Mesterséges intelligencia-orientált programozási nyelvek

- A speciális programozási nyelvek kifejlõdését egy-egy **szakterület sajátos elvárásai**, a szakterület feladatait jellemzõ közös tulajdonságok motiválják.
- A mesterséges intelligencia kutatások is kiérleltek több olyan programozási nyelvet, ill. keretrendszert, amelyben a tématerület alkalmazásait könnyebb elkészíteni, hiszen a fejlesztõi környezetek már **fél megoldást** jelentõ eszközöket nyújtanak.
- A speciális segédeszközöktõl az általános felhasználhatóság igényével fellépõ programnyelvekig terjed a kínálat.
- Az MI nyelvek a szimbólumkezelés eszközei.
- Az utóbbi idõben a klasszikus MI nyelvek mellett elõtérbe kerül a C++ is.

Forrás: Yoshiaki Shirai - Jun-ichi Tsujii: Mesterséges intelligencia Alapelvek és alklamazások NOVOTRADE Rt. 1987.

dr.Dudás László

Klasszikus mesterséges intelligencia nyelvek

- Legismertebb és legrégebbi a **LISP** (*John Mc Carthy*,1957).
- Számtalan LISP dialektus van, a legismertebbek: a Common LISP, az INTERLISP, a MacLISP, valamint a Standard LISP. Mivel a LISP-re nincs szabvány, ezek eltérései az alkalmazói programok hordozhatóságát megnehezítik.
- Erõs hatást gyakorolt a késõbbi nyelvekre.
- **QA1, QA2, QA3** programnyelvek, 1960-as évek. Általános problémamegoldó rendszerek, illetve **tételbizonyítás** céljára. **Szimbolikus logikát, rezolúciót** alkalmaztak.
- **QA4,** 1968. A PLANNER programozási nyelv hatása érezhetõ rajta. Szimbolikus reprezentáció mellett procedurális reprezentációra is képes volt.

dr.Dudás László

Klasszikus mesterséges intelligencia nyelvek ..


1.

- A **PLANNER** programozási környezetet *C.Hewitt*, a MIT hallgatója dolgozta ki 1968-ban.
- A PLANNER-nél már megjelent a deklaratív nyelvek fõ jellemvonása: csak a problémakört deklaráló, objektumokra és eljárásokra vonatkozó **tudást**, valamint a **megoldandó problémát** kellett megadni, a probléma megválaszolásához vezetõ utat, a szükséges tudás- és eljáráselemeket az út bejárásához maga a rendszer keresi meg és használja fel.
- Az ábrázolt kétféle tudásforma a PLANNER-nél fizikailag is elkülönült: a deklaratív tudást a **tényadatbázis**ban, a procedurális tudást a **procedure** (eljárás)-**adatbázis**ban tárolta.

1. http://www.ai.mit.edu/people/hewitt/hewitt.html

Klasszikus mesterséges intelligencia nyelvek ..

- A PLANNER elõnyei:
  - **Mintaillesztés:** ha adott az atomok változókat tartalmazó listája, akkor az illeszkedõ tények megkeresése, és ezzel a változók értékeinek meghatározása automatikus.
  - **Az eljárásokat a minták hívják.** Egy adott célhoz illeszkedõ eljárás hívását egyszerûen a cél megadása váltja ki automatikusan, anélkül, hogy az eljárás nevének meghatározása szükséges lenne.
  - **A keresés automatikus.** A PLANNER az adatbázisokból alkalmas tényeket és eljárásokat választ ki a cél eléréséhez.
  - **Visszalépés**: a kiválasztott tételek nem feltétlenül helyesek. Ha a választás a keresés sikertelenségéhez vezet, a PLANNER visszatér a közvetlenül megelõzõ döntési ponthoz, és újraválaszt. (Depth first)
  - **Dinamikus tényadatbázis**: automatikusan is bõvül a kikövetkeztetett tényekkel, tények törölhetõk.

Klasszikus mesterséges intelligencia nyelvek ..

