Válogatott Fejezetek a Logikai Programozás Világából

Windisch Zoltán
wz420@hszk.bme.hu

ASP

2004. október 11.

ASP

ASP = Active Server Pages

☺

ASP

ASP ≠ Active Server Pages
ASP = Answer Set Programming

ASP

ASP ≠ Active Server Pages
ASP = Answer Set Programming

- Gyökerek: tagadás szemantikája a logikai programozásban, keresési problémák kielégítő megoldása
- Válasz halmaz: logikai állításokból felépülő elmélet stabil modellje.
- Az ASP solver egy elmélet válasz halmazát számítja ki
- Probléma megoldás ASP-ben:

  A probléma elméletként való felírása, mely elmélet válasz halmaza (modellje) a probléma megoldását is jelenti.

ASP születése

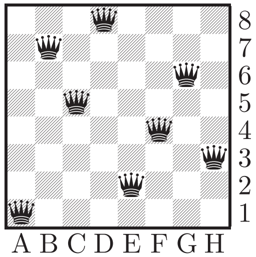
- 70': a logikai programozás felvirágzása, a tagadás megjelenése
- 80' első fele: autoepistemic logic (nem monotonikus érvelés)
- 80' második fele: logikai programok leíró logikai szemantikája
- 80' vége: stable model szemantika
- '95: smodels: stable model szemantika első hatékony implementálása
- '99: ASP mint új programozási paradigma: Vladimir Lifschitz

Az ASP és a Prolog viszonya

- A Prolog alapgondolata
  - A program: logikai állítások halmaza
  - A kérdés: egy adott állítás következik-e a programból
  - A válasz: az adott kérdést igazzá tevő behelyettesítések (egyetlen vagy összes)
- Az ASP alapgondolata
  - A program: logikai állítások halmaza (esetleg paraméterekkel)
  - A kérdés: Kielégíthető-e az adott állításhalmaz
  - A válasz: Az adott állításhalmaz modellje (egyetlen vagy összes), azaz az összes igaz elemi állítás halmaza (=válaszhalmaz)
- Mind a Prolog, mind az ASP a logikai programozáshoz tartozik
  - mindkettőben fontos szerepe van a meghiusulásos negációnak
  - más a megengedett formulák halmaza
  - ugyanaz a feladat megoldható Prologban és ASP-ben, de más megfogalmazást igényel

N - királynő

Az N-királynő-probléma célja, hogy N darab királynőt úgy helyezzünk el egy $N \times N$-es sakktáblán hogy egyáltalán ne támadják egymást. (Egy királynő támad egy vele azonos sorban, oszlopban vagy átlóban lévő bábut.)




```
queens(N, Qs, LabOpts) :-
        queens_nolab(N,Qs), labeling(LabOpts,Qs).

queens_nolab(N, Qs) :-      % A Qs lista egy biztonságos N királynő elhelyezés.
        length(Qs, N),
        domain(Qs, 1, N),
        safe(Qs).

safe([]).                   % safe(Qs): A Qs királynő-lista biztonságos.
safe([Q Qs]):-
        no_attack(Qs, Q, 1), safe(Qs).

no_attack([],_,_).          % no_attack(Qs, Q, I): A Qs lista által leírt királynők
no_attack([XXs], Y, I):-    % egyike sem támadja a Q által leírt királynőt, ahol
        no_threat(X, Y, I), % Qs a (j, j+1, ...) sorbeli királynőket írja le,
        I1 is I+1,          % Q a i. sorbeli királynőt, és I = j-i > 0.
        no_attack(Xs, Y, I1).

% Az X és Y oszlopokban I sortávolságra levő
% királynők nem támadják egymást.
% Globális korláttal megvalósítva
no_threat(X, Y, I) :-
         Y #\= X,
         Y #\= X-I,
         Y #\= X+I.
```

```
number(1..n). % =  number(1). number(2). ... number(n).

1{q(I,J):number(I)}1 :- number(J).
% (Minden J-hez) pontosan egy I van, melyre g(I,J) igaz.

:- q(I,J), q(I,J1),  % egy sorban nem lehet 2 királynő
   number(I;J;J1),J<J1.

:- q(I,J), q(I1,J1), % átlósan sem lehet ütés
   number(I;I1;J;J1),J<J1,abs(I-I1)==abs(J-J1).

hide number(I). % elrejti a megoldásból a számokat
```



```
% lparse -c n=8 queen | smodels 1

smodels version 2.28. Reading...done
Answer: 1
Stable Model: q(4,1) q(6,2) q(1,3) q(5,4)
              q(2,5) q(8,6) q(3,7) q(7,8)
True
Duration: 0.030
Number of choice points: 3
Number of wrong choices: 0
Number of atoms: 89
Number of rules: 552
Number of picked atoms: 240
Number of forced atoms: 5
Number of truth assignments: 2535
Size of searchspace (removed): 64 (0)
```



```
{ q(3,1), q(2,1), q(1,1) } .
{ q(3,2), q(2,2), q(1,2) } .
{ q(3,3), q(2,3), q(1,3) } .
_false :- q(1,1), q(1,2).
_false :- q(2,1), q(2,2).
_false :- q(3,1), q(3,2).
_false :- q(1,1), q(1,3).
_false :- q(2,1), q(2,3).
_false :- q(3,1), q(3,3).
_false :- q(1,2), q(1,3).
_false :- q(2,2), q(2,3).
_false :- q(3,2), q(3,3).
_false :- q(2,1), q(1,2).
_false :- q(1,1), q(2,2).
_false :- q(3,1), q(2,2).
_false :- q(2,1), q(3,2).
_false :- q(3,1), q(1,3).
_false :- q(1,1), q(3,3).
_false :- q(2,2), q(1,3).
_false :- q(1,2), q(2,3).
_false :- q(3,2), q(2,3). _false :- q(2,2), q(3,3).
_false :- __int2(_,1).
__int2(_,1) :- 3 { not q(3,1), not q(2,1), not q(1,1) }.
_false :- __int2(_,2).
__int2(_,2) :- 3 { not q(3,2), not q(2,2), not q(1,2) }.
_false :- __int2(_,3).
__int2(_,3) :- 3 { not q(3,3), not q(2,3), not q(1,3) }.
_false :- __int1(_,1).
__int1(_,1) :- 2 {  q(3,1),  q(2,1),  q(1,1) }.
_false :- __int1(_,2).
__int1(_,2) :- 2 {  q(3,2),  q(2,2),  q(1,2) }.
_false :- __int1(_,3).
__int1(_,3) :- 2 {  q(3,3),  q(2,3),  q(1,3) }.
compute 1 { not _false }.
```


Az ASP program felépítése

- program
- szabályok
- formulák
- literálok
- atomi állítások

Formulák építőelemei

- *atom* (atomi állítás, $\neq$ „Prolog atom”)
- klasszikus tagadás ($\neg$)
- *literál*: A vagy $\neg A$ alakú kifejezés, ahol A *atom*
- tagadás (not), azaz meghiusulásos negáció („negation as failure”)

  $\neg \neq not\ \ (\backslash+)$

  Pl.: ¬zöld $\leftarrow$ not fotoszintetizál
- konjunkció (,)
- diszjunkció (;)
- $\top$ $\equiv$ true $F_1, F_2, \ldots, F_m \quad m = 0$ esetén
- $\bot$ $\equiv$ false $F_1; F_2; F_3; \ldots; F_m \quad m = 0$ esetén

Tagadás, de hogyan?

- Az alap stable model szemantika csak a meghiusulásos tagadást tartalmazza
- *kiterjesztett logikai program*nak nevezzük a klasszikus negációt tartalmazó programokat.

  Ez utóbbi egy sokkal erősebb tagadás. Nézzük pl. az alábbiak különbségét a sinen való áthaladásra

  biztonságos $\leftarrow$ not *vonat*

  biztonságos $\leftarrow$ ¬*vonat*
- Valós tagadás esetében gondot jelenthet a nem konzisztens válasz halmaz, de ez utóbbi szűrhető.

  (Tehát a és $\neg\, a$ egyszerre jelenhet meg egy válaszhalmazban)
- Továbbiakban, ahol nem hangsúlyozom külön a jelenlétét, ott eltekintek ettől a valós tagadástól

Szabályok építőelemei

- *szabály elem* = L vagy $not L$, ahol L egy literál.
- *Fej* $\leftarrow$ *Törzs* *Fej* és *Törzs* szabály elemek véges halmaza
- Ha

$$\begin{aligned} \textit{Fej} &= \{L_1, \ldots, L_k, not\ L_{k+1}, \ldots, not\ L_l\} \\ \textit{Törzs} &= \{L_{l+1}, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n\}, \ \ (n \geq m \geq l \geq k \geq 0) \end{aligned}$$

akkor

$$L_1, \ldots, L_k, not\ L_{k+1}, \ldots, not\ L_l \leftarrow L_{l+1}, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$$

- Üres fej esetén *korlát* szabályról beszélünk: tagadjuk a törzs igazságát!
- Ha a fej kardinalitása ≥ 1, *diszjunktív* szabályról beszélünk
- $\leftarrow$ elhagyható, ha a törzs üres $(n = m = l)$

Program

Program $\equiv$ szabályok véges halmaza

Különbség a „megszokott” logikai programokhoz képest:

- Valójában ítéletkalkulusbeli állításaink vannak
  - a megadott sematikus szabályokból a *grounding* folyamán küszöböljük ki a változókat
- A „hagyományos” programoknál általánosabb szabály definíció
  - Bármely L_i szabály elem tartalmazhat valódi negálást ($\neg$)

    (hagyományosan csak „negálás meghiusulás esetén” van)
  - A fej akárhány szabály elemből állhat, és üres is lehet.

    (hagyományos esetben ez csak egy atomból állhat)
  - A „meghiusulásos negáció” a fejben is előfordulhat.

    (hagyományosan csak a törzsben)

Bevezetés a válaszhalmaz fogalmába

- Tegyük fel, hogy a programban nincs not és a *Fej* egy literálból áll.

$L_0 \leftarrow L_1, \ldots, L_m$[a], ahol $m \geq 0$ és $L_0, \ldots, L_m$ literálok

- Π a program, X formulák konzisztens halmaza
- X-et *zártnak* mondjuk Π felett[b], ha Π minden szabályára igaz, hogy *Fej* $\cap X \neq 0$ ha *Törzs* $\subseteq X$.

  pl. minden ilyen tipusu program kielégíthető a szabályok L_0 fejeinek halmazával
- X-et Π válaszhalmazának[c] mondjuk, ha X minimális[d] a Π felett zárt halmazok közül

[a] $l = k$ és $n = m$ minden szabályban
[b] azaz X kielégíti a Π programot
[c] stabil modell
[d] A be foglalt halmazokra vonatkoztatva

- Pl.: adott program

$$p$$
$$r \leftarrow p, q$$

  amit kielégít $\{p\}$, $\{p, r\}$, $\{p, q, r\}$, de csak egy válaszhalmaza van: $\{p\}$.
- vagy adott program

$$p; q$$
$$\neg r \leftarrow p$$

  2 válaszhalmazzal rendelkezik:

$$\{p, \neg r\}$$
$$\{q\}$$

„Negation as failure” a törzsben

- Π egy tetszőleges program, szabályainak alakja:

$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$, ahol $n \geq m \geq 0$ és $L_0, \ldots, L_n$ atomok

- X formulák konzisztens halmaza
- Π X-en értelmezett Π^X redukáltja: $L_0 \leftarrow L_1, \ldots, L_m$ alakú szabályok halmaza, Π azon szabályaira, amelyekre $L_{m+1} \notin X, \ldots, L_n \notin X$.

Redukált előállítása

1. Π minden szabálya törlendő, amelyik törzsében $not L_x$ szerepel, ha $L_x \in X$
2. A maradék szabályok törzséből minden negált formula törlendő.

- Π^X már egy definit[a] program $LM(\Pi^X)$ válaszhalmazzal. (LM=Least Model)
- X-et Π válaszhalmazának mondjuk, ha $X \equiv LM(\Pi^X)$ (fixpont)

- A korábbi definíció általánosítása a mostani

Pl.: Ha Π:

$$\begin{aligned} p &\leftarrow not\ q \\ q &\leftarrow not\ r \\ r &\leftarrow not\ s \end{aligned}$$

és $X = \{p, r\}$, akkor a Π^X redukált csak a p és r szabályt tartalmazza. Tehát, X válaszhalmaza Π-nek.[b]

[a] tagadás mentes
[b] Általános Prolog program csak a p vagy r kérdésre mondana yes-t.

$X \models F$ kielégíthetőségi reláció[a] rekurzív definíciója

- egy A atomra[b], $X \models A$ ha $A \in X$
- $X \models \top$
- $X \not\models \bot$
- $X \models (F, G)$ ha $X \models F$ és $X \models G$
- $X \models (F; G)$ ha $X \models F$ vagy $X \models G$
- $X \models not\ F$ ha $X \not\models F$

X kielégít egy Π programot ($X \models \Pi$),
ha minden Π beli $F \leftarrow G$ szabályra $X \models F$ ha $X \models G$.

[a] X atomok halmaza és F formula között
[b] $\neg A$ esetén $X \models \neg A$ ha $\neg A \in X$

A cél: not kiküszöbölése (*definit program* létrehozása)

Az F formula X-re vonatkoztatott F^X lezártjának rekurzív definíciója

- Ha F atom, akkor $F^X = F$ (F lehet litarál is...)
- $(F, G)^X = F^X, G^X$
- $(F; G)^X = F^X; G^X$
- $(not\ F)^X = \begin{cases} \bot, \text{ha } X \models F \\ \top, \text{egyébként} \end{cases}$

Π program X-re vonatkoztatott Π^X redukáltján minden Π-beli $F \leftarrow G$ szabályból képzett

$$F^X \leftarrow G^X$$

szabályok halmazát értjük.

Válaszhalmaz

Egy tetszőleges Π program válaszhalmaza X, hogy ha X minimális azon (atomokat tartalmazó) halmazok közül, melyek kielégítik a Π^X redukáltat.

X tehát válaszhalmaza Π-nek, ha kielégíti Π^X-t, de semelyik valós részhalmaza nem teszi ezt.

Ez a def. az előző definíció további általánosítása.

PL.:

$$p; not\ p \qquad \text{AS: } \emptyset \text{ és } \{p\}$$

$$p \leftarrow not\ not\ p \qquad \text{AS: } \emptyset \text{ és } \{p\}$$

$$\left.\begin{array}{l} p; not\ p \leftarrow q, r \\ q \leftarrow not\ r \\ r \leftarrow not\ p \end{array}\right\} \qquad \text{AS: } \{r\}$$

- *choice expression* avagy választási lehetőség:

$$(F_1; not\ F_1), \ldots, (F_n; not\ F_n) \qquad \equiv \qquad \{F_1, \ldots, F_n\}^c$$

- *constraint*ről avagy korlátról beszélünk a fejként $\bot$-ot tartalmazó szabályok esetében, ez ilyenkor elhagyható:

$$\bot \leftarrow F \qquad \equiv \qquad \leftarrow F$$

- Tetszőleges atomok X halmaza $\Pi \cup \{\leftarrow F\}$ válaszhalmazának tekinthető a.cs.a ha X egy válaszhalmaza Π-nek és nem elégíti ki F-et. (bármely Π programra és F formulára)

- Bármely $F_1, \ldots, F_n$ $(n \geq 0)$ formula és tetszőleges nemnegatív l („lower bound”) egész esetén

$$l \leq F_1, \ldots, F_n$$

alatt az

$$F_{i_1}, \ldots, F_{i_l} \; (1 \leq i_1 \leq \cdots \leq i_l \leq n)$$

kifejezés minden konjukciójának diszjunkcióját értjük.

  - PL.: $2 \leq \{p, q, r\}$ írható a $(p, q); (p, r); (q, r)$ helyébe

- Ugyanígy u („upper bound”) nemnegatív egészre

$$\{F_1, \ldots, F_n\} \leq u$$

alatt a

$$not(u + 1 \leq \{F_1, \ldots, F_n\})$$

kifejezést értjük.

- $(l \leq \{F_1, \ldots, F_n\}), (\{F_1, \ldots, F_n\} \leq u) \quad \equiv \quad l \leq \{F_1, \ldots, F_n\} \leq u$

- $l \leq \{F_1, \ldots, F_n\}^c \quad \equiv \quad \{F_1, \ldots, F_n\}^c, l \leq \{F_1, \ldots, F_n\}^c$
- $\{F_1, \ldots, F_n\}^c \leq u \quad \equiv \quad \{F_1, \ldots, F_n\}^c, \{F_1, \ldots, F_n\}^c \leq u$
- $l \leq \{F_1, \ldots, F_n\}^c \leq u \quad \equiv \quad \{F_1, \ldots, F_n\}^c, l \leq \{F_1, \ldots, F_n\}^c \leq u$

`http://www.tcs.hut.fi/Software/smodels/`

ASP megoldók

- A szabályok leírása korlátozottab mint az eredeti logikai program szintakszisban, ugyanakkor sok használható speciális szabály is létezik.
- Szabály feje lehet
  - atom ($\neg$ atom, $(-)$) és ezek diszjunciója ($|$)
  - üres (ez felel meg $\perp$-nak.)
  - kardinális kifejezés, amennyiben az F_i formulák literálok.

    Ilyenkor a ^{c} és $\leq$ jeleket elhagyjuk. Pl. `1{p,q}` áll $1 \leq \{p, q\}^c$ helyett
- $\leftarrow$ jelölése `:-`
- A törzs üres lehet, vagy az alábbiak konjukciója
  - atomok $\neg$ atomok
  - negált atomok (not)
  - kardinális kifejezések

- Vannak speciális elemek
  - domain generátor pl. `number(1..5)` $\equiv$ `number(1). number(2). ...` vagy pl.: `{a(I):number(I)}` $\equiv$ `{a(1),a(2),...,a(5)}`
  - azonos kifejezések egybe írása: `a(1;2;3)` $\equiv$ `a(1),a(2),a(3)`
  - Kardinális kifejezéseket súlyokra is alkalmazhatunk, pl.:

    `5[a=2,b=3,c=2,d=1]6.` szabály válaszhalmaza $\{d, c, a\}$, $\{b, a\}$, $\{d, b, a\}$, $\{c, b\}$ és $\{d, c, b\}$
  - Deklarációk, melyeket # vezet be, bár ez jelenleg még el is hagyható
    - `hide` elrejti a felesleges atomokat a végeredményből
    - `show` bizonyos atomokat megmutat.
    - `constant` kifejezés értékét használhatjuk konstansnak.
    - `domain` domain megkötést tehetünk
    - `external` korábban ground alakra hozott literál használható
    - `function` numerikus függvény deklaráció
    - `option` opció beállítás (pl. `-true-negation`)
    - `weight` súly hozzárendelés
  - standard numerikus c fügvények meghívása pl. `abs`
  - *compute* és *optimie* (pl. `maximize [p=2,q=3,r=2]`) kifejezések

Pl.:

```
p :- not q.
q :- not p.
r :- p.
r :- q.
```

Legyen p1.3, ezután a program hívás:

```
lparse p1.3 | smodels 0
```

```
smodels version 2.28.Reading... Done.
Answer: 1
Stable Model: r p
Answer: 2
Stable Model: r q
False
Duration: 0.000
```



```
Number of choice points: 1
Number of wrong choices: 1
Number of atoms: 3
Number of rules: 4
Number of picked atoms: 4
Number of forced atoms: 1
Number of truth assignments: 10
Size of searchspace (removed): 2 (0)
```


Programozási elv

- Az ASP probléma megoldásra úgy alkalmazható, hogy írunk egy logikai programot, aminek a válasz halmazai megfelelnek a megoldásoknak és utána ASP megoldó segítségével konkrét megoldást keresünk.
- Az ASP alap ötelete a *„generál-és-tesztel”* elv.
  1. Olyan szabályokat írunk amelyek feje számtalan választási lehetőséget tartalmaz

     Ezek rejtik a „*lehetséges megoldás*okat”
  2. Teszt szabályokkal kiszűrjük a hamis megoldásokat.
- A tesztelő szabályok használhatnak segéd atomokat, melyeket a generáló szabályok hoznak létre, ez a *„definiáló”* rész. (pl. él tranzitív lezártja = útvonal)

Ki a gyilkos?

- Viktort meggyilkolták, 3 gyanusított van: András, Bálint és Csárli.
- András:
  - Nem én voltam
  - Bálint volt az áldozat barátja
  - Csárli utálta Viktort
- Bálint:
  - A gyilkosság éjszakáján a városon kívül voltam
  - Úgy hiszem, nem is ismertem az áldozatot.
- Csárli:
  - Ártatlan vagyok!
  - A gyilkosság előtti estén Andrással és Bálinttal láttam az áldozatot.
- Feltételezhetjük, hogy a gyilkoson kívül mindenki igazat mond.

- **Jelmagyarázat**
  - m = murderer, gyilkos
  - i = innocent, ártatlan
  - ht = hate, utálja
  - f = firend, barátja
  - o = outside, a városon kívül
  - nk = not knows, nem ismeri
  - t = together, együtt
  - h = holds, tartható azaz igaz állítás

- 4 emberünk van: `person(a), person(b), person(c)` és szegény `person(v)`
- András:
  - `says(a,i(a)).`
  - `says(a,ht(c,v)).` Csárli utálta Viktort
  - `says(a,f(b,v)).` Bálint és Viktor barátok
- Bálint:
  - `says(b,o(b)).` A városon kívül volt.
  - `says(b,nk(b,v)).` Nem ismeri Viktort.
- Csárli:
  - `says(c,i(c)).` Ártatlan.
  - `says(c,t(a,v)).` Andrást látta Viktorral
  - `says(c,t(b,v)).` Bálintot látta Viktorral

- A gyilkos kivételével mindenki igazat mond:

```
h(F) :- says(P,F), not m(P).
```

- Általános háttér tudást kell a rendszerbe vinni:

1. Az „együtt lét” szimmetrikus és tranzitív.

```
h(t(A,B)) :- person(A),person(B), h(t(B,A)).
h(t(A,B)) :- person(A),person(B),person(C),
h(t(A,C)), h(t(C,B)).
```

2. A „barát” szimmetrikus fogalom

```
h(f(A,B)) :- person(A),person(B), h(f(B,A)).
```

3. A gyilkos nem lehet ártatlan

```
:- h(i(P)),m(P), person(P).
```

4. Nem lehetsz olyannal együtt, aki a városon kívül van

```
:- person(P), person(P1), h(o(P)),h(t(P,P1)).
```

5. A barátok ismerik egymást

```
:- person(A), person(B), h(f(A,B)),h(nk(A,B)).
```


6. Városon kívüli ember nem lehet gyilkos.

```
:- h(o(P)), m(P), person(P).
```

7. Pontosan 1 gyilkos van

```
1{m(a),m(b),m(c)}1.
```

- A megjelenítést szabályozó `SMODELS` direkítvák:

```
hide.
show m(F).
```

```
% lparse murder | smodels 0

smodels version 2.28. Reading...done
Answer: 1
Stable Model: m(b)
False
Duration: 0.000
Number of choice points: 0
Number of wrong choices: 0
Number of atoms: 76
Number of rules: 161
Number of picked atoms: 2
Number of forced atoms: 1
Number of truth assignments: 58
Size of searchspace (removed): 0 (0)
```