**Prolog, IQSYS, deklaratív programozás – Interjú Szeredi Péterrel**

**Szeredi Péter matematikusként végzett az ELTE-n, 1972-ben. 1975-ben, másodikként a világon készített Prolog interpretert. Később az MProlog-on dolgozott sokáig, majd a nyolcvanas évek második felében a Manchesteri és a Bristoli Egyetemen folytatta kutatásait. Hazatérése óta, a nemrég az IQSYS-be integrálódott IQSOFT-nál dolgozik, kutatás-fejlesztési projekteket irányít. A BME-n deklaratív programozást tanít. Számos szakmai elismerés után, 2000-ben tudományos diákköri munkájáért kapott emlékplakettet az egyetem rektorától: évente két-három diákja nyer különböző díjakat.**

**- Mi az IQSYS profilja?**

- 2003 márciusában a KFKI-csoport – amelynek része volt az IQSOFT is – IQSYS néven több szoftvercéget egybeolvasztott. Az így létrejött cég fő profilja a korszerű szoftvertechnológiákon alapuló egyedi alkalmazásfejlesztés.

Az IQSOFT története tulajdonképpen a hetvenes-nyolcvanas évekre nyúlik vissza. A mag az SZKI (Számítástechnikai Kutató és Innovációs Központ) Dömölki Bálint vezette Elméleti Laboratóriumában dolgozott. Az 1990-ben indult IQSOFT-ot sokféle aktivitás jellemezte, az alkalmazásfejlesztés mellett nagy hangsúlyt kapott a korszerű technológiák és módszerek bevezetése is. 1990 és 1993 között például az IQSOFT volt az Oracle kizárólagos magyarországi disztribútora. Később a cég fontos szerepet játszott az objektumorientált programozási

módszerek hazai bevezetésében, és máig is ennek a technológiának az éllovasa.
Visszatérve saját érdeklődési körömre: a logikai programozás viszonylag kisebb szerepet tölt be a cég életében. Pedig az IQ szócska ebből származott: a kilencvenes évek elején még úgy gondoltuk, hogy az MI fontosabb szerepet kap. Aztán az IQSOFT tevékenysége eltolódott a hagyományos technológiák irányába, de sokszor sikerül becsempészni egy kis MI-szerű részt egyes fejlesztéseinkbe. Emellett – részben az Európai Unió támogatásával – több kutatási projektünk is volt a logikai programozás területén. Én személy szerint főleg ezek vezetésében és kivitelezésében veszek részt. Az alapkutatást és az alkalmazást próbálom valamilyen módon egyensúlyba hozni. Úgy látom, van rá igény.

**- Mi a fő kutatási területe?**
- 1975 óta lényegében a logikai programozással illetve annak legelterjedtebb eszközével, a Prolog nyelvvel foglalkoztam. Ez a szakmai területem; egy kicsit kérdéses is, hogy az MI-hez tartozik-e. Határterület.
A logikai programozás alapgondolata, hogy programjainkat logikai állítások segítségével írjuk le, és ezek végrehajtását gépi következtetési módszerekkel végezzük. A következtetés az MI területére sorolható, tehát ilyen értelemben a logikai programozás is oda tartozik.
A Prolog meglehetősen sajátos programozási nyelv, alapvetően más, mint a többi. A hagyományos nyelvekben parancsokat, utasításokat írunk le, míg egy Prolog program leíró jellegű, kijelentésekből épül fel. Itt a programozás során megalkotjuk és

formálisan leírjuk annak a világnak a modelljét, amelyben a feladatokat meg akarjuk oldani. Ennek a modellnek a segítségével tud a Prolog rendszer a feltett kérdésekre választ adni.
A Prolog logikai nyelv az ún. deklaratív programozási nyelvek családjába tartozik, szemben a hagyományos, imperatív nyelvekkel. Az imperatív nyelveknél felszólító módban programozunk, és programjaink alapvetően az állapotváltozásra, a változtatható memóriára épülnek. A deklaratív nyelvek leíró jellegűek, kijelentő módban programozunk bennük, és nem használunk hagyományos értelemben vett program-változókat. A reláció-fogalomra építő logikai nyelvek mellett a deklaratív nyelvcsaládba tartoznak még a matematika függvényfogalmát használó funkcionális nyelvek is.

**- Hogyan jutott el a Prologig?**

- A Prolog története még a hatvanas évekre nyúlik vissza, amikor megszülettek az első automatikus tételbizonyítási módszerek. Először Amerikában kísérelték meg az alkalmazásukat, és meglehetősen rossz eredményeket tapasztaltak. Ennek fő oka az, hogy a tételbizonyítás folyamatában óriási keresési teret kell bejárnunk, hiszen minden ponton többféle következtetési szabályt alkalmazhatunk. Egy lépéssel később megint nagyon sok lehetőség közül választhatunk… Nagyon nehéz megtalálni az eredményhez vezető utat, a helyes bizonyítást. Emiatt az MI alkalmazások tekintetében az amerikaiak hamar eldobták a logikai módszereket, és speciális, pl. heurisztikákon alapú rendszereket dolgoztak ki.
Ezzel szemben, Európában– talán a kevésbé sikerorientált

szemlélet miatt – az 1970-es évek elején találtak egy, a tételbizonyítás szinte végletes leegyszerűsítésén alapuló kompromisszumot. Ez a Robert Kowalski (Edinburgh) és Alain Colmerauer (Marseille) nevéhez fűződő felfedezés azt eredményezte, hogy a tételbizonyítás folyamata valamilyen szintig ember által is követhető és befolyásolható lett, és így alapját képezhette a logikai programozásnak, azaz a logikai állításokkal történő programírásnak. Nem írhatunk akármilyen állításokat, csak speciális alakúakat, de óriási eredmény, hogy a logikai programok sokkal könnyebben olvashatók, mint a hagyományos, imperatív nyelven kódoltak. Azért is, mert viszonylag rövid, önállóan értelmezhető állításokból épülnek fel. Ha ezeket igaznak fogadjuk el, akkor ezzel már valamilyen szinten verifikáltuk is a program helyességét.

A matematikai logika számítógépes alkalmazásával első munkahelyemen, a NIM IGÜSZI-ben (NIM Ipargazdasági és Üzemszervezési Intézet) találkoztam 1972-ben. A Németi István által vezetett kutatócsoport foglalkozott ezzel a területtel. Készült egy automatikus tételbizonyító, és egy erre épülő program-verifikációs rendszer is. Már látszott – különösen az akkori számítástechnikai eszközök sebességét tekintve –, hogy ennek a megközelítésnek megvannak a korlátai.
Ugyanez a csapat hozta el Angliából az egyébként Marseille-ben készült Prolog első megvalósítását. Szörnyen felkeltette az érdeklődésemet: itt egy tételbizonyító, amellyel hasznos dolgokat lehet tenni!
Akkor vágtam a témába, azóta benne vagyok. Előtte leginkább a programozási nyelvek és megvalósításuk, azon belül a szintén

matematikusi szemléletű, mára a süllyesztőbe tűnt Algol 68 izgatott. Érdekes módon a Prologra is volt befolyása, mert a Prolog mai formáját az Algol 68 szintaxisleíró nyelvéből kölcsönözte. Mégpedig azért, mert a készítő, David Warren szintén Algol 68-cal foglalkozott a Prolog előtt.
A szakmai munkám első két évtizedében elég intenzíven használt CDL (Compiler Definition Language) is az Algol 68 egyik leszármazottja volt.
1975 májusában készítettem el CDL-ben az első magyar Prolog-rendszert. A Marseille-iről annyit tudtam csak, amennyi három előadás-fólia formájában eljutott hozzám. A fóliák alapján rekonstruáltam a Marseille-i megvalósítás működését. Azóta is ezeket a fóliákat használom amikor a Prolog megvalósítási módszereit tanítom…
Érdekes módon, a Prolog nagyon termékeny talajra talált Magyarországon. Azonnal három-négy alkalmazó csapat vetette rá magát. Akkoriban itthon viszonylag kevés MI-kutatás volt, talán azért is, mert nem volt alkalmas MI-nyelv (az Amerikában azóta is leggyakrabban használt MI-nyelv, a LISP, nem nagyon terjedt el Magyarországon). Így a LISP-nél lényegesen erősebb Prolog nagy lökést adott az MI-alapú fejlesztéseknek. Míg külföldön szinte csak egyetemeken foglalkoztak a Prologgal, addig a hazai fejlesztők ipari környezetben mozogtak, és így meglepően sok alkalmazási ötlet merült fel. A világon először nálunk készült nyomkövető a Prologhoz, és világújdonság volt a Futó Iván nevével fémjelzett T-Prolog szimulációs kiterjesztés is.
1976-ban, 1977-ben Angliából és más helyekről többen jöttek megnézni, mi folyik itt. Több tucatnyi kísérleti alkalmazás született, de – főként az akkori nagyon drága számítógép-idő

miatt – nagyon kevés jutott el a gyakorlati hasznosításig.

**- Hogyan látja a második magyar Prologot, az MPrologot?**

- 1978 és 1987 között dolgoztam rajta. Ez akkor egyike volt azon kevés magyar szoftvertermékeknek, amelyeket nyugatra lehetett exportálni. Az 1975-ös Prolog rendszert egy ideig bővítettük, foltozgattuk, de néhány év elteltével látszott, hogy érdemes újra kezdeni. Elölről, új architektúrával felépítettünk egy új Prolog-rendszert, amelyet MProlognak – Moduláris, vagy Magyar Prolognak – neveztünk el. Mindkétféleképpen lehet magyarázni. Amikor felbukkant az ötödik generációs japán projekt, az MProlog elég jó állapotban volt ahhoz, hogy a világon elsőként jelenjen meg a nagy számítógépek, az ún. mainframe-ek piacán. Mások szintén forgalmaztak Prolog rendszereket, de inkább a mini- és mikro-gépekre. Az MProlog másik nagyon jó tulajdonsága a széleskörű hordozhatóság volt: az akkor még nagyon sokszínű számítógép-paletta szinte minden architektúráján alkalmazható volt.

A nyolcvanas évekre a korábbi néhányszemélyes kutató-fejlesztő csapatból tíz-tizenöt-húszfős gárda alakult ki az SZKI-ban. Sőt, az észak-amerikai terjesztésre külön cég jött létre Kanadában, a LogicWare. A Quintus Prolog sajnos elég hamar kiszorította az ottani piacról. Az egyik ok az volt, hogy az MProlog még a hetvenes évek architektúráira épített, míg a David Warren vezette Quintus csapat az akkori legmodernebb módszereket alkalmazta. A másik esetleg az, hogy a kanadai partnereink nem jól találták el a hangsúlyokat.

Mindent összevéve,  termékként talán mégis az MProlog volt a legnagyobb eredmény az életemben.

**- A nyolcvanas években már gyakran dolgozott külföldön is. Merre fejlődött akkoriban a Prolog?**

- Túlzás, hogy gyakran: 1982-ben ösztöndíjasként fél évet töltöttem Edinburgh-ban és Londonban, majd 1987-től 1990-ig Manchesterben és Bristolban dolgoztam. Ugyanannál a David Warren professzornál, aki a Prolog egyik atyja.

De mielőtt eljutottam odáig, kellett hozzá, hogy a japánok felfedezzék a Prologot. 1981 nyarán meghívtak egy workshopra Los Angelesbe. Utóbb kiderült, ez annak volt köszönhető, hogy az amerikaiak már tudták, hogy a japánok 1982-ben be fogják jelenteni: a logikai programozást tekintik az általuk az évtized végére kidolgozandó ún. ötödik generációs számítógép-architektúra alapjának. Ezért szervezték a workshopot.

Sajnos a japánoknak több okból sem sikerült a kitűzött célokat elérniük. Egyrészt túlzott módon ambiciózusak voltak, másrészt egy kicsit idő előtt kezdték el erőltetni a párhuzamosságot. Még nem voltak meg a teljes Prolog nyelv párhuzamos végrehajtásához szükséges elméleti alapok és gyakorlati módszerek. 1987-től kezdve pont ilyeneken dolgoztam Angliában. A japánok akkor már kevésbé tudták ezeket – a hetvenes évek technológiájára épülő projektjükbe – beilleszteni. Nem mondanám, hogy az ötödik generációs projekt kudarcba fulladt, de a beígértnél sokkal kevesebb eredményt adott. És emiatt mindaz a pluszpublicitás, amelyet a nyolcvanas években kapott a téma, a következő évtizedben mínuszba váltott, negatív hatásként ütött vissza. Az egyensúly nagyjából most már visszaállt. Úgy érzem, ma a logikai programozás is az általánosan elfogadott programozási paradigmák egyike. A deklaratív

programozás általában is halad szép lassan előre, mert bizonyos területeken az átláthatóság, a biztonság különösen fontos. Olyan nagy cégek, mint az Ericsson vagy a Motorola szintén használnak deklaratív nyelveket.
Visszatérve a nyolcvanas évek végére: szerencsés voltam, mert a logikai programozás párhuzamos megvalósításainak egyik vezető kutatócsoportjában dolgoztam, Warren professzor vezetésével. 1977-ben ő készítette el az első – a mai szabvány szintaxisát megalapozó – Prolog fordítóprogramot az Edinburgh-i Egyetemen. 1983-ban szintén ő fejlesztette ki a ma is széles körben használatos megvalósítási modellt, az ún. Warren féle absztrakt gépet (WAM). A nyolcvanas évek végén vette az irányt a párhuzamos programozás felé, és akkor hívott munkatársának. Így kerültem Manchesterbe, majd, amikor átment Bristolba, én is vele mentem. Tulajdonképpen virtuális kutatócsoportról volt szó, mivel három helyen végeztük a munkát: a Svéd Számítástudományi Kutatóintézetben (SICS), a Chicago melletti Argonne National Laboratory-ban, és a Manchesteri, aztán a Bristoli Egyetemen.

**- Hogyan jött képbe, és mi a VAGY-párhuzamosság?**

- Kezdjük a Moore-törvénnyel: a gépek teljesítménye másfél évente megkétszereződik. Ezt a törvényt bizonyos idő után már a fénysebesség fogja korlátozni. A további sebességnövekedéshez egy chipre több processzort kell majd feltenni. És ekkor jön majd el a deklaratív programozás igazi ideje. Ha egy feladatot több processzorra kell szétosztanunk és úgy elvégeztetnünk, a változtatható memórián alapuló, Neumann János zseniális elméjéből származó architektúra komoly gondot jelent. Súlyos

bonyodalmak származhatnak abból, ha a memóriát – szinkronizálás nélkül – egyszerre többen módosítják.
Emiatt az imperatív programok párhuzamosítása még messzemenően nem triviális. Két imperatív eljáráshívásról nagyon nehéz eldönteni, hogy párhuzamosan futtathatóak-e, hiszen bármelyik módosíthatja a memória állapotát. Egy deklaratív programban viszont nem beszélhetünk állapotok változtatásáról, hanem nevén kell neveznünk az új állapotokat. Emiatt a deklaratív programok sokkal könnyebben párhuzamosíthatóak. Ez volt a japánok egyik alapgondolata is: az intelligens rendszerekhez egyrészt nagy számítási erő, azaz párhuzamosság, másrészt következtetési képesség kell. Mindkettőt megtalálták a logikai programozási paradigmában.
A VAGY-párhuzamosságról: logikai állítások megfogalmazásában az ember kötőszavakat használ: ÉS, VAGY, HA … AKKOR stb. A Prolog programok is ilyen kötőszavakkal épülnek fel. Amikor azt állítjuk, hogy valami, VAGY valami más igaz, a két állítás eldöntését, bizonyítását – amelyek egymástól független feladatok – nagyon könnyen rá tudjuk bízni két különböző processzorra.
A nyolcvanas évek végére értek meg azok az eredmények, amelyekkel lehetővé vált a Prolog programok alternatív ágainak párhuzamos bejárása. Az egyik első, s talán a legkomolyabb vívmány az Aurora rendszer volt, amit angolul *orora*-nak ejtenek, és akkor kétszer is szerepel benne az angol „vagy” szócska. A másik, Warrenhez fűződő magyarázat: ha az említett három csapat székhelyéhez – Chicagóhoz, Manchesterhez, Stockholmhoz – keresünk egy olyan pontot, ahonnan mindhárom egyforma távolságra van, akkor várhatóan valahol az Északi sark

környékén leszünk, s ott látható lesz az angolul *aurora* sarki fény.
Az Aurora eredményeit az Andorra rendszerben terjesztettük ki ÉS-párhuzamos folyamatokra is. Pillanatnyilag ezek nem élő kutatási témák, de jelen vannak más eredményekben.

**- Hol tart ma a logikai programozás?**
- A nyolcvanas évek dereka óta újabb eszközökkel bővült ki, amelyeket legtöbbször Prolog-rendszerekbe integrálnak. Például az úgynevezett korlát-programozással (*constraint programming*), amely egy nagyon izgalmas terület. Ez egy általános módszer, integrációs paradigma, amely különböző tudományágak – operációkutatás, MI stb. – eredményeit hozza össze a logikai programozás kalapja alatt, így tudnak egymással szimbiózisba lépni.
A mostani, lassan húszéves – szintén szójátékon alapuló (a fejlesztő svédországi SICS kutatóintézetre, a hatodik generációra, illetve a Sixtusi Kápolnára utaló) – SICStus Prologban többféle korlát-kiterjesztés is megtalálható. Így például lehetőségünk van lineáris egyenlet- ill. egyenlőtlenség-rendszerek megoldására, ítéletkalkulusbeli formulákon való következtetésre, valamint az ún. korlát-kielégítési problémák (CSP – *Constraint Satisfaction Problems*) megoldására.
A korlát-logikai programozás eredményei annyira meggyőzőek, hogy módszereit hagyományosan elfogadottabb környezetbe, például C++-ba, Jávába is átültetik. Erre példa a New York-i tőzsdén is jegyzett francia cég, az ILOG programcsomagja, amely optimalizációs, ütemező és konfigurációs feladatok megoldására alkalmas szolgáltatásokat nyújt C++-ban.

**- Miből állt a 2000 és 2002 közötti Európai Uniós SILK (System Integration via Logic and Knowledge) projekt, melyben részt vett?**

- A SILK a logikai programozás világát az élvonalbeli kommerciális technológiákkal kapcsolja össze. Közismert szituáció az az alapvető feladat, amelyet a SILK segítségével szeretnénk támogatni: valamely vállalkozás, intézmény életében sok számítástechnikai alkalmazást fejlesztettek ki az elmúlt évtizedekben. Sokszor nem jutott idő arra, hogy ezeket egységes alapra helyezzék, információcserére alkalmassá tegyék. Így kialakult független alkalmazásoknak egy készlete; mindegyiket más-más csoport használja. Nyilvánvaló, hogy fontos ezeknek az információforrásoknak az összeépítése.

Erre két út képzelhető el: az egyik, a revolúciós megközelítés, egy új vállalatirányítási rendszer megvásárlása, és az arra való átállás – ez általában nagyon nagy beruházást igényel. Az evolúciós átmenet a másik lehetőség: a rendszereket megpróbáljuk fokozatosan összekapcsolni és felettük egy virtuális egyesített adatbázist felépíteni. Ez utóbbi átmenetet támogatja a SILK eszközkészlet.

A heterogeneitásból származó problémák áthidalását viszonylag klasszikus technológiával, csatolók készítésével oldottuk meg. A különböző adatforrásokban lévő információk szemantikus összekapcsolása viszont sokkal nehezebb. Ennek a feladatnak a megoldásában használtuk a logikai programozást, míg specifikációs nyelvként az objektumorientált világ vezető szabványosítási szervezetének, az Object Management Groupnak, az OMG-nek a szabványait fogadtuk el. Így az UML

(Unified Modelling Language) segítségével írjuk le az adatforrások modelljeit, míg az OCL (Object Constraint Language) nyelvet használjuk a modell-elemek közötti kapcsolatok leírására. Viszonylag komoly szoftver készült: a grafikus felülete Jávában van, míg az agyát a logikai programozás és az MI adja. Nagyon izgalmas terület; részben hazai, részben EU kutatási projektekben próbáljuk folytatni.

**- Miben látja a kutatásfejlesztési projektek sikerének, eredményességének titkát?**

- Egy K+F projekt sokkal nehezebben tervezhető, mint egy ipari fejlesztési projekt, hiszen a megoldási módszerek, technológiák, sőt általában a pontos követelmények sem ismertek előre. Emiatt nagyon fontos az, hogy a K+F projekt előrehaladása során rendszeresen megmérettessék, mind a finanszírozó szervezet, mind a rokonterületeken dolgozó társkutatók által. Ebből a szempontból jobb helyzetben vannak az alkalmazott kutatási projektek, hiszen itt komoly előnyt jelenthet az eredmények alkalmazásában érdekelt ipari partnerek részvétele.
A hosszabb távú kutatások esetén – és a mai kutatási témák többsége ilyen – a jelenlegi, 2-4 éves projekt-időtartamot lehetővé tevő finanszírozási gyakorlat komoly gátat jelent. A komolyabb eredmények eléréséhez így általában egymáshoz kapcsolódó projekt-láncokra van szükség, és ez a lánc sokszor megszakad ...

**- Ha visszamehetnénk az időben, és most lenne egyetemista, mivel foglalkozna legszívesebben? Milyen témakörben, kutatási területben látna komoly perspektívát?**

- Mint már említettem, közel harminc éve foglalkozom logikai programozással. A programozási nyelvek tudománya néhány évtizedes múlttal bír, míg a logika nyelve több mint 2000 éves (és a modern logika is több mint egy évszázada született). Véleményem szerint a logika, mint programozási nyelv sokkal kiforrottabb és sokkal közelebb áll az emberi gondolkodáshoz, mint a ma általánosan használatos programozási nyelvek. Úgyhogy a logikai programozás területét most is aktuálisnak, fontosnak tartom, és erről jónéhány mai egyetemistát is sikerül meggyőznöm :-).
A számítógépek teljesítménynövekedése azt jelenti, hogy a logikai programozás, nagyobb számításigénye ellenére, egyre szélesebb körben alkalmazható. Egy ilyen izgalmas, új terület a „mindenütt jelen levő intelligencia” (Ambient Intelligence), amely a közelmúltban került az Európai Unió informatikai kutatási programjainak a középpontjába.

**- Milyen alapelveket érdemes képviselni ahhoz, hogy a csúcstechnológiai kutatásokban komoly eredményeket érjünk el?**

- A kutatási munkákhoz pénz, berendezések, és természetesen jó kutatók kellenek. Az első kettőt egyik napról a másikra meg lehet szerezni, de a a jelenlegi magyar kutatói utánpótlás magas színvonala mögött a magyar oktatási rendszer elmúlt évszázadának teljesítménye áll (hogy mást ne említsek, gondoljunk csak a 110 éves Középiskolai Matematikai és Fizikai Lapokra). Csak remélni tudom, hogy az oktatási rendszerek jelenleg folyó átalakítása, az Európai Unió oktatási struktúrájához való igazítás során sikerül megőrizni a magyar

iskolai és egyetemi oktatás magas színvonalát...

http://cs.bme.hu/~szeredi (Szeredi Péter)