- **CONNIVER** programnyelv, *G.J.Sussman*, MIT kutató, 1972.
  - A CONNIVER a PLANNER kiterjesztett változata, használja a tényadatbázist és az eljárás-adatbázist, valamint a minta alapján történõ keresést.
  - **Újdonsága** abban volt, hogy **elhagyta az automatikus keresést**, és a problémamegoldás vezérlését a programozóra bízta. Ezáltal a keresés **hatásfoka** megnõtt.
  - További beépített függvények.
- **POP-x** programok: európai fejlesztés.
  - Legújabb verziójuk, a POP-11 négy MI nyelvet integrál egy fejlesztõi környezetbe, közte: LISP, PROLOG, **POPLOG**. A nyelvek közül egy feladaton belül is szabadon választhatunk.
  - A POPLOG nyelv a POP-11 készítõinek teljesen saját fejlesztése.
  - Erõteljes grafikai lehetõségek, többféle hardver- platform.
  - Interpreteres üzemmód, inkrementális fordítás.

A PROLOG

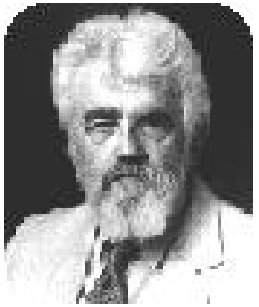

1.


1.

- **PROLOG** (Programming in Logic, Logika alapú programozás) *Alain Colmerauer, Bob Kowalski,* 1970.
- **Predikátum konstanson alapul**ó általános célú programnyelv, amely azonban legelõnyösebben logikai problémák megoldására alkalmazható.
- A PROLOG létrehozására kihatott az a tapasztalat, amelyet a korábbi MI programozási nyelvekkel szereztek: minél több a rendszerbe épített lehetõség, a rendszer annál inkább lelassul.
- **Deklaratív** programnyelv: elegendõ megadnunk csak a problématerületre vonatkozó tényeket, szabályokat és a megválaszolandó kérdést, ill. minõsítendõ állítást, és a **program generálja a megoldáshoz vezetõ lépéseket** és szolgáltatja a választ.

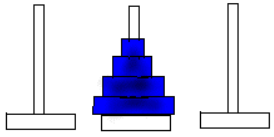
1. http://www.york.ac.uk/depts/lang/webstuff/people/sjh/L333intro_to_prolog/creators.html

Bevezetés a LISP programozási nyelvbe



John Mc Carthy, 1957, MIT

- Mesterséges intelligencia feladatok megoldására elõnyös.
- List Processor - listafeldolgozó, a listák kulcsszerepben.
- Funkcionális nyelv - a mûködést függvények kiértékelése jelenti, utasítások nincsenek.
- Rendkívül homogén felépítés, tömör szintakszis: atom, lista, függvény.
- Nagy tárigény - oka a rekurzív függvényhívások sokasága.
- A procedurális nyelvekbõl ismert ciklus- és elágazásszervezést is függvényekkel végezhetjük. A ciklusszervezés a rekurzió miatt háttérbe szorul.
- A feladatok deklaratív leírási módját támogatja.
- Alapvetõen interaktív, interpreteres használat, de létezik hozzá fordító is.
- Hardveres implementációja is van, gyors.



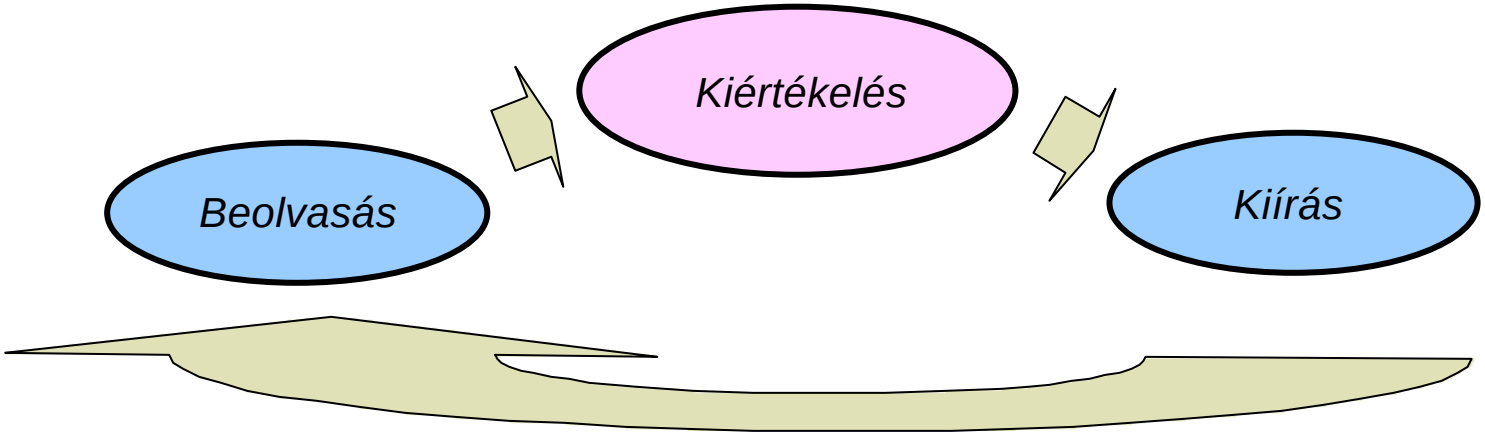
dr.Dudás László

A LISP programozási nyelv ..

- A LISP interpreter, értelmezõ

  LISP programunkat, amely nem más, mint egy lista, a listafeldolgozó dolgozza fel. Ez egy parancssoros értelmezõ, amely akár egy atomot, akár egy lista alakban adott összetettebb kifejezést azonnal feldolgoz és megadja az eredményt: egy atomot, vagy egy listát.

- Végtelen ciklusban keringve, a parancs prompt-tal mindig kiértékelendõ listára vár.



A LISP programozási nyelv építõelemei

- **Atomok** - más nyelvek azonosítóinak, konstansainak felelnek meg
  - **Szimbolikus atomok:** betûvel kezdõdõ, tetszõleges hosszúságú betû-szám kombinációval folytatódó nevek, szimbólumok. Mindig van értékük is. Más nyelvek változóit, függvényneveit, felsorolt típusú értékeit helyettesítik. Típusuk nincs.
    Pl.: VALT LISTAERTEKU SKALAR U11 MEG-VAN
  - **Numerikus atomok:** más nyelvek számkonstansainak felelnek meg, de exponenciális alak nincs, ugyanakkor hosszuk tetszõleges.
    Pl.: 11 -22 +33.44 -555.666 12345678901234567890123456789O

- **Listák** ( ) zárójelek között szóközzel elválasztva felsorolt atomok
  - **Üres lista:** ( ), szimbólum alakja a NIL.
  - A listák elemei listák is lehetnek tetszõleges mélységû beágyazottsággal.
    Pl.: (ALFA BETA GAMMA) (+12 VALT -24.25 SZO)
    (HAPCI MORGO TUDOR (DOLGOZO-TORPEK))
    ((((1 A) 2) 3) B)

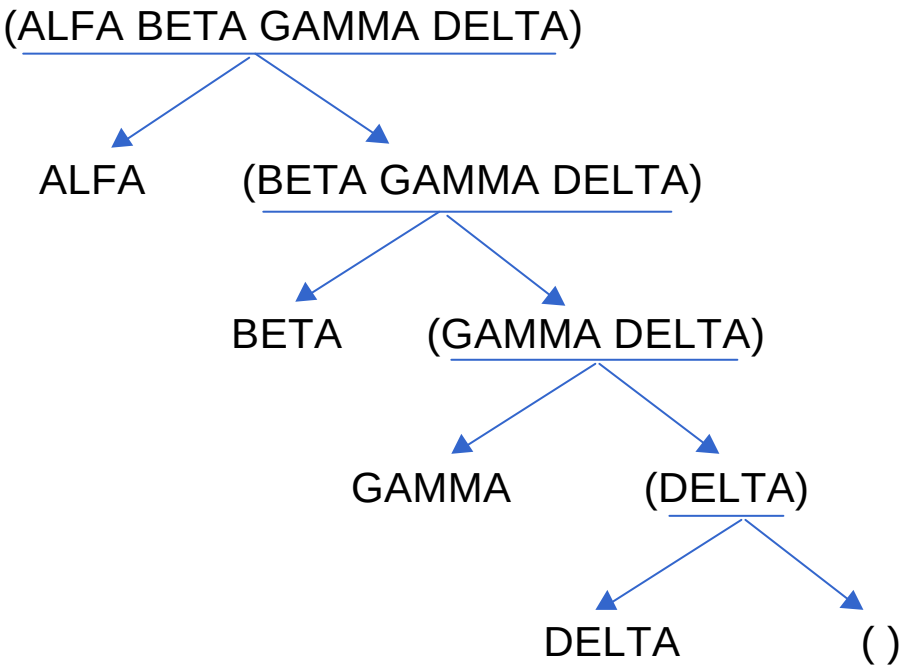
  - **Listák szerepe:** a függvények definíciója egy lista, a függvényhívás egy lista, az adatok felsorolása egy lista.

A LISP programozási nyelv építõelemei ..

- **A listák szerkezete:**

  (fej faroklista)

  A fej az elsõ elem, de mivel bármely elem bármilyen típusú lehet, a fej is lehet akár lista is.
  Egy homogén, csak elemekbõl álló lista tagolódása az általában rekurzív feldolgozás során:



dr.Dudás László

A LISP programozási nyelv építõelemei ..

- **A függvénykifejezés** a függvény programbeli meghívására szolgál. A függvénykifejezésben megadott függvény lehet beépített, vagy a programban elõzetesen a programozó által megadott.
  - A függvénykifejezés egy lista, melynek elsõ eleme a függvény neve, a többi a függvény aktuális paramétere. Létezik paraméternélküli és tetszõleges számú paraméterrel hívható függvény is.

    (*függvénynév paraméter1 paraméter2 … paramétern*)

  - Pl.: (PRINT EGY-ARGUMENTUMA-LEHET)
    (TIMES TENYEZO1 TENYEZO2 TENYEZO3 TENYEZO4)
  - A függvénynév is lehet egy másik függvény eredményeként adott.
  - A függvény visszaadott értéke lehet egy atom, vagy egy lista.

A LISP programozási nyelv építõelemei ..

- **Beépített függvények**

  - QUOTE - mivel egy függvényhívás formailag megegyezik egy szimbólummal kezdõdõ listával és a függvényhívás az alapértelmezett, ha el akarjuk kerülni, hogy egy listát függvénynek nézzen a rendszer, akkor a QUOTE függvénnyel ezt elérhetjük, mert a függvényhívásból listát csinál. Mivel nagyon gyakori ez az igény, helyettesíthetjük az aposztróffal: ‘.
    Pl.: * (QUOTE (ELEM1 ELEM2))
    (ELEM1 ELEM2)
    * ‘(ELEM1 ELEM2)
    (ELEM1 ELEM2)

  - CAR - a lista fejét adja:
    Pl.: * (CAR ‘(ELEM1 ELEM2 ELEM3))
    ELEM1
    * (CAR (1 2 3 4 5 6))
    1
    * (CAR ‘((ALFA BETA) GAMMA DELTA))
    (ALFA BETA)

A LISP programozási nyelv építõelemei ..

- **Beépített függvények ..**
  - CDR - A lista faroklistáját adja.
    Pl.: * (CDR ‘(A B C D))
    (B C D)
  - CONS - A lista elejéhez csatol egy újabb elemet.
    Pl.: * (CONS ‘BOLHA ‘(PIAC ECSERI))
    (BOLHA PIAC ECSERI)
    * (CONS ‘(ELEM1 ELEM2) ‘(ELEM3 ELEM4 ELEM5))
    ((ELEM1 ELEM2) ELEM3 ELEM4 ELEM5) ; a fej egy lista
  - APPEND - Két listát egyesít.
    Pl.: * (APPEND ‘(TALPRA MAGYAR) ‘(HI A HAZA))
    (TALPRA MAGYAR HI A HAZA)
  - LENGTH - A lista elemeinek számát adja.
    Pl.: * (LENGTH ‘(A B C))
    3
    * (LENGTH ‘(A (B C)))
    2

A LISP programozási nyelv építõelemei ..

- **Beépített függvények ..**

  - NTH - A lista n. elemét adja.
    Pl.: * (NTH 3 '(A B C D))
    C
    * (NTH 5 '(A B C D))
    NIL
    * (NTH 2 '(A (B C D)))
    (B C D)

  - MEMBER - A megadott elem elsõ elõfordulásával kezdõdõ részlistát adja.
    Pl.: * (MEMBER 'JANCSI '(PAPRIKA JANCSI MOSOGAT))
    (JANCSI MOSOGAT)
    * (MEMBER 'KENNEDI '(JOHNSON CLINTON BUSH NIXON))
    NIL
    * (MEMBER '(TUZROL PATTANT) '(CSINOS (TUZROL PATTANT)
    LEANYZO))
    ((TUZROL PATTANT) LEANYZO)
    * (MEMBER 'PATTANT '(CSINOS (TUZROL PATTANT)))
    NIL

A LISP programozási nyelv építõelemei ..

- **Beépített függvények ..**

  - SUBST - Magadott listaelem cseréje megadott elemmel.
    Pl.: * (SUBST ‘BAJNOKSAG ‘KUPA ‘(LABDA RUGO BAJNOKSAG))
    (LABDA RUGO KUPA)
    * (SUBST ‘AROL ‘BRE ‘(AROL (AROL) AROL))
    (BRE (AROL) BRE)
    * (SUBST ‘E ‘(L1 L2) ‘(A C D E F))
    (A C D (L1 L2) F)
    * (SUBST ‘(L1 L2) ‘E ‘(A C D (L1 L2) F))
    (A C D E F)

  - LIST - A felsorolt elemeket listává egyesíti.
    Pl.: * (LIST 1 2 3 4)
    (1 2 3 4)
    * (LIST ‘ABC ‘DEF ‘GHI)
    (ABC DEF GHI)
    * (LIST ‘ABC ‘(DEF GHI))
    (ABC (DEF GHI))

A LISP programozási nyelv építõelemei ..

- **Beépített függvények ..**
  - SETQQ - Szimbolikus atomhoz értéket rendel. A QUOTE implicite benne van, azaz nem akar kiértékelni. A hozzárendelt értéket kiírja.
    Pl.: * (SETQQ VALT1 12.4)
    12.4
    * VALT1
    12.4
    * (SETQQ LISTA (CAR (12 24)) ; nem értékel ki
    (CAR (12 24))
  - SETQ - Szimbolikus atomhoz értéket rendel, a második argumentumot elõbb kiértékeli.
    Pl.: * (SETQ LISTA (CAR (12 24))
    12
    * LISTA
    12
  - SET - A szimbolikus atomhoz értéket rendel, elõtte kiértékeli mindkét paramétert.
    Pl.: * (SET (CAR '(ELEM1 ELEM2)) (NTH 2 (115 335 555))
    335

A LISP programozási nyelv építõelemei ..

- **Beépített függvények ..**
  - PLUS, vagy + - A numerikus argumentumok összegét adja.
    Pl.: * (PLUS 1 2 3)
    6
    * (SETQQ SZAM 21)
    21
    * (PLUS SZAM 44)
    65
  - DIFFERENCE, vagy - - Két numerikus paramétere különbségét adja.
    Pl.: * (DIFFERENCE 55.5 23.4)
    32.1
  - TIMES, vagy * - A numerikus argumentumok szorzatát adja.
    Pl.: * (TIMES 3 5 7)
    105
  - QUOTIENT, vagy / - A két numerikus paramétere hányadosát adja, vagy egészek esetén, ha az nem egész, akkor p/q alakot.
    Pl.: * (QUOTIENT 6 3)
    2

A LISP programozási nyelv építõelemei ..

- **Beépített függvények ..**
  - LISTP - Eldönti, hogy a paramétere lista-e, vagy sem (akkor atom).
    Pl.: * (LISTP ‘(EGY KETTO HAROM))
    T
    * (LISTP ‘VALT1)
    NIL
    * (LISTP (CAR ‘(ALMA KORTE DIO)))
    NIL
  - AND - A paraméterek és kapcsolatának eredményével tér vissza. NIL a hamis, T az igaz (true) érték.
    Pl.: * (AND (LISTP ‘(EGY KETTO HAROM)) NIL)
    NIL
    * (AND T T)
    T
  - OR - A paraméterek vagy kapcsolatának eredményével tér vissza.
    Pl.: * (OR (LISTP ‘(EGY KETTO HAROM)) NIL)
    T
    * (OR NIL T NIL T)
    T

A LISP programozási nyelv építõelemei ..

- **Beépített függvények ..**

  - COND - Feltételek alapján való elágaztatás. Alakja:

```
(COND
  ( feltétel1   kiértékelendõ-1a  kiértékelendõ-1b … kiértékelendõ-1m )
  ( feltétel2   kiértékelendõ-2a  kiértékelendõ-2b … kiértékelendõ-2n  )
  …
  ( feltételk   kiértékelendõ-ka  kiértékelendõ-kb  … kiértékelendõ-ks  )
)
```

Értéke az elsõ igaz feltétel utolsó kiértékelendõjének értéke lesz.

```
Pl.:  *  (COND
             (  (EQUAL 5 6)  (PRINT OT-EGYENLO-HATTAL) )
             (  (LISTP '(A B))  (SETQQ SZAM 12.67))
         )
        12.67
```

A LISP programozási nyelv építõelemei ..

- **Beépített függvények ..**
  - EQUAL, vagy = - Egyenlõséges reláció. A visszatérõ érték T, ha a két paraméter értéke azonos.

    Pl.: * (EQUAL 15 15)
    T
    * (EQUAL 15 (TIMES 3 4))
    NIL
    * (EQUAL '(A B C) (LIST 'A 'B 'C))
    T

  - / = - Nemegyenlõ reláció. A visszatérõ érték T, ha az összes paraméter eltérõ. Egyébként NIL.

    Pl.: * (/= 15 15)
    NIL
    * (/= 15 (TIMES 3 4))
    T
    * (/= '(A B C) (LIST 'A 'B 'C))
    NIL

A LISP programozási nyelv építõelemei ..

- **Beépített függvények ..**
  - < - A visszatérõ érték T, ha a paraméterek értékei monoton növekvõ sorozatot alkotnak. Egyébként NIL.

    Pl.: * (< 12 15)
    T
    * (< 12 (TIMES 3 4))
    NIL
    * (< '(A B C) (LIST 'A 'B 'D))
    T

  - <= - A visszatérõ érték T, ha a paraméterek értékei monoton nem csökkenõ sorozatot alkotnak. Egyébként NIL.

    Pl.: * (<= 12 15)
    T
    * (<= 15 (TIMES 3 5))
    T
    * (<= '(A B C) (LIST 'A 'B 'C))
    T

A LISP programozási nyelv építõelemei ..

- **Beépített függvények ..**
  - > - A visszatérõ érték T, ha a paraméterek értékei monoton csökkenõ sorozatot alkotnak. Egyébként NIL.

    Pl.: * (> 12 11)
    T
    * (> 11 (TIMES 3 4))
    NIL
    * (> ‘(A B C) (LIST ‘A ‘B ‘D))
    NIL

  - >= - A visszatérõ érték T, ha a paraméterek értékei monoton nem növekvõ sorozatot alkotnak. Egyébként NIL.

    Pl.: * (>= 17 15)
    T
    * (>= 15 (TIMES 3 5))
    T
    * (>= ‘(A C D) (LIST ‘A ‘B ‘C))
    T

A LISP programozási nyelv építõelemei ..

- **Beépített függvények ..**
  - WHILE - Ciklusszervezõ függvény. Elsõ paramétere a feltétel, mindaddig újra és újra kiértékelõdnek a további paraméterek, amíg az elsõ paraméter T, azaz igaz. Visszatérési értéke az elsõ paraméter leállási értéke, azaz NIL.
  - Pl.: Számok összege 12-tõl 12000-ig:

```
*  (SETQQ OSSZEG 0)
0
*  (SETQQ CIKLUSVALTOZO 12)
12
*  (WHILE (<= CIKLUSVALTOZO 12000)
     (SETQ OSSZEG (+ OSSZEG CIKLUSVALTOZO))
     (SETQ CIKLUSVALTOZO (+ CIKLUSVALTOZO 1))
   )
NIL
*  OSSZEG
71999928
```

    - A WHILE függvényt ritkán használjuk, inkább rekurzív függvényhívást alkalmazunk.

A LISP programozási nyelv építõelemei ..

- **Saját függvény definiálása**
  - DEFUN - Függvénydefiniáló függvény.

    (DEFUN *függvénynév (param1 param2 … paramn*)
    kiértékelendõ1
    kiértékelendõ2
    …
    kiértékelendõw
)

    A függvénydefiniáló függvény hívásának eredményeként a függvénydefiníció eltárolódik.

  - Pl.: Számok négyzetét adó függvény definiálása:

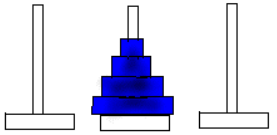
```
*  (DEFUN NEGYZET (SZAM)
       (TIMES SZAM SZAM)
   )
NEGYZET
```

A LISP programozási példa

- **Hanoi tornyai feladat megoldása rekurzív függvénnyel**

A KR kezdõrúdról kell a CR célrúdra átrakni a csökkenõ átmérõ szerint felsorolt K4,K3,K2,K1 korongokat az SR segédrúd felhasználásával. Mindig csak nagyobb korongra rakhatunk kisebb korongot.
Induló helyzet, a három rúdnak megfelelõ három listában megadva:

```
(K4 K3 K2 K1)  ; KR kezdõrúd
()             ; CR célrúd
()             ; SR segédrúd.
```



A rekurzív függvény bemenõ paraméterei a KR, CR, SR listák induló tartalommal, kimenõ értéke ugyanezen listák tartalma a végállapotnak megfelelõen.

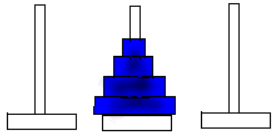
Az alkalmazott technika:

1. Az átrakni kívánt alsó korong fölötti korongokat képzeletben egyben, valