# DEKLARATIV NYELVEK FÜGGŐSÉGI ANALIZISE ÉS TANULÁSA

című Doktori Értekezés Tézisei

Szerző: Kocsisné Szilágyi Gyöngyi
Témavezető: Dr. Tibor Gyimóthy

Informatikai Doktori Iskola
Szeged
2003

# Absztrakt

*Ezen tanulmány a "Deklaratív Nyelvek Függőségi Analízise és Tanulása" című doktori értekezés eredményeinek tézisszerű összefoglalását tartalmazza.*
*Az* **imperatív** *programozási nyelvek mellett egy másik fontos programozási paradigma a* **deklaratív** *irány. A deklaratív programok sokkal leíróbb jellegűek, közelebb állnak az emberi gondolkodáshoz.*

*A doktori értekezés három fő deklaratív irányvonallal foglalkozik, a* **Logikai Programozással (LP)** *[20,17], a* **Korlátozásos Logikai Programozással (CLP)** *[18,12], és az* **Attribútum Nyelvtanokkal** *[1]. Ezen területek közt szoros kapcsolat áll fenn, [6] a Logikai Programok Attribútum Nyelvtanokkal történő leírásával foglakozik, a Logikai Programok pedig a Korlátozásos Logikai Programok (CLP) speciális eseteként kezelehetők [12]. A doktori értekezés egyik fő témája a Logikai illetve Korlátozásos Logikai Programok szeletelése. A* **szeletelés** *[11] egy olyan program analízis technika, mely segíti az adatfolyam jobb megértését, a nyomkövetést, a párhuzamosítást stb. Egy adott változóra vonatkozó program szelet a program azon részeit tartalmazza, melyek hatással lehetnek a változóra ("hátrafele irányuló szeletelés"), illetve amelyekre hatással lehet az adott változó ("előre irányuló szeletelés").*

*A doktori értekezésben kidolgozásra került egy olyan* **dinamikus szeletelő algoritmus** *amely az adatfolyam vizsgálata mellet a kontroll függőségeket is figyelembe veszi, így segítve a program összefüggő komponenseinek megtalálását, illetve a különböző típusú program hibák forrásának feltárását.*

*Megadunk egy olyan általános szelet definíciót mely az SLD-fa sikeres ágai mellett a sikertelen ágakra is érvényes. Az adatfolyam analízisnek a sikertelen ágakra ilyen módon történő kiterjesztése segíti már létező nyomkövetési technikák hatékonyságának növelését, "holt" kódrészek megtalálását, a program jobb megértését, párhuzamosítását, stb.*

**A Korlátozásos Logikai Programozás (CLP)** *két deklaratív programozási paradigma, az egyenlet megoldás és a Logikai Programozás (LP) fúziójaként jött létre. A doktori disszertációban részletesen kidolgozzuk a* **Korlátozásos Logikai Programok Szeletelésének** *elméleti hátterét [24,25,26], mely megfelelő alapot teremt változó megosztáson alapuló szeletelési algoritmusok kidolgozására is.*

*Az* **Induktív Korlátozásos Logika Programozás** *egy új kutatási irány, mely a Korlátozásos Logikai Programozás (CLP) [12,18] területén induktív logikai tanulási módszereket alkalmaz (ILP [19,16]).*

*A disszertációban ismertetésre kerül egy Korlátozásos Logikai Programok tanulására kidolgozott olyan módszer, mely CLP programok specializálásán alapul [28]. A specializáló eljárás kombinálja az úgynevezett "kicsomagolás" (unfolding) transzformációt klóz elhagyással [3], illetve a hatékonyság növelésének érdekében szeleteléssel.*

*A doktori értekezésben formalizáljuk az unfolding-on alapuló tanuló algoritmus alapvető fogalmait CLP-re vonatkozóan, kimondjuk és bizonyítjuk az ezekhez tartozó tételeket, például, hogy az unfolding transzformáció megőrzi a CLP programok műveletei és logikai szemantikáját. Formalizáljuk a tanuló algoritmusokat, és belátjuk azok helyességét.*

*A Fordító Programok világában az* **Attribútum Nyelvtanok (AG)** *használata az egyik legszélesebb körben alkalmazott módszer [29,15,1]. Az Attribútum Nyelvtanok a Környezet Független Nyelvek [14] általánosításának tekinthetők. Az Attribútum Nyelvtanokat gyakran használják különböző programozási nyelvek specifikálására és implementálására. Mivel az Attribútum Nyelvtanoknak, illetve szemantikus függvényeinek definiálása sokszor nagyon sok munkával jár, ezért igen hasznos lehet egy olyan eszköz kidolgozása, mely példák alapján képes AG-k szemantikus függvényeinek következtetésére. A Disszertációban bemutatunk egy olyan párhuzamos tanuló eljárást, mely az LP és az AG közti kapcsolaton alapulva [6] képes AG-k szemantikus függvényeinek tanulására. Egy a Logikai Programokra kidolgozott tanulási módszer lett adoptálva Attribútum Nyelvtanokra a [8] cikkben (Gyimóthy T. és Horváth T.), a Doktori értekezésben megadjuk ennek egy párhuzamosított változatát. A párhuzamosság nem csak hatékonyabb végrehajtási időt eredményez, hanem segít az interaktív algoritmus során a felhasználónak feltett kérdések számát is redukálni [27].*

## 1. BEVEZETÉS

A következőkben megadunk néhány fontos alapvető definíciót, hogy formalizálni tudjuk eredményeinket. A többi definíciót a Doktori értekezés részletesen tárgyalja.

**Definition 1 Korlátozásos (Constraint) Logikai Program**

*A* **constraint domain** *egy $\langle L, D\rangle$ pár, ahol L egy olyan A abc feletti elsőrendű nyelv, melynek az abécéje tartalmazza a predikátum szimbólumokat (beleértve az egyenlőséget is), a függvény szimbólumokat (beleértve a konstansokat is), és a változókat. D pedig egy alaphalmaz (domain).*

*Az L-beli függvényszimbólumok mindegyike adott interpretációval rendelkezik D-n. A* **konstansok** *0 aritású függvényszimbólumoknak tekinthetőek.*
*Egy* **term** *egy változó, egy konstans, vagy egy olyan n-aritású ($n > 0$) függvényszimbólum, melyet egy termekből álló elem n-es követ (ez utóbbit összetett termnek nevezik).*
*Az L-beli* **predikátumszimbólumok** *két diszjunkt halmazra vannak felosztva a következő módon:*

1. *constraint predikátumok $\Sigma$, melyek D-n előre megadott fix interpretációval rendelkeznek (ide tartozik az $=$ szimbólum is)*
2. *definiált predikátumok $\Pi$, melyek megjelenhetnek egy program klózban fejbeli predikátumként, és melyekre a felhasználó adhat meg megfelelő interpretációt D-n.*

*Egy* **definiált atom** *egy formula a $p(t_1, \cdots, t_n)$ alakban, ahol p egy n-aritású definiált predikátum ($p \in \Pi$), és $t_1, \cdots, t_n$ termek.*
*A* **constraint atomok** *olyan formulák, melyek* constraint predikátumokból *vannak konstruálva (egy előre adott interpretációval). Egy constraint-re tipikus példa egy olyan lineáris egyenlőség (illetve egyenlőtlenség), ahol az alkalmazott constraint predikátum a racionális számok halmazán interpretált $=$ szimbólum, az alaphalmaz a racionális számok halmaza: $X - Y = 1$ (illetve $X - Y \leq 1$).*
*Egy* **klóz** *egy olyan $h : -b_1, ..., b_n$, $n \geq 0$ alakú formula, ahol $h, b_1, ..., b_n$* atomi formulák. *A $b_1, ..., b_n$ predikátumok lehetnek constraint illetve definiált predikátumok is, de a fejbeli h predikátum egy definiált predikátum.*
*Egy* **cél (goal)** *egy fej nélküli klóz. Egy* **tény (fact)** *egy olyan $h \leftarrow c_1, \cdots, c_n$ klóz, ahol $c_1, \cdots, c_n$ constraint predikátumok.*
*Egy* **constraint logikai program** klózok *halmazából áll.*

A **Logikai Programok** csak definiált predikátumokat tartalmaznak.

**Definition 2 Szkeleton (váz)**
*Egy P programhoz tartozó szkeleton egy olyan címkézett, rendezett fa, melyre:*

- *A gyökér címkéje egy cél (goal) klóz*
- *Az egyes csomópontok $< c, \sigma >$ alakú klóz példányokkal vannak címkézve; néhány csomópont lehet "?"-vel is címkézve, az ilyet* nem teljes *csomópontnak nevezzük.*
- *minden nem levél csomópontnak pontosan annyi gyermeke van, ahány definiált (nem constraint) predikátum van a testében.*
- *Az i-edik gyerek fej predikátuma ugyanaz, mint az adott csomópontot címkéző klóz i-edik definiált predikátuma.*

**Definition 3 Szkeletonhoz tartozó constraint halmaz**
*Egy adott S szkeletonhoz tartozó $C(S)$ constraint halmaz a következő constraint-ekből áll:*

- *az S csomópontjait címkéző klózokhoz tartozó constraint-ek.*
- *az összes $\vec{x} = \vec{y}$ alakú egyenlőség, ahol $\vec{x}$ jelöli az S szkeleton n csomópontját címkéző klóz i-edik definiált test atomjának argumentumait, és $\vec{y}$ jelöli a hozzá tartozó gyerek (az n csomópont i-edik gyereke) fej atomjának argumentumait. Nem hozunk létre egyenlőséget, ha az n csomópont i-edik gyermeke nem teljes.*

Egy adott $P$ programhoz tartozó **derivációs (levezetési) fa** egy olyan szkeleton, melyhez tartozó constraint halmaz kielégíthető. Ha a szkeleton teljes (nem tartalmaz nem teljes csomópontot), akkor a levezetési fát **bizonyítási fának** hívjuk.
A szeletelés definiálása során hivatkozni fogunk **"program / levezetési fa / bizonyítási fa pozícióra"**. A szeletelés mindig egy változó adott előfordulására, egy adott pozícióra nézve kerül megkonstruálásra. Egy $T$ levezetési fa (vagy szkeleton) összes pozíciójának halmazát $Pos(T)$-vel fogjuk jelölni.

## 2. LOGIKAI PROGRAMOK ADATFOLYAM ANALíZISE

A szerzőnek a témához kapcsolódó tudományos publikációi: [23,10,9].

A **Logikai Programok [20,17] adatfolyam analízise** fontos szerepet játszik többek közt a nyomkövetésben, tesztelésben és program megértésben.
A **Szeletelés** [11] egy olyan program analízis technika, melyet eredetileg imperatív programokra dolgoztak ki. A szeletelés szintén alkalmazható az előzőekben felsorolt területeken. Egy adott változóra vonatkozó program szelet a program azon részeit tartalmazza, melyek hatással lehetnek a változóra ("visszafele irányuló szeletelés"), illetve amelyekre hatással lehet az adott változó ("előre irányuló szeletelés").
A Logikai Programok adatfolyama nem explicit, ezért az imperatív programoknál alkalmazott szeletelési módszerek közvetlenül nem adoptálhatóak. Az adatfolyam implicit volta tovább nehezíti a program viselkedésének megértését. Ezért olyan program elemző eszközök, melyek segítik az adatáramlás megértését nagy gyakorlati hasznossággal bírnak. Ebben a témában az egyik kutatási eredmény korábbi Prolog programokra kidolgozott nyomkövetési módszerek [22] optimalizálása, illetve a program analízis segítése szeletelési technika alkal mazásával. A szeletelési technika egy úgynevezett Program Függőségi Gráfon alapul. Kidolgozásra került egy olyan **dinamikus szeletelő algoritmus** amely az adatfolyam vizsgálata mellet a kontroll függőségeket is figyelembe veszi, így segítve a program összefüggő komponenseinek megtalálását, illetve a különböző típusú program hibák forrásának feltárását.
Megadunk egy olyan általános szelet definíciót mely az SLD-fa sikeres ágai mellett a sikertelen ágakra is érvényes. Az adatfolyam analízisnek a sikertelen ágakra ilyen módon történő kiterjesztése segíti már létező nyomkövetési technikák hatékonyságának növelését, "halott" kódrészek megtalálását, a program jobb megértését, párhuzamosítását, stb.
Elkészült egy prototípus is Prolog programok szeletelésére alkalmazva a fenti meggondolásokat, mely nyomkövetéssel lett kombinálva.

Vizsgáljuk meg a következő példát.

**Example 1 Csupán az adatfolyam analízisének vizsgálata nem elég a hiba forrásának feltárásához**

*A hibás program a következő:*

*1. p(A,X) :- q(A,X).*
*2. q(A,X) :-* $\mathbf{A > 0}$*, X is 2.*
*3. q(A,X) :- X is 3.*

*A helyes program ez lenne:*

*1. p(A,X) :- q(A,X).*
*2. q(A,X) :-* $\mathbf{A = 0}$*, X is 2.*
*3. q(A,X) :- X is 3.*

Végrehajtva a fenti (hibás) programot a $p(0, X)$ célra a program által adott válasz $X = 3$ lesz, bár az általunk elvárt megoldás $X = 2$ lenne. Ezért valahol egy hiba kell hogy legyen a programban. Végrehajtva az $X$ változóra nézve egy "sima" adatfolyam analízisen alapuló szeletelést, a kapott szelet nem tartalmazza a hibás $A > 0$ predikátumot, mert $X$ nem függ közvetlenül a 2. klóz predikátumaitól, csupán kontroll függőség áll fenn köztük. Ez azt jelenti, hogy ha az $A > 0$ predikátum másképp értékelődött volna ki (igazra az esetünkben), akkor ez hatással lett volna az $X$ változó értékére is. Az általunk javasolt új szeletelési technika, melyet "Debug slice"-nak hívunk már tartalmazza az $A > 0$ predikátumot, figyelembe veszi a kontroll függőségeket is.

*2.1. Logikai Programok Kibővített SLD-fája, Szkeleton(n), Levezetési-fa Függőségi Gráfja*

Egy $P$ programnak egy adott célra vonatkozó lehetséges levezetései reprezentálhatóak az SLD-fa segítségével. Az SLD-fa minden egyes ága pontosan egy levezetésnek felel meg [21]. A sikeres levezetésnek megfelelő ágakat *sikeres ágaknak* nevezzük, a sikertelenekhez tartozókat *sikertelen ágaknak*, míg a végtelen levezetéshez tartozókat *végtelen ágaknak* hívjuk.
Egy SLD-fának lehet sok sikertelen, és néhány, vagy csak egy sikeres ága. A *cut* (!) predikátum használható a Prolog logikai programozási nyelv esetén arra, hogy megakadályozza a felesleges sikertelen ágak felépítését. A *cut* beépített predikátum hatása a következő: a "!" sikeres végrehajtása után az interpreter már nem hajthat végre visszalépést az adott klózban a "!" baloldalán lévő predikátumokra vonatkozóan, viszont a tőle jobbra lévő predikátumokra már a normál végrehajtási szabály vonatkozik. Jelöljük $cut(W)$-vel az SLD fa ágának azt a részét, ami magába foglalja azokat a csomópontokat, melyekre a "!" ily módon hatással van. Ezen információkkal kiegészítve az SLD-fa definícióját egy olyan struktúrát kaptunk, melyet *Bővített SLD-fának* neveztünk el. Az általunk definiált dinamikus szeletelő algoritmus (Debug slice) ezen struktúra egy szeletét adja eredményül. Azonban arra nem alkalmas ez a fa szerkezet, hogy az adatfolyam analízist végrehajtsuk rajta. Ehhez a már előzőekben definiált úgynevezett $Szkeleton(n)$ struktúrát használtuk, ahol $n$ egy levél csomópontot jelöl az SLD-fában. Egy-egy értelmű megfeleltetés van a $T$ SLD-fa $n$ levél csomóponttal azonosított egy ága, illetve a megfelelő $S$ $Szkeleton(n)$ közt. Ezt a megfeleltetést adja meg a $\phi : nodes(S) \rightarrow nodes(T)$ leképezés.

A $Szkeleton(n)$ definícióját még tovább bővítettük változók lekötésére vonatkozó információval.

Egy derivációs fa adatfolyamát szeretnénk reprezentálni valamilyen módon. Egy logikai programban az adat kétféle módon áramolhat: egyrészt egyik klózból a másikba unifikáción keresztül, másrészt egy klózon belül egy adott változó többszöri előfordulása vezethet adatfüggőséghez [4,5]. A következő definíció tükrözi ezeket az információkat.

**Definition 4 Levezetési Fa Függőségi Gráf (PTDG: $T_{g,n} = (Pos(S), \sim_T)$)**
*Legyen $T$ egy SLD-fa a $g$ célra vonatkozóan, $n \in nodes(T)$ egy levele $T$-nek, és $S$ legyen Szkeleton(n), $\beta, \delta \in Pos(S)$.*

- *A PTDG csomópontjai a $Pos(S)$ elemei.*
- *$\beta \sim_T \delta$ pontosan akkor, ha a következő feltételek valamelyike fennáll:*
  1. *$\beta$ és $\delta$ közös változót tartalmaznak a $V$ változóhalmazban (***lokális él***)*
  2. *a $\delta$ predikátuma a $\beta$ predikátumával volt unifikálva, és $\beta$ is és $\delta$ is a hozzájuk tartozó predikátumok k-dik pozíciói (***tranzíciós él***).*

A fenti definícióból már látszik, hogy a függőségi gráf az SLD-fa egyetlen ágára van megkonstruálva (az $n$-nel azonosított ágára), Szkeleton(n)-re. Természetesen a PTDG megkonstruálható minden Szkeleton(n)-re ($n$ a $T$ egy csomópontja), azaz a $T$ SLD-fa minden ágára.

Ezt a gráfot tovább bővítjük adatfolyam irányítási információkkal ($\vec{T}_{g,n} = (Pos(S), \rightarrow_T)$).

*2.2. Általános szelet definíció és a Debug Slice*

## *1. Tézis*

A továbbiakban egy *általános szelet definíciót* adunk meg, amely azt mutatja, hogy a Szkeleton(n) egy adott argumentum pozíciója mely más pozícióktól függ (az adatfolyam szempontjából).

**Definition 5 Slice($T_{g,n}, \alpha$)**
*Legyen $P$ egy logikai program, $T$ egy adott $g$ célra vonatkozó SLD-fa, $n \in nodes(T)$ a $T$ egy levélcsomópontja, $S$ Szkeleton(n), és $\vec{T}_{g,n} = (Pos(S), \rightarrow_T)$ a hozzá tartozó Irányított Levezetési Fa Függőségi Gráf. Legyen $\alpha \in Pos(S)$.*
*A $\vec{T}_{g,n}$ egy $\alpha$ pozícióra vonatkozó* **szelete** *($Slice(T_{g,n}, \alpha)$):*

- *a $\vec{T}_{g,n}$ egy részgráfja*
- *egy $\beta \in Pos(S)$ pontosan akkor van benne a szeletben, ha $\beta \rightarrow_T^* \alpha$*

Ez egy olyan általános szelet definíció, amely az SLD-fa egy adott ágára vonatkozik, ez az ág lehet egy sikertelen levezetéshez tartozó ág is.

A Debug slice megkonstruálásához első lépésként az ún. **Potenciális Predikátum Függőségi Halmaz**t hozzuk létre (PDPS).

**Definition 6 Potenciális Predikátum Függőségi Halmaz(PDPS)**
*Legyen $P$ egy logikai program, $T$ a $g$ célhoz tartozó levezetési fa. A $T$ egy csomópontjának legbaloldalibb (kiválasztott) predikátuma benne van a* **Potenciális Predikátum Függőségi Halmaz (PDPS)***-ban, ha aktuálisan nincs ugyan hatással $T$-nek egy sikeres ágához tartozó predikátum argumentumára, de hatással lehetne rá, ha logikailag máshogyan értékelődött volna ki.*

A következő tétel azt határozza meg, hogy mely predikátumok tesznek eleget ezen feltételnek.

**Theorem 1 Potenciális Predikátum Függőségi Halmaz**

*Legyen $P$ egy logikai program, $T$ egy a $g$ célra vonatkozó levezetési fa. Ekkor*
*$PDPS = \{T$ sikeres ágaihoz tartozó predikátumok$\} \cup \{T$-nek a sikertelen levélcsomópontjai$\}$.*

**Definition 7 Debug Slice**
*A $g$ célhoz tartozó Bővített SLD-fa ($T$) Debug Slice-a a következő halmaz:*

*Debug slice= $PDPS \cup Data\_Flow\_of\_PDPS \cup Cut(W)\_Set =$*

$=$ $\{A\ T$ *sikeres ágainak a predikátumai*$\}$
$\bigcup \{A\ T$ *sikertelen levélcsomópontjaihoz tartozó predikátumok*$\}$
$\bigcup \phi(\cup_{n,\alpha}\{k \in nodes(S) \mid k$*-nak van legalább egy argumentum pozíciója* $slice(T_{g,n}, \alpha)$*-ben,* $\alpha\ p$ *egy argumentum pozíciója,* $n$ *a* $T$ *egy sikertelen levélcsomópontja*$\}$*)*
$\bigcup$ *(*$\cup_{Mark}$ $\{p \in nodes(T) \mid p$ *egy legbaloldalibb predikátum azon a* $T$*-beli úton, amely a* $cut(Mark)$ *-ot legbaloldalibb predikátumként tartalmazó csomópontból indul és megy felfele a* $Mark$*-al azonosított csomópontig*$\}$*)*

□

A Debug Slice ötlete az imperatív nyelveken végrehajtott vizsgálatokból származik, amelyet ott "releváns slice"-nak hívnak [7]. A releváns slice használható olyan hibaforrások feltárására, amelyek nem találhatóak meg csupán az adatfolyam analízisének segítségével. A releváns slice esetén az adatfüggőségek ki vannak egészítve ún. kontroll függőségekkel.

A keresési stratégia vezérlését befolyásolják egyrészt **(1)** a predikátumok logikai kiértékelése (az interpreter folytatja a keresést sikeres predikátum kiértékelés esetén, visszalép sikertelen esetén), és **(2)** egy speciális beépített predikátum *cut*(!) (a cut sikeres végrehajtása után nem lehet visszalépni a cut-tól balra lévő literálokhoz). Az általunk létrehozott Debug Slice definíció figyelembe veszi ezt a két fajta kontrol hatást, valamint ki van bővítve **(3)** azon predikátumok adatfolyam függőségével, amelyek részt vesznek a kontrol

függőségekben. Erre a kiterjesztésre az ad okot, hogy egy $p$ predikátum sikertelenségét okozhatta egy a $p$-t az adatfolyamon keresztül elérő hibás adat.

A Debug Slice egy informális definíciója a következő.
Legyen $P$ egy logikai program, $T$ egy a $g$ célra vonatkozó Bővített SLD-fa. A **T Debug slice-a** a következő predikátumokat tartalmazza:

1. A Potenciális Függőségi Halmazhoz (PDPS) tartozó predikátumok
2. A PDPS-hez tartozó predikátumok által meghatározott adatfolyam által specifikált predikátumok
3. $T$-nek valamely cut(W)-jéhez tartozó predikátumai

A tesztelés során különböző típusú hibákat sikerült feltárni, amelyeket szimpla adatfolyam analízist alkalmazva nem sikerült megtalálni, viszont a Debug Slice tartalmazta azokat. Ilyen hibatípus pl. mikor a cut rossz helyre került, egy sikertelen predikátum el lett gépelve (a neve vagy az aritása), vagy egy relációs jel $(<, >, =)$ lett elrontva, illetve egy rossz adatérték érte el a sikertelen csomópontot.

## 3. KORLÁTOZÁSOS LOGIKAI PROGRAMOK ADATFOLYAM ANALíZISE

A szerző ezen témához tartozó cikkei: [24,25,26].

A disszertáció egy másik eredménye a **Korlátozásos Logikai Programok Szeletelése**.
A **szeletelés pontos elméleti háttere** került kidolgozásra (a minimális szelet megtalálása általános esetben eldönthetetlen probléma). Az elméleti alapozás segített abban, hogy kidolgozzunk olyan **szeletelési technikák**at (dinamikus, statikus), amelyek **változó megosztás** vizsgálatán alapulnak. A módszereket tovább bővítettük lekötési információk figyelembevételével. Egy prototípus is implementálásra került.

A szeletelés célja, hogy megtaláljuk a program azon részeit, amelyek hatással lehetnek egy programbeli $X$ változóra (lásd Figure 1).

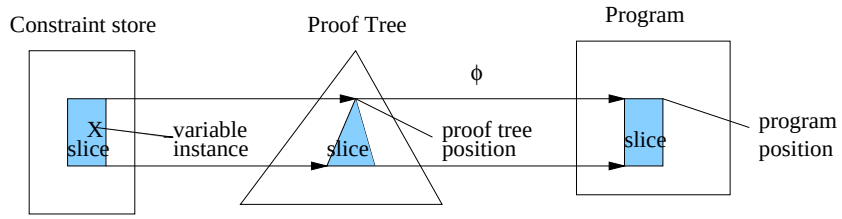


Fig. 1. Egy constraint halmaz, egy levezetési fa és egy programszelet.

Először egy constraint halmaz szeletét definiáljuk, amit felhasználunk egy levezetési fa szeletének meghatározásához. Végül a programszeletet a hozzá tartozó levezetési fák szeletelése segítségével adjuk meg.
Ezen tézis formalizálásához használjuk fel a következő definíciókat, jelöléseket.

Legyen $C$ egy constraint halmaz. $T$ egy levezetési fa a hozzá tartozó $C(T)$ constraint halmazzal. A $C(T)$-beli változók a $T$ fa pozícióiból származnak. Legyen $\mathcal{P}$ a $T$-beli pozíciók egy halmaza, azaz $\mathcal{P} \subseteq Pos(T)$. Ekkor $\Psi(\mathcal{P})$ azon $C(T)$-beli változó előfordulásokat azonosítja, amelyek $\mathcal{P}$-beli pozíciókból származnak. Jelölje $C_{\mathcal{P}}$ az összes olyan $C(T)$-beli constraint-et, amelyek tartalmazzák ezen változókat.

A $T$-beli összes pozíció egy programpozíció vagy egy célpozíció egy példánya. Ez megad egy természetes $\Phi_T$ leképezését a $T$-beli pozícióknak a program ill. a célpozícióiba. Jelölje $\Phi_T^{-1}(q)$ azon levezetési fabeli pozíciók halmazát, melyekre ha $r \in \Phi_T^{-1}(q)$, akkor $\Phi_T(r) = q$.

**Definition 8 Egy constraint halmaz megoldása**
*Egy $X$ változónak egy $v$ értékre történő kiértékelését a $C$ megoldásának nevezzük pontosan akkor, ha létezik egy olyan $\nu$ kiértékelés, melyre $\nu(X) = v$ és $\nu$ kielégíti $C$-t. A $C$ constraint halmaz $X$ változóra vonatkozó megoldásainak halmazát jelölje $Sol(X, C)$.*

# 2. Tézis

## 2.a A Korlátozásos Logikai Programokra, Bizonyítási Fákra és hozzájuk tartozó Constraint Halmazra vonatkozó szeletelés elméleti hátterének pontos formalizálása

**Definition 9 A $C$ kielégíthető constraint halmaz egy szelete**
*Egy $C$* **kielégíthető constraint halmaz egy $X$ változóra vonatkozó szelete** *egy olyan $S \subseteq C$ részhalmaz, melyre $Sol(X, S) = Sol(X, C)$.*

A 9. definíciót felhasználva definiáljuk az $X$-re vonatkozó **minimális szelet** fogalmát.

**Definition 10 $C$ egy minimális szelete**
**A $C$ egy minimális szelete** *egy olyan $S$ szelete $C$-nek az $X$ változóra vonatkozóan, melyet ha tovább redukálunk $S'$-t kapva, akkor $Sol(X, S')$ különbözni fog $Sol(X, C)$-től.*

Megjegyezzük, hogy a $C$ halmaz maga egy szelete önmagának, és szeletek bővítése is szelet. A fenti definíció nem adja meg közvetlenül, hogyan lehet egy minimális szeletet meghatározni. A minimális szelet nem egyértelmű. Általános esetben a minimális szelet megtalálása eldönthetetlen problémát eredményez.
A következőkben a Bizonyítási Fa Szelet definícióját adjuk meg.

**Definition 11 Egy Bizonyítási fa Szelet**
*Egy $T$* **bizonyítási fa egy $X$ változóra vonatkozó szelete** *egy olyan $\mathcal{P}$ részhalmaza a $T$ pozícióinak, hogy $C_{\mathcal{P}}$ a $C(T)$-nek egy $X$-re vonatkozó szelete.*

Végül egy CLP program szeletét definiáljuk egy adott változó pozícióra vonatkozóan.

**Definition 12 Egy CLP program szelet**
*A $P$* **CLP program egy adott $q$ program pozícióra vonatkozó szelete** *egy olyan $S$ részhalmaza a $P$ pozícióinak, hogy $P$ bármely $T$ levezetési fájára valahányszor $T$-nek egy $r$ pozíciója benne van a $\Phi_T^{-1}(q)$ halmazban, mindannyiszor létezik egy $Q$ $r$-re vonatkozó szelete $T$-nek, úgy hogy $\Phi_T(Q) \subseteq S$.*

## 2.b CLP programok, Bizonyítási fák és Constraint halmazok Adatfolyam Analízisen Alapuló Automatikus Szeletelése

Általános esetben eldönthetetlen probléma, hogy egy constraint halmaz részhalmaza szelet-e vagy sem. A következőkben olyan szintaktikai elemzésen alapuló szeletelési módszereket mutatunk be, melyek a szemantikus definíciónak is eleget tesznek mind programra, mind bizonyítási fára, mind constraint halmazra vonatkozóan.

**Definition 13 Constraint halmaz közvetlen függőségi relációja**
*Legyen $C$ egy constraint halmaz, és $vars(C)$ a $C$-beli constraint-ekben megjelenő változók halmaza. Legyen $X, Y$ két $vars(C)$-beli változó. Azt mondjuk, hogy $X$* **közvetlenül függ** *$Y$-tól pontosan akkor, ha mindkettő ugyanabban a $c$ ($c \in C$) constraint-ban szerepel.*

**Definition 14 Constraint halmaz függőségi relációja**
*A $vars(C)$-n értelmezett* **függőségi reláció** **($dep_C$)** *a közvetlen függőségi reláció tranzitív lezártja.*

Megjegyezzük, hogy $dep_C$ egy ekvivalencia reláció a $vars(C)$-n. Az $[X]_{dep_C}$ ekvivalencia osztályt hozzárendeljük a $C$ egy olyan $C_X$ részhalmazához, amely tartalmazza az összes olyan constraint-et, amelyben van $[X]_{dep_C}$-beli változó.

**Definition 15 Bizonyítási Fa Közvetlen Függőségi Relációja**
*Legyen $T$ egy bizonyítási fa, $\alpha, \beta \in Pos(T)$. Jelölje $\sim_T$ a $Pos(T)$-n értelmezett közvetlen függőségi relációt. Ekkor $\alpha \sim_T \beta$ pontosan akkor, ha a következő feltételek valamelyike fennáll:*

1. *$\alpha$ és $\beta$ egy klóz ugyanazon constraint-jéhez tartozó pozíciók (constraint él)*
2. *$\alpha$ és $\beta$ ugyanazon csomópont egyenlőségben szerepelnek (tranzíciós él)*
3. *$\alpha$ és $\beta$ ugyanazon termhez tartozó pozíciók (funktor él)*
4. *$\alpha$ és $\beta$ pozíciók közös változót tartalmaznak (lokális él).*

A közvetlen függőségi reláció tranzitív lezártját ($\sim_T^*$) a $Pos(T)$-n értelmezett **függőségi relációnak** nevezzük. Ezért $\sim_T^*$ egy ekvivalencia reláció.

Minden egyes $T$-beli pozíció egy $P$-beli program pozícióból származik. Ezt a $\Phi : Pos(T) \rightarrow Pos(P)$ leképezéssel fogjuk jelölni.

