapgondolata a fordítóprogramok géptől függő és független részének szétválasztása [9]. A rendszer géptől függő bázisa egy bármely gépre könnyen megírható értelmező program. Az UTRÁ-t több alkalommal használtuk fel sikerrel (pl. COBOL fordító az 1010/B-re, FORTRAN fordító PRACTICOMP-ra stb.).

Ugyancsak e problémakörben való tapasztalatgyűjtés érdekében implementáltuk az ALMO (Algoritmicseszkij masinnoorientyirovannüj jázük) nyelvet az R–10-en. Ezt a rendszert azonban érdemben nem használtuk fel [13].

A fordítás géptől való függetlenné tételének következő lépéseként a CDL-et (Compiler Description Language) implementáltuk több gépre [21], és segítségével több programozási nyelv fordítóprogramját hoztuk létre (pl. BCPL, COBOL, PASCAL, CDL, APTRA, DIL). A tapasztalatok mind a géptől való függetlenséget, mind a programok hatékonyságát illetően kielégítőek [7], [8], [20], [22].

E nyelvet sikerrel alkalmaztuk fordítóprogramoktól eltérő *rendszerprogramok* implementálására is, elsősorban kisszámítógépekre. Így pl. ezen a nyelven írtuk meg az R–5 operációs rendszerét [31].

A nyelvnek e téren való alkalmazhatósága elsősorban annak köszönhető, hogy a nyelv

- nyílt,
- lokális változói stack szervezésűek – azaz reentrant programrészek írására alkalmas,
- jól szervezett vezérlési struktúrákkal rendelkezik,
- a makroutasítások helyettesítésével a lefordított program memóriaszükséglete redukálható.

Az utóbbi problémát illetően kísérletet végeztünk, amely során CDL-ben beprogramoztuk az R–10 monitorának egyes moduljait, s a programok méretét összehasonlítottuk az assembly-ben megírt változatok méretével. A különbségek nem voltak szignifikánsak. 100 assembly utasításnál kisebb méretű modulok esetén valamivel az assembly, ennél nagyobb modulok esetén valamivel a CDL változat volt rövidebb.

Az R–5 operációs rendszer kidolgozása közben új eljárást adtunk a „holtpont-probléma" megoldására, kis memóriák esetén [32].

A rendszerprogramozási nyelvek használatára vonatkozóan több egyéb eredményes kísérletünk is volt (pl. SIMULA, LP15, Konkurrens PASCAL, BCPL). Érdekessége kedvéért megemlítjük, hogy külföldi megrendelésre még COBOL nyelven is írtunk fordítóprogramot [35].

Rendszerprogramok magas szintű nyelven való írásával kapcsolatos törekvéseink között megemlítjük az intézetünk egyik munkatársa által kifejlesztett PL/M nyelvet, amely az R–10 IDOS rendszerére épül fel [26]. A nyelv típus nélküli változókon dolgozik, blokk-struktúrája nincs. Vezérlési struktúráit tekintve közel áll a strukturált programozás alapstruktúráihoz. A szekció és a modul fogalmát ismeri. Az R–10-en kívül implementálták és felhasználták a VT–50 és az INTELL 8008 gépeken is.

Az algoritmikus programozási nyelveken kívül kísérleteztünk a nem-algoritmikus nyelvek területén is.

A PROLOG nyelv implementálásán túlmenően gyakorlati eredményeket hozott az MM (Management Modul)

rendszer, és a központjában levő „AP nyelv", amelynek az R–10-re és a nagyobb ESZR gépekre készült implementációját több helyen alkalmazzák [28].
Az MM rendszer alapgondolata: a konkrét feldolgozási programok előállítása az adatfeldolgozási folyamatban tipikusnak nevezhető tevékenységeket megvalósító típusprogramokból és a feladatot nem procedurálisan leíró paraméterekből. A NIM IGÜSZI által végzett összehasonlító számítások szerint egy közepes méretű adatfeldolgozási feladatot harmad annyi idő alatt készítettek el, mint PL/I-ben.*
Ugyancsak adatfeldolgozási feladatok programozására dolgoztuk ki és implementáltuk a FORS rendszert [17], amely döntési táblázat formájában megadott bemenő nyelven RPG-re történő fordítás segítségével tudja előállítani a tipikus adatfeldolgozási programokat.
A korábban már említett CDL nyelvet használtuk fel azokban a kutatásainkban, amelyeknek célja a megoldandó problémák absztrakt megfogalmazása és az absztrakt leírás implementálása volt. A feladatok absztrakt leírására a VDL-et (Vienna Definition Language) használtuk fel. A VDL-hez hasonló vezérlési struktúrái miatt jól tudtuk alkalmazni a CDL–et a probléma implementálásához. E módszer segítségével fordítóprogramokat hoztunk létre (PASCAL, BCPL). A VDL-ben leírt absztrakt fordítóprogramokat kézzel fordítottuk át (nem nagy munka!) CDL-re, míg az elemzést és a kódgenerátort egyenesen CDL-ben írtuk [22], [24].
A VDL-et mint szemantika leíró nyelvet kiterjedten vizsgáltuk, és számos nyelvet írtunk le benne: a már említett PASCAL, ill. BCPL-en kívül az APL-et [23], BASIC-et [38] és McCarthy egy egyszerű nyelvének fordítóprogramját is.
A szemantika-leírással kapcsolatos kutatásaink eredményeként kidolgoztuk a (gépfüggő tulajdonságokkal is rendelkező!) programozási nyelvek szemantikájának korrekt, modellelméleti leírási módját [14].
Ennek során olyan alapvető fogalmakat írtunk le matematikai logikai eszközökkel, mint amilyen a program, a program futása, utasítások jelentése stb.
E munka folytatásaként jelenleg a programozáselmélet kategóriaelméleti formalizálásával foglalkozunk.

Az adatábrázolás, tudás-reprezentáció területén számottevő kutatási eredményt nem értünk el; e téren éppen jelenleg indulnak vizsgálatok.

Sokat foglalkoztunk *a feladat dekomponálásának* elméleti és gyakorlati kérdéseivel. SAM (Structured Abstract Model) kutatásainkban [11] a
– VDL szemantika-leírási módszereit,
– Dijkstra strukturált programozással kapcsolatos elveit,
– Hoare axiomatikus programhelyesség-bizonyítási módszerét próbáltuk összeötvözni abból a célból, hogy egy olyan módszert kapjunk, amely alkalmas hierarchikusan rendezett feladat-családok implementáció- és gép-független leírására. A módszer mind tervezési eljárásként, mind software-elemek enciklopédiaszerű, általános szinten történő leírására felhasználható. A módszer segítségével leírtunk néhány software-elemet, így többek között egy absztrakt assembler modellt [10], makro assemblerek családját [3], egy nyomozó rendszert [12], egy file kezelő rendszert [34], s ezek segítségével vizsgáltuk a leíró nyelvvel szemben támasztott követelményeket mind a hierarchikusan kibomló algoritmusok leírása, mind a szintenként történő helyesség-bizonyításhoz felhasználható feltételek leírása szempontjából.
A SAM modellekkel kapcsolatban terveztük meg és első változatában implementáltuk a VERGEN-t (Verification Condition Generator) [30], amely segítségével a programtervezés egyes szintjei helyességének (manuális) belátásához felhasználható verifikálási feltételeket lehet előállítani. E rendszer segítségével bizonyítottuk be a Hoare által leírt FIND program SAM leírásának helyességét [29].
A dekomponálás gyakorlati kérdéseit illetően számos módszertani kísérletet folytattunk, konkrét munkáinkkal összekapcsolva. Így gyakorlati munkaszervezési, programozás-módszertani eredményeink vannak a moduláris programozás [18], a strukturált programozás [4], a különféle programozói szervezetek munkája (pl. chief programmer's team) [2], a programozási munkák irányítása, ellenőrzése [15] stb. területén.
E kísérletek azt mutatják, hogy jóllehet egy-egy módszertani fogásnak önmagában véve kimutatható előnyei vannak egy-egy minta projekt keretein belül, jelentős eredményt akkor lehet elérni e téren, ha e módszereket – megfelelő eszközökkel párosítva, a hazai gazdasági-szociológiai helyzetet figyelembe véve – a teljes programozási folyamat egészére, integráltan alkalmazzuk.

A programok helyességének belátása alapvető fontosságú programozási tevékenységünkben, mivel kifejlesztett programjaink döntő része nem belső felhasználásra készül, hanem legtöbbször olyan felhasználókhoz kerül, akikkel már nincsenek is közvetlen kapcsolataink. Helyesen működő programok előállítására nincsenek abszolút érvényű módszerek, így munkánkban több oldalról próbáljuk közelíteni a problémát.
Mindenekelőtt létrehoztunk egy külön csoportot, amelynek feladata az intézet egyéb részlegeinél készülő programoknak tesztelése, minőségi felülvizsgálása, (elsősorban használói szemléletű!) értékelése. E kísérleti jellegű tevékenység elsősorban a bonyolultabb rendszerprogramok bevizsgálását tűzte ki célul és végezte el. Számos fordítóprogramot, adatkezelő rendszert, operációs rendszert vizsgáltunk meg ilyen módon, s a munka eredményeként kialakult egy általános programkiértékelési metodika [36]. Jelenleg foglalkozunk eredményeinek felhasználói programokra – így elsősorban ügyviteli feladatok programjaira – történő adaptálásával.
E csoport foglalkozik olyan programozási módszerek, programeszközök vizsgálatával és létrehozásával is, amelyek elősegítik a programok alapos tesztelését. Így készítettünk többféle tesztadat-generátort (különféle eloszlású és típusú tesztesetek előállítására), modulok tesztelését elősegítő – paraméterek segítségével illeszthető – tesztágyakat [33], a legkritikusabb termékeink részére érvényesítő rendszereket (különös gonddal válogatott, nagyszámú tesztfeladat együttese) stb.
Az ilyen típusú tesztelési eszközök a programok helyességének ellenőrzése mellett általában jó alapot szolgáltatnak az elkészült programok hangolására, optimalizá-

*ESZR Klub (szervező: NOTO OSZV) 1977. dec. 12-i ülésén elhangzott szóbeli közlés.

lására is. Ilyen irányú munkáink egy része real-time rendszerek működését megfigyelő, statisztikai adatokat gyűjtő és elemző módszerek kidolgozása, amelyeket részben ilyen rendszereink fejlesztéséhez, hangolásához [27], részben pedig meglévő operációs rendszerek működésének elemzésében [1] használtunk fel. Az ilyen irányú munkáink másik részének célja fordítóprogramok optimalizálása, különös tekintettel arra a törekvésünkre, hogy igyekszünk magas szintű nyelveket használni a fordítóprogramok létrehozásához. [8]-ban beszámolunk egy kísérletről, amelyet COBOL fordítóprogramokkal kapcsolatban végeztünk. Párhuzamosan készítettük el egy COBOL fordítóprogram assembly és CDL változatát, ügyelve arra, hogy ugyanaz legyen a forrás, a tárgyszöveg, s a fordításhoz is ugyanazokat a táblázatokat használjuk. Az összehasonlítások azt mutatták, hogy egyenlő memóriafoglalású programokat kaptunk, a CDL esetében 30–40%-nyi munkaidő-megtakarítással, amely CDL program azonban 25–30%-kal alacsonyabb hatásfokkal dolgozott. Ezt a változatot a fenti eszközökkel kimérve, majd felgyorsítva az assembly-vel egyenértékű sebességű változatot kaptunk.

A helyes működésű programok fejlesztésével kapcsolatos kutatásaink másik ága olyan fordítóprogramok létrehozására irányul, amelyek a szintaktikus hibák szempontjából önjavító tulajdonságokkal rendelkeznek. E tárgyban kidolgoztuk a szintaktikus elemzésnek egy olyan módszerét, amely képes bizonyos típusú szintaktikai hibák kijavítására [5].

Végül foglalkozunk programok helyességbizonyításának elméleti kérdéseivel is. A VERGEN-ről már korábban említést tettünk. Az ugyancsak korábban már leírt modellelméleti modellünk alapján megvizsgáltuk a Manna és Floyd által bevezetett programhelyesség-bizonyítási eljárások alapvető tulajdonságait [14].

A programozási munka adminisztratív részét megkönnyítendő számos apróbb-nagyobb programot, ill. módszert dolgoztunk ki. Így foglalkoztunk a dokumentáció gépesítésével, a programok archiválásával, a határidők számítógépes ellenőrzésével, a „szabványos kész modulok" könyvtárának kérdésével stb. Lokális eredményeken túlmenően itt is az a tapasztalatunk, hogy reálisan, nagyüzemileg használható eredményeket csak akkor kapunk, ha ezeket az eszközöket egységes folyamatban integráljuk.

Kutatómunkánk folytatásának fő irányai

Módszertani vizsgálataink, amelyeket több irányban folytattunk, számos eredményt hoztak. A jövőt illetően legfontosabb feladatunknak azt tartjuk, hogy ezeket az eredményeket integráljuk, s felhasználásukkal olyan egységes programozói környezetet alakítsunk ki, amelyben a javasolt programozási módszereket megfelelő programeszközök támogatják, s amely környezet lefedi a programkészítés teljes folyamatát. Véleményem szerint ezeknek a módszereknek és eszközöknek a zöme a hagyományos programozásban is ismert, de csak mint részeredményeket, egyedileg használják fel őket, egymáshoz való kapcsolódás nélkül, anélkül, hogy teljes „technológiai sort" alkotnának.

Az integrálás céljából két fő kutatási területen kívánunk tovább dolgozni.

Kutatásokat és fejlesztéseket végzünk olyan interaktív programozó környezet létrehozására, amely segítségével bonyolult programok (alap-, ill. cél-software) dolgozhatók ki. Néhány ilyen integrált környezet az irodalomból ismert (pl. CADES, SEF, OPUS, REVS stb.).

Mi – sajátos feladatainkat figyelembe véve – azt tűzzük ki, hogy rendszerünk

- segítse elő a magas szintű nyelven történő, interaktív programozást,
- oldja meg a különböző gépeken is használható, hordozható software készítésének feladatát,
- vegye le a programozók válláról a programozási munka adminisztratív tevékenységeit,
- segítse elő a problémák megtervezésének strukturált megközelítését.

Egy ilyen rendszer első kísérleteként készítjük el az ANSWER rendszert [6]. A rendszer – amelyet ESZR gépekre fejlesztünk, de a hordozhatóság próbájaként más gépekre is át kívánjuk írni – magja a CDL nyelv továbbfejlesztését jelentő CDL2, amely új jellemvonásai tisztább lehetőségeket nyújtanak a programok világos szerkezetének kidolgozására. A nyelv köré integránsan épül a „szemantikus" szövegszerkesztő és a programrészek kipróbálására szolgáló próbapad [25]. A rendszer egy információs bázis kezelésére alkalmas alrendszerrel és egy PROLOG-szerű nyelvvel – mint a tervezés eszközével – egészül ki.

Fejlesztő munkát végzünk a vállalati irányítási rendszerek (VIR) számítógéppel segített tervezésével és létrehozásával kapcsolatban is. Programozás-módszertani elképzeléseink magját itt egy olyan programrendszer kifejlesztése jelenti, amely az információs rendszerek tervezését gépi úton segíti elő, s amely egy programgenerátorba torkollik, ami – megfelelő környezet felhasználásával – a programok gépi előállítását teszi lehetővé. Első változatként itt a FORS, ill. MM rendszereink továbbvitelét tűztük ki célul, a szükséges programozási módszertanok kidolgozása mellett.

E két fő kutatási irány egybeesik a Számítástechnikai Központi Célprogram CF–31-es célfeladatának célkitűzéseivel [16].

E vizsgálatainkon túlmutatóan, de szorosan hozzájuk kapcsolódva kívánjuk folytatni munkánkat az autonóm problémamegoldás alapjainak kidolgozása terén. Itt a megalapozó matematikai, valamint a tudásreprezentációs, dedukciós stb. alkalmazott kutatásokon túlmenően, célul tűzzük ki egy-két speciális terület – pl. a fent említett VIR-ek – feladatait megoldó autonóm rendszerek konstruálását is. Bár meggyőződésünk az, hogy e rendszerek igazi felhasználása már a jövő „új-architekturájú" gépein történik, a kísérleti realizálását már a jelenlegi gépeken meg akarjuk kezdeni.

Mindezen kutatásaink során nemcsak saját erőinkre hagyatkozunk, hanem be kívánjuk vonni e munkába mindazon intézményeket és szakembereket, akik hivatásuknak érzik a számítógépes feladatmegoldás sikerén való munkálkodást. Ezt az együttműködést szándékaink szerint részben a közös, koordinált kutató-fejlesztő munka, részben egymás eredményeinek felhasználása kell hogy jelentse.

Irodalom

[1] ADAMY L.: Logikai rendszerterv a „Közép-kategóriájú R-gépek általános rendszerelemző, hatékonyságvizsgáló és elszámolási programrendszerére" OS operációs rendszer alatt. (SZÁMKI 1941/77)

[2] ARNOLD L.-né–ESZTERGÁR ZS.: Egy munkaszervezési kísérlet tapasztalatai. Programozási Rendszerek'75. Szeged. p. 127–132.

[3] ASZALÓS J.: Makroassemblerek strukturált absztrakt modellcsaládja. SAM–III. 3. kötet. (INFELOR 1555/75)

[4] ASZALÓS J.: A strukturált programozás irodalmának áttekintése. SAM–IV. 2. kötet. (SZÁMKI 1654/76)

[5] BAKOS T.: Automatic Error Correction in Formal Languages. 2nd Hungarian Computer Science Conference. Bp. 1977. p. 144–158.

[6] BEDŐ Á.: Az ANSWER rendszer általános leírása. SOFTTECH D2. 1976.

[7] BÁRÁNY S.: A PASCAL nyelv utasításainak fordítása. Szakdolgozat. Bp. 1975. (INFELOR 1477/75)

[8] BOLGÁR G.: On the Speed Measurement and Optimization of a Compiler Written in High-Level Language (SZÁMKI 1758/77)

[9] DÖMÖLKI B.: A Universal Compiler System Based on Production Rules. BIT 1968. 4. szám, p. 262–275.

[10] DÖMÖLKI B.: On the Formal Definition of Assembly Languages. Symposium and Summer School on the Mathematical Foundations of Computer Science. High-Tatras, Czechoslovakia, 1973. szept. p. 27–39.

[11] DÖMÖLKI B.–SÁNTÁNÉ TÓTH E.: Software komponensek formális leírása Strukturált Absztrakt Modellek segítségével. Információ-Elektronika, 1977. 4. szám, p. 203–211.

[12] FARKAS ZS.: Strukturált absztrakt nyomkövető rendszer modellje. SAM–III. 4. kötet. (INFELOR 1557/76).

[13] FOLTÉNYI V.: Az R10 ALMO fordítóprogramja. Programozási Rendszerek '75. Szeged, p. 306–315.

[14] GERGELY T.–SZŰTS M.: Programozási nyelvek szemantikájának matematikai logikai megalapozása. (SZÁMKI 1933/77)

[15] HAVASS M.: Software-fejlesztési munkák vezetése. (INFELOR 1219/1974)

[16] HAVASS M.: A nagyüzemi software-gyártás és az automatizált rendszertervezés és szervezés módszereinek és eszközeinek kutatás-fejlesztése és létrehozása. SOFTTECH D1. 1977.

[17] KARLI GY.: Általános adatfeldolgozási programgenerátor. Információ-Elektronika, 1975. 5. szám, p. 209–214.

[18] KISDI G.: Moduláris programozás. INFELOR tanulmány. 1974.

[19] KOMOR T.–VU–LUC: Határozott levezető elemzés. Programozási Rendszerek '75. Szeged, p. 248–255.

[20] KOMOR T.: A DIL programgeneráló rendszer. Információ-Elektronika, 1977. 6. szám, p. 337–341.

[21] LABORCZI Z.–LANGER T.: A compiler-compiler on the R10 minicomputer. Minicomputer Forum, 1975.

[22] LANGER T.–BEDŐ Á.: BCPL fordítóprogram – egy módszer próbája. Programozási Rendszerek'75. Szeged, p. 89–105.

[23] LANGER T.: Az APL formális leírása a bécsi definíciós nyelv segítségével. (INFELOR 1080/1972)

[24] LANGER T.: Strukturált Absztrakt Fordítóprogram mint verifikált fordítóprogramok tervezésének eszköze. SAM–I. 2. fejezet. (INFELOR 1368/74)

[25] LANGER T.: Az ANSWER operációs rendszer CDL2 nyelvi rendszere. Információ-Elektronika, 1978. 1. szám, p. 24–28.

[26] MANDLER GY.: Interactive Disc Operating System, IDOS. PLM leírás. 1977. VII. 21. (Kézirat)

[27] SÁGHY A. (szerk.): Operációs rendszerek és fordítóprogramok minőségvizsgálata. SZÁMKI Közlemények (megjelenés alatt).

[28] SIKLAKY I. és munkaközössége: Management Modul rendszer. Közgazdasági és Jogi Könyvkiadó. Bp. 1976.

[29] SIKLÓSI I.: VDL-programok verifikálása. SAM–I. 3. fejezet. (INFELOR 1368/1974)

[30] SIKLÓSI I.: Verifikálási feltétel generáló VERGEN program leírása. SAM–III. 2. kötet (INFELOR 1564/76)

[31] SOMOGYI J.: Minicomputer Software Design and Implementation Based on the Use of a System Programming Language. Minicomputer Software. North-Holland, 1976, p. 31–37.

[32] SOMOGYI J.: Deadlock Problems of Dynamic Memory Allocation on Minicomputers with Multilevel Interrupt System. Acta Cybernetica, 1977. 2. kötet, p. 91–98.

[33] SOÓS K.–VÁRKONYI ZS.: Az R10 univerzális tesztágygenerátora. Programozási Rendszerek'75. Szeged. p. 166–181.

[34] SZENDI G.: Általános file-kezelő rendszer Stukturált Absztrakt Modellje. SAM–IV. 1. kötet. (SZÁMKI 1635/76)

[35] SZENTES R.: A DAL assembly nyelv és fordítóprogramjának megvalósítása magasszintű programozási nyelven. Programozási Rendszerek'75. Szeged. p. 274–287.

[36] SZŐKE P.: Some Remarks on New Instances and Garbage Collection. ALGOL–68 Conference. Glasgow. 1977. márc.

[37] VÁRKONYI ZS.: A programtermékek ellenőrzése. SZÁMKI Közlemények. 15. szám. 1976.

[38] VERBOVSZKI L.: Új módszer programozási nyelvek szemantikájának leírására, és ennek alkalmazása a BASIC esetére. Szakdolgozat. Budapest. 1976. (SZÁMKI 1592/76)

Резюме

В статье обощаются различные методы решения задоставляются результаты работы теории и способов дач с применением вычислительных машин и препрограммирования. После этого дается схематичное изображение дальнейших целей исследовательской работы, которые включаются в Центральную Целовую Программу Вычислительной Техники при стремлении создания массового производства системы математического обеспечения.

Summary

The different ways of computer aided problem solving are summarized and the result of programming theory and methodology studies carried out at SZÁMKI (Research Institute for Applied Computer Sciences) are described. After this, the further objectives of research activity are outlined in accordance with the SZKCP's (Central Development Project in Computing) effort to establish large-scale software production.

SZÁMKI KÖZLEMÉNYEK

Szilárdi Ferenc

Elektronikus adatfeldolgozás egységes dokumentációs rendszere

A dokumentációs rendszerjavaslat kidolgozását az tette indokolttá, hogy hazánkban az adatfeldolgozás mind szélesebb körű elterjedése ellenére is a legkülönbözőbb összetételű és mélységű dokumentációt készítik. Míg az egyes programrendszereket vagy feldolgozásokat kizárólag a kidolgozásnál használták fel és módosították, ez nem is okozott különösebb problémát. A központosított programnyilvántartás bevezetése, továbbá a központi alapokból finanszírozott adatfeldolgozási software fejlesztése, majd e termékek szélesebb körű forgalmazása már hangsúlyozottan vetette fel a dokumentációk formai és tartalmi egységesítését. A tanulmány éppen az egységesítés minimális formai előírásait foglalja össze javaslat formájában.