**Definition 16 Egy program Közvetlen Függőségi Relációja**
*Legyen $P$ egy CLP program, $\alpha, \beta \in Pos(T)$. Jelölje $\sim_P$ a $Pos(P)$-n értelmezett közvetlen függőségi relációt. Ekkor $\alpha \sim_P \beta$ pontosan akkor, ha az alábbi feltételek valamelyike fennáll:*

1. *$\alpha$ és $\beta$ ugyanahhoz a constraint-hez tartoznak (constraint él)*
2. *$\alpha$ a $c$ klóz egy fejbeli atom pozíciója, $\beta$ a $d$ klóz egy olyan testbeli atom pozíciója, hogy az $\alpha$-hoz ill. $\beta$-hoz tartozó atomoknak ugyanaz a predikátum szimbóluma (tranzíciós él)*
3. *$\alpha$ és $\beta$ ugyanazon függvényhez tartozó pozíciók (funktor él)*
4. *$\alpha$ és $\beta$ ugyanabban a klózban szerepelnek, és tartalmaznak közös változót (lokális él).*

A $\sim_T$ és a $\sim_P$ relációk összehasonlításából láthatjuk, hogy valahányszor $\alpha \sim_T \beta$ $P$-nek egy $T$ levezetési fájában, mindannyiszor $\Phi(\alpha) \sim_P \Phi(\beta)$. Ezért bármely $T$ levezetési fára $(\alpha \sim_T^* \beta) \Rightarrow \Phi(\alpha) \sim_P^* \Phi(\beta)$. A $\sim_P^*$ reláció, amely a $\sim_P$ tranzitív lezártja, egy ekvivalencia reláció a $Pos(P)$-n.

A Bizonyítási Fa Függőségi Gráfot kiegészítettük az adatfolyam irányítását lehetővé tevő információval ($T_{DG} = (Pos(T), \rightarrow_T)$), hogy még pontosabb szeletelést érjünk el.

## 2.c. Annak bizonyítása, hogy a szintaktikus elemzésen alapuló szelet definíció kielégíti a szelet definíciójának szemantikus feltételeit is.

A következő tétel azt formalizálja, hogy $C_X$ eleget tesz a szelet definíció szemantikus követelményeinek is.

**Theorem 2 Constraint Halmaz egy Szelete**
*Legyen $C$ egy kielégíthető constraint halmaz, $X \in vars(C)$.*
*Ekkor $C_X$ a $C$-nek egy $X$-re vonatkozó szelete.*

A következő tétel $\sim_T^*$ és $dep_{C(T)}$ kapcsolatát írja le.

**Theorem 3 $\sim_T^*$ és $dep_{C(T)}$ viszonya**
*Legyen $\alpha$ a $T$ bizonyítási fa egy pozíciója, és $\Psi(\{\alpha\}) = \{X\}$.*
*Ekkor $\Psi([\alpha]_{\sim^*}) = [X]_{dep_{C(T)}}$.*

Ennek következményeként kapjuk a következő tételt.

**Theorem 4 Bizonyítási Fa Szelet**
*Legyen $T$ egy bizonyítási fa, és $\alpha$ a $T$ egy változó pozíciója. Ekkor $[\alpha]_{\sim^*}$ a $T$ egy szelete az $\alpha$-ra nézve.*

Tehát egy bizonyítási fa egy adott $X$ változóra vonatkozó szeletét úgy tudjuk megkapni, ha meghatározzuk az $X$-et tartalmazó ekvivalencia osztályt.
A következő eredmény azt mutatja, hogyan lehet a $\sim_P^*$ relációt a $P$ program szeletelésére használni.

**Theorem 5 Program Szelet**
*Legyen $P$ egy CLP program, és $\beta$ egy pozíciója $P$-nek. Ekkor $[\beta]_{\sim_P^*}$ a $P$ egy $\beta$-ra vonatkozó szelete.*

A 16. definíció a 5. tétellel együtt egy olyan szeletelési módszert ad, mely már független a bizonyítási fáktól, így egy statikus program szeletelési módszert kapunk CLP programokra vonatkozóan.

2.d **A szeletek méretének további finomítására két javítási módszert alkalmaztunk**.

Egyrészt változó lekötésen alapuló adatfolyam irányítási információkat vettünk figyelembe, másrészt statikus szeletelés esetén kezeltük az úgynevezett "hívási környezet" (calling context) problémát.

**Theorem 6 Bizonyítás Fa "Irányított" Szeletelése**
$\{\beta|\beta \rightarrow_{T(G)}^* \alpha\}$ *a $T$ egy $\alpha$-ra vonatkozó szelete.*

□

*3.1. A 2. Tézis Összefoglalása*

**1.a** Megadtuk a CLP programok, Bizonyítási Fák és Constraint halmazok szeletelésének szemantikus definícióját: **Definíció 9, 11, 12**.

**2.b** Adatfolyam analízisen alapuló automatikus szeletelési technikákat dolgoztunk ki:

- CLP programok statikus szeletelésére: **Definíció 16**
- Bizonyítási Fák dinamikus szeletelésére: **Definíció 15**
- Constraint Halmazok dinamikus szeletelésére: **Definíció 14**.

**2.c** Bebizonyítottuk, hogy a különböző szelet típusok automatikus megkonstruálása eleget tesz a megfelelő szemantikus szelet definícióknak is: **Tétel 5, 4, 2**.

**2.d** A szeletek méretének további finomítására két javítási módszert alkalmaztunk: (Tétel 6).

A fenti dinamikus illetve statikus szeletelési módszerek egy prototípus rendszerben kerültek implementálásra.

## 4. CONSTRAINT LOGIKAI PROGRAMOK TANULÁSA

A témához tartozó tudományos publikáció: [28].

Az **Induktív Constraint Logikai Programozás (ICLP)** a Constraint Logikai Programozás(CLP) [12,18] és az Induktív Logikai Programozás (ILP) [19,16] fúziójaként jött létre.
Az **Induktív Logikai Programozás (ILP)** olyan gépi tanulási algoritmusokat foglal magába, amelyek során első rendű teória kerül megtanulásra tanító példák és háttérinformáció alapján. Az ILP rendszerek azonban gyengék numerikus információ kezelésében. A **CLP nyelvek** a Logikai Programozás éppen ezen hiányosságát pótolják.

A Doktori Értekezésben megadtunk egy olyan **Constraint Logikai Programok tanulására vonatkozó módszert**, mely az úgynevezett "unfolding" (kicsomagolás) technikán alapul. A módszer további javítására irányult egy olyan algoritmus kidolgozása, mely a specializálási algoritmust szeleteléssel kombinálja. Az unfolding transzformációs szabály klóz eltávolítással történő kombinálását a SPECTRE rendszernél [3] alkalmazták először Logikai Programokra vonatkozóan.

A következőkben először megadjuk a tanuló algoritmus formalizálásához szükséges alapvető fogalmakat szemantikájukkal együtt, mint például a CLP programok specializációja, és az unfolding transzformáció. Ezután kimondunk két tételt, mely arra vonatkozik, hogy a definiált unfolding transzformáció megőrzi a CLP programok műveleti és logikai szemantikáját is.
Ezután kerül sor magának a (CLP_SPEC) algoritmusnak a formalizálására, illetve az algoritmus helyességét leíró tétel kimondására.
Kidolgozásra került a CLP_SPEC algoritmus egy módosított interaktív verziója is, mely a specializálást szeleteléssel és algoritmikus nyomkövetéssel kombinálja, hogy a specializálandó klózt minél hatékonyabban lehessen kiválasztani.
Egy olyan prototípus rendszer került implementálásra, mely LP és CLP programok specializálására is alkalmas, képes a specializálást szeleteléssel is kombinálni.

A következő definíciók az eredmények formalizálását segítik.

**Definition 17** $D$**-Interpretáció**
*Legyen $\langle L, D\rangle$ egy constraint domain, ahol $L$ egy elsőrendű nyelv változókból, predikátum szimbólumokból (beleértve az egyenlőséget is), és függvényszimbólumokból (beleértve a konstanst is) álló ábécé fölött. $D$ pedig az alaphalmaz (domain).*
*Az $L$ nyelv $A$ ábécéjének egy $\Im$ $D$-interpretációja a $D$ domain, és egy leképezés, amely hozzárendel*
- *minden $c \in A$ konstanshoz egy $c_\Im \in D$-t, mely ugyanaz az interpretáció, amely $c$-t interpretálja $D$-n*
- *minden $n$-aritású $f \in A$ függvényhez egy $f_\Im : D^n \to D$-t, mely szintén azon interpretációval egyezik meg, mely $f$-t interpretálja $D$-n*
- *minden $n$-aritású $p \in \Pi$ definiált predikátumhoz egy $p_\Im \subseteq D \times \cdots \times D$ relációt*
- *minden $n$-aritású $p \in \Sigma$ constraint predikátumhoz ugyanazt a $p_\Im$-t, amely $p$ interpretáltja $D$-n.*

Legyen $B_D = \{p(\widetilde{d}) \mid p \in \Pi, \widetilde{d} \in D^n\}$.
Ekkor egy $D$-interpretáció reprezentálható $B_D$ egy részhalamazaként.

A $P$ constraint logikai program **top-down operációs szemantikája** tekinthető úgy, mint az $\langle A, C, S\rangle$ elemhármasokon végzett tranzíciós műveletek sorozata, ahol $A$ atomok és constraint-ek halmaza, $C$ és $S$ pedig constraint-ek halmaza [12]. $A$ tekinthető úgy, mint a még láthatatlan atomok és constraint-ek gyűjteménye, $C$ olyan constraint-ek halmaza, amelyek aktív szerepet játszanak ("felébredtek"), $S$ pedig olyan constraint-ekből áll, melyek passzív szerepet játszanak ("alszanak").
Feltesszük, hogy a műveleti szabályok során kiválasztásra kerül a megfelelő átmenet (tranzíció) típusa és $A$ egy megfelelő eleme is.
A **kezdeti célt** $G$ a $\langle G, \emptyset, \emptyset\rangle$ állapot reprezentálja. Egy **levezetés** tranzíciók (átmenetek) sorozatának tekinthető. Egy tovább már nem módosítható állapotot **végállapotnak** nevezünk.
Egy levezetés akkor **sikeres**, ha véges és a végállapot $\langle \emptyset, C, S\rangle$ alakú.
Az átmeneti szabályok részletes leírása a Doktori Tézisben található.
A specializációs problémát a következőképpen formalizáljuk.

**Definition 18 A specializációs probléma**
*Egy constraint logikai program specializációs problémája adott pozitív ($E^+$) és negatív ($E^-$) példák halmazára nézve a következőképp definiálható:*
**Adott:** *egy $P$ Constraint Logikai Program, és kötött termek két diszjunkt halmaza ($E^+$ és $E^-$).*
**A cél:** *egy olyan $P'$ Constraint Logikai Program megtalálása (a $P$ specializáltja $E^+$-ra és $E^-$-ra nézve), hogy $M_{P'} \subseteq M_P$, $E^+ \subseteq M_{P'}$ és $M_{P'} \cap E^- = \emptyset$, ahol $M_P$ a $P$ egy $D$-modeljét jelöli.*

Feltesszük, hogy minden pozitív illetve negatív példa a $G$ célpredikátum egy kötött példánya.

# *3. Tézis*

## 3.a Az unfolding transzformáció

A következőkben formalizáljuk az unfolding transzformációt és megadunk két tételt arra vonatkozóan, hogy a definiált transzformáció megőrzi a CLP programok műveleti és logikai szemantikáját.

**Definition 19 Az unfolding transzformáció**
*Legyen $P$ egy CLP program az $R_1, \cdots, R_n$ szabályokkal úgy, hogy*
*$R_j : h_j \leftarrow b_{j_1}, \cdots, b_{j_{m_j}}, c_j$ $(j = 1, \cdots, n)$, ahol $h_j, b_{j_1}, \cdots, b_{j_{m_j}}$ definiált predikátumok, és $c_j$ jelöli az $R_j$ testében megjelenő atomi constraint-ek konjunkcióját.*
*Legyen $R : h \leftarrow b_1, \cdots, b_m, \cdots, b_k, c$ egy $P$-beli programklóz, legyen továbbá $\overline{R} = \{R_1, \cdots, R_q\}$ programklózok átnevezett példányainak halmaza oly módon, hogy minden egyes $R_i \in \overline{R}$ $(i = 1, \cdots, q)$ szabály fejének és $b_m$-nek ugyanaz a predikátum szimbóluma, valamint a számítási szabály $b_m$-et választotta ki.*

*Ekkor az unfolding alkalmazása után a kapott $P'$ program a következő:*

$$P' = Unf(P, R, b_m) = \text{argumentum egyenletek} \qquad test(R_j)$$
$$= P \setminus \{R\} \cup (\bigcup_{Rj \in \overline{R}} h \leftarrow \overbrace{(\overline{b}_m = \overline{h}_j)}, b_1, \cdots, b_{m-1}, \overbrace{b_{j_1}, \cdots, b_{j_{m_j}}, c_j}, b_{m+1}, \cdots, b_k, c),$$

*ahol $\overline{b}_m = \overline{h}_j$ egy rövidítés a $b_m$ és $h_j$ közti megfelelő argumentum egyenlőségekre vonatkozóan.*

Megjegyezzük, hogy csak definiált predikátumokra lehet alkalmazni az unfolding-ot, constraint predikátumokra nem.

A műveleti szemantika megőrzésére vonatkozó tételünk a következőképp fogalmazható meg:

**Theorem 7 Az unfolding transzformáció megőrzi a műveleti szemantikát**
*Legyen $P$ egy CLP program az $\langle L, D\rangle$ constraint domain-el és az $R_1, \cdots, R_n$ szabályokkal úgy, hogy*
*$R_j : h_j \leftarrow b_{j_1}, \cdots, b_{j_{m_j}}, c_j$ $(j = 1, \cdots, n)$, ahol $h_j, b_{j_1}, \cdots, b_{j_{m_j}}$ definiált predikátumok, és $c_j$ jelöli az $R_j$ testében megjelenő atomi constraint predikátumok konjunkcióját.*
*Legyen $\Im$ egy $D$-interpretation és jelölje $\exists_{-\widetilde{X}} Q$ a $Q$ formula egzisztenciális lezártját, kivéve az $\widetilde{X}$ változóit, melyek kvantálatlanul maradnak.*

*Minden egyes $P \cup \{G\}$ SLD-fára, ahol $G = p(\widetilde{X})$, minden $R \in P$ szabályra és minden $b_m \in body(R)$ definiált predikátumra:*

$$SS(P) = SS(P'),$$

*ahol $P' = Unf(P, R, b_m)$ és $SS(P) = \{p(\widetilde{X}) \leftarrow c \mid \langle p(\widetilde{X}), \emptyset, \emptyset\rangle \rightarrow^* \langle \emptyset, C', C''\rangle, \Im \models c \longleftrightarrow \exists_{-\widetilde{X}} C' \wedge C''\}$ tartalmazza a $p(\widetilde{X})$ célra ($\widetilde{X}$ a szabad változók halmazát jelöli) vonatkozó válasz constraint-ek halmazát.*

*Tehát az unfolding transzformáció megőrzi a CLP programok műveleti szemantikáját.*

**Theorem 8 A logikai szemantika megőrzése**
*Az unfolding transzformáció megőrzi a CLP programok logikai szemantikáját (D-szemantikáját).*

## 3.b A CLP_SPEC algoritmus

A CLP_SPEC algoritmus constraint logikai programok specializálására alkalmas olyan módszer, amely az unfolding transzformációt klóz eltávolítással kombinálja.

**Theorem 9 A CLP_SPEC algoritmus helyessége**
*A CLP_SPEC algoritmusnak $P'^{(n)}$ output-ja a $P$ $E^+$-ra és $E^-$-ra vonatkozó specializáltja, ha az algoritmus nem azért terminált, mert nem alkalmazható több unfolding lépés. Ez egyben azt is jelenti, hogy $P'^{(n)}$ teljes és konzisztens (azaz lefedi az összes pozitív példát, de nem fed le egy negatívat sem).*

## 3.c A CLP_SPEC_SLICE algoritmus

A CLP_SPEC algoritmus úgy specializál klózokat, hogy a specializálandó célpredikátumot különböző stratégiák alapján választja ki. A specializálásra kiválasztott predikátum meghatározása az algoritmus hatékonyságának tekintetében kulcsfontosságú [2]. Ha az aktuális program lefed egy negatív példát, akkor létezik legalább egy olyan klóz, amely felelős ezért a hibás lefedésért. A CLP_SPEC_SLICE algoritmus a szeletelést egy nyomkövetési technikával kombinálja, hogy megtaláljuk ezt a "felelős" klózt.

**Theorem 10 A CLP_SPEC_SLICE algoritmus helyessége**
*A CLP_SPEC algoritmusnak $P'^{(n)}$ output-ja a $P$ $E^+$-ra és $E^-$-ra vonatkozó specializáltja, ha DEB_SLICE algoritmus képes azonosítani a "felelős" klózt, és ha az algoritmus nem azért terminált, mert nem alkalmazható több unfolding lépés.*

□

# 5. ATTRIBÚTUM NYELVTANOK SZEMANTIKUS FÜGGVÉNYEINEK TANULÁSA

A témához kapcsolódó tudományos publikáció: [27].

Az **Attribútum Nyelvtanok (AG)** [29,15,1] a Környezetfüggetlen Nyelvek általánosításának tekinthetők. [14]. Az AG-ket széles körben alkalmazzák programozási nyelvek specifikálásánál és implementálásánál. Mivel az Attribútum Nyelvtanok szemantikus függvényeinek definiálása igen összetett lehet, ezért nagy gyakorlati haszna van egy olyan eszköznek, amely képes példák alapján AG-k szemantikus függvényeinek megtanulására. Ezért került sor a PAGELEARN algoritmus kidolgozására.
Az Attribútum Nyelvtanok (AG) és a Logikai Programok (LP) közti megfeleltetés alapján néhány LP-re kidolgozott tanuló algoritmus (ILP) [19,16] alkalmazható AG-kre [6]. Ha az ILP rendszerekben bevezetünk egy AG-n alapuló leíró nyelvet, akkor formalizálható egy Attribútum Nyelvtan Tanuló. A Doktori Értekezésben megadtunk egy olyan párhuzamos tanuló eljárást, amely ILP megközelítéssel képes AG-k szemantikus függvényeit tanulni (PAGELEARN). A kifejlesztett eljárás S-attribútum és L-attribútum nyelvtanok tanulására is alkalmas. A párhuzamosság segít csökkenteni az interaktív eljárás során feltett kérdések számát, így egy időben és a feltett kérdések számában is hatékonyabb algoritmust kapunk, mint a korábban kidolgozott AGLEARN [8] eljárás.
Az **Attribútum Nyelvtanokat** (AG) Knuth [13] vezette be.

**Definition 20 S-attribútum Nyelvtanok**
*Az Attribútum Nyelvtanok egy speciális osztálya az S-attribútum nyelvtanok, melyek csak szintetizált attribútumokkal dolgoznak.*

Az $S$-attribútum nyelvtanokon alkalmazott erős megszorítások miatt a gyakorlatban inkább $L$-attribútum nyelvtanokkal dolgoznak.

**Definition 21 L-attribútum Nyelvtanok**
*Egy Attribútum Nyelvtant L-attribútum nyelvtannak nevezünk pontosan akkor, ha az $X_{p,j}$ minden egyes örökölt attribútuma a $p : X_{p,0} \rightarrow X_{p,1}, \ldots, X_{p,n_p}$ szabályban csak a $\bigcup_{k \in \{1,\cdots,j-1\}} Inh(X_{p,k}) \cup Syn(X_{p,0})$ $(j = 1, \ldots, n_p)$ attribútum halmaztól függ.*

Az **AGLEARN** [8] egy olyan interaktív eljárás, amely Attribútum Nyelvtanok szemantikus függvényeinek tanulására alkalmas. Az eljárás alapötlete, hogy a tanulási problémát propozícionális formára transzformáljuk, és a hipotézist egy propozíciós tanuló algoritmus állítja elő, majd az így kapott eredményt ismét visszatranszformáljuk AG formára. Ez a módszer azon alapul, hogy szoros kapcsolat van az Attribútum Nyelvtanok és a Logikai Programok közt [6].
Az AGLEARN az Induktív Logikai Programozáshoz (ILP) hasonló koncepciót alkalmaz, de más reprezentációs nyelvet használ. A háttérinformáció és a koncepció AG formájában van reprezentálva. A példák olyan karaktersorozatok, amelyek levezethetők a cél nemterminálisból. Feltesszük, hogy a vizsgált környezetfüggetlen nyelvtan adott. Az AGLEARN feladata, hogy a szintaktikai szabályokhoz tartozó szemantikus függvényeket következtessen. A tanulás folyamata során a környezetfüggetlen nyelvtan, a háttértudás és a tanulópéldák használhatóak.
Az általunk fejlesztett párhuzamos eljárás (PAGELEARN) párhuzamosan képes tanulni szemantikus függvényeket az AGLEARN eljárás koncepcióját alkalmazva. A párhuzamosítás során kezeltük a fölmerülő úgynevezett kereszthivatkozási problémát, és megadtuk a módszer S- illetve L-attribútum nyelvtanokra kidolgozott változatát is.

## *4. Tézis*

Az általunk fejlesztett PAGELEARN eljárás több szinten is párhuzamosított:

1. A tanulandó szabály összes szemantikus függvényét egyszerre, párhuzamosan tanulja a rendszer.
2. Maga az AG kiértékelő is egy párhuzamos rendszer, így az adott példához tartozó levezetési fa maga is párhuzamosan van kezelve. Ez még több szemantikus függvény egyidejű tanulását teszi lehetővé.
3. A szemantikus függvények párhuzamos tanulása lehetővé teszi az interaktív algoritmus során feltett kérdések számának redukálását. Egyetlen kérdést sem tesz fel a rendszer abban az esetben, ha nem jelenik meg az úgynevezett kereszt hivatkozás problémája.
   Mivel a szekvenciális eljárás egyik fő problémája éppen a felhasználónak feltett kérdések túl nagy száma volt, ezért a párhuzamosítás ilyen szempontból is növeli a tanulás hatékonyságát.

## 6. A DOKTORI ÉRTEKEZÉS SZERZŐJÉNEK SAJÁT EREDMÉNYEI

**I. Logikai Programok Adatfolyam Analízise:**

Az imperatív programokra kidolgozott un. "releváns szelet" alkalmazásának ötlete Logikai Programokra Gyimóthy Tibortól származik. Ebben a témában született tudományos publikációk nagy részben Szilágyi Gyöngyi által lettek megírva, de az algoritmus kidolgozása, fejlesztése a szerzők közös munkája. Tehát a Bővített SLD-fa, Skeleton(n), az Általánosított Szelet definíció, és a szükséges definíciók tételek Szilágyi Gyöngyi által lettek formalizálva. Szilágyi Gyöngyi néhány ötlettel részt vett az implementációban, de a fejlesztést Harmath László végezte.

**II. Constraint Logikai Programok Adatfolyam Analízise:**

Az ezen témában készült tudományos publikációk nagy részben Szilágyi Gyöngyi által lettek megírva. A legtöbb eredmény is Szilágyi Gyöngyitől származik Jan Małuszyński és Gyimóthy Tibor támogatásával. Jan Małuszyńskitől származik a Constraint halmaz szemantikus szeletének definíciója, és néhány formalizmus letisztázása. Az implementálást Harmath László végezte Szilágyi Gyöngyi támogatásával.

**III. Constraint Logikai Programok Tanulása:**

A témához kapcsolódó tudományos publikációt Szilágyi Gyöngyi írta. Gyimóthy Tibor szorgalmazta az LP-re kidolgozott specializálási módszer CLP-re történő alkalmazhatóságának vizsgálatát. A módszer CLP-re történő adoptálását, a szükséges elméleti háttér kidolgozását (beleértve az unfolding transzformáció definícióját, a logikai és műveleti szemantika megőrzésére vonatkozó tételek kidolgozását is), illetve az algoritmusok formalizálását Szilágyi Gyöngyi végezte. Az implementálást egy diák végezte Szilágyi gyöngyi támogatásával.

**IV. Attribútum Nyelvtanok Szemantikus Függvényeinek Tanulása:**

Ezt a kutatást Szilágyi Gyöngy végezte. A kapcsolódó tudományos közleményt is ő írta, egy fejezet kivételével, mely a PAGE rendszert mutatja be (ez a fejezet Aggelos Thanos által lett írva). Maga a PAGE rendszer egy tetszőleges párhuzamos AG kiértékelővel helyettesíthető.

## 7. A DOKTORI ÉRTEKEZÉSHEZ KAPCSOLÓDÓ TUDOMÁNYOS PUBLIKÁCIÓK

[**1**] **Szilágyi, Gy.**, Małuszyński, J., Gyimóthy, T., 2002. Static and Dynamic Slicing of Constraint Logic Programs. Journal of Automated Software Engineering, Kluwer Academic Published, Vol. 9, No. 1, Jan 2002, pages 41-65.

[**2**] **Szilágyi, Gy.**, Gyimóthy, T., Małuszyński J., 2000. Slicing of Constraint Logic Programs. In Proceedings of the Fourth International Workshop on Automated Debugging (AADEBUG'2000), Munich, Germany, pages 176-187.

[**3**] **Szilágyi, Gy.** and Thanos, A. M., 2000. PAGELEARN: Learning Semantic Functions of Attribute Grammars in Parallel. Journal of Computing and Information Technology (C.I.T.), Vol. 8, No. 2, pages 115-131.

[**4**] **Szilágyi, Gy.**, Harmath, L. and Gyimóthy, T., 2001. Debug Slicing of Logic Programs. Acta Cybernetica, Vol. 15, No.2, pages 257-278 .

---

[**5**] Harmath, L., **Szilágyi, Gy.** and Gyimóthy, T., 2000. Debug Slicing of Logic Programs. Conference of PhD Students on Computer Sciences 2000, Hungary, pages 43-44.

[**6**] Harmath, L., **Szilágyi, Gy.**, Gyimóthy, T., Csirik, J., 1999. Dynamic Slicing of Logic Programs. In Poceedings of the Program Analysis and verification, Fenno- Ugric Symposium (FUSST'99), Tallin, Estonia, pages 101-113.

[**7**] **Szilágyi, Gy.** and Gyimóthy, T., 2003. Learning of Constraint Logic Programs by Combining Unfolding and Slicing. Submitted to AI Communications, The European Journal on Artificial Intelligence.

[**8**] **Szilágyi, Gy.**, Gyimóthy, T. and Małuszyński, J., 1998. Slicing of Constraint Logic Programs. Technical Report, Linköping University Electronic Press 1998/020,
*www.ep.liu.se/ea/cis/1998/020.*

**<u>EGYÉB TUDOMÁNYOS KÖZLEMÉNYEK:</u>**

[**9**] Juhos, I., **Szilágyi, Gy.**, Csirik, J., Szarvas, Gy., Szeles, T., Kocsis, A., Szegedi, A., 2002. Time Series Prediction Using Artificial Intelligence Methods. In Proceedings of Conference of PhD Students in Computer Science ($CS^2$ '2002), Szeged, Hungary.

[**10**] Hócza, A., **Szilágyi, Gy.** and Gyimóthy, T., 2002. LL Frame System of Learning Methods. In Proceedings of Conference of PhD Students in Computer Science ($CS^2$ '2002), Szeged, Hungary.

## References


[1] Alblas, H., 1991. Introduction to Attribute Grammars. LNCS 545, Springer Verlag.

[2] Alexin, Z., Gyimóthy, T., Boström, H., 1996. IMPUT: An Interactive Learning Tool based on Program Specialization submitted to the Intelligent Data Analysis Journal published by the Elsevier Ltd.

[3] Boström, H. and Idestam-Almquist, P., Specialization of Logic Programs by Pruning SLD-trees. Proceedings of the Fourth International Workshop on Inductive Logic Programming (ILP-94), Bad Honnef/Bon, Germany, pages 31-47.

[4] Boye, J. Paakki, J. and Małuszyński, J., 1993. Dependency-Based Groundness Analysis of Functional Logic Programs. Research Report LiTH-IDA-R93-20, Department of Computer and Information Science, Linköping University.

[5] Boye, J. Paakki, J. and Małuszyński, J., 1993. Synthesis of Directionality Information for Functional Logic Programs. In Proceedings of 3rd International Workshop on Static Analysis, LNCS 724, Springer-Verlag, pages 165-177.

[6] Deransart, P. and Małuszyński, J., 1993. A grammatical view of logic programming. The MIT Press.

[7] Gyimóthy, T. Beszédes, À. and Forgács, I., 1999. An Efficient Relevant Slicing Method for Debugging. In Proceedings of the 7th European Software Engineering Conference (ESEC'99), LNCS 1687 Springer Verlag, Toulouse, France, pages 303-322.

[8] Gyimóthy, T. and Horváth, T., 1997. Learning Semantic Functions of Attribute Grammars. Nordic Journal of Computing, Vol. 4 (1997), pages 287-302.

[9] Harmath, L., Szilágyi, Gy., Gyimóthy, T., Csirik, J., 1999. Dynamic Slicing of Logic Programs. In Proceedings of the Program Analysis and verification, Fenno- Ugric Symposium (FUSST'99), Tallin, Estonia, pages 101-113.

[10] Harmath, L., Szilágyi, Gy. and Gyimóthy, T., 2000. Debug Slicing of Logic Programs. Conference of PhD Students on Computer Sciences 2000, Hungary, pages 43-44.

[11] Horwitz, S. and Reps, T., 1992. The Use of Program Dependence Graphs in Software Engineering. In Proceedings of the Fourteenth International Conference on Software Engineering, Melbourne, Australia, pages 392-411.

[12] Jaffar, J. and Maher, M.J., 1994. Constraint logic programming: A Survey. The Journal of Logic Programming, 19/20, pages 503-582.

[13] Knuth, Donald E., 1968. Semantics of Context-Free Languages. Mathematical Systems Theory, volume 2, number 2, pages 127 -145.

[14] Knuth, Donald E., 1968. Semantics of Context-Free Languages, correction. Mathematical Systems Theory, Volume 5, Number 1, pages 95 - 96.

[15] Knuth, Donald E., 1990. The Genesis of Attribute Grammars. In Proceedings of the International Conference on Attribute Grammars and their Applications, Springer-Verlag, LNCS, 1990, Volume 461, pages 1 - 12.

[16] Lavrac, N. and Dzeroski, S., 1994. Inductive Logic Programming: Techniques and Applications. Ellis Horwood.

[17] Lloyd, J.W., 1987. Foundations of Logic Programming, Spinger Verlag.

[18] Marriott, K. and Stuckey, P.J., 1998. Programming with Constraints. An Introduction. The MIT Press.

[19] Muggleton S., 1994. Inductive Logic Programming. The ACM Press.

[20] Nilsson, U. and Małuszyński, J., 1995. Logic, Programming and Prolog. John Wiley and Sons Ltd.

[21] Pereira, L.M. and Calejo, M., 1988. A Framework for Prolog Debugging. In Proceedings of ICLP/SLP'88, pages 481-495.

[22] Shapiro, E., 1983. Algorithmic Debugging. The MIT Press.

[23] Szilágyi, Gy., Harmath, L. and Gyimóthy, T., 2001. Debug Slicing of Logic Programs. Acta Cybernetica, Vol. 15, No.2, pages 257-278.

[24] Szilágyi, Gy., Małuszyński, J., Gyimóthy, T., 2002. Static and Dynamic Slicing of Constraint Logic Programs. Journal of Automated Software Engineering, Kluwer Academic Published, Vol. 9, No. 1, Jan 2002, pages 41-65.

[25] Szilágyi, Gy., Gyimóthy, T., Małuszyński, J., 2000. Slicing of Constraint Logic Programs. In Proceedings of the Fourth International Workshop on Automated Debugging (AADEBUG'2000), Munich, Germany, pages 176-187.

[26] Szilágyi, Gy., Gyimóthy, T. and Małuszyński, J., 1998. Slicing of Constraint Logic Programs. Technical Report, Linköping University Electronic Press 1998/020, *www.ep.liu.se/ea/cis/1998/020*.

[27] Szilágyi, Gy. and Thanos, A. M., 2000. PAGELEARN: Learning Semantic Functions of Attribute Grammars in Parallel. Journal of Computing and Information Technology (C.I.T.), Vol. 8, No. 2, pages 115-131.

[28] Szilágyi, Gy. and Gyimóthy, T., 2003. Learning of Constraint Logic Programs by Combining Unfolding and Slicing. Submitted to AI Communications, The European Journal on Artificial Intelligence.

[29] Wilhelm, R., 1979. Attributierte Grammatiken. Informatik Spektrum, Volume 2, pages 123 - 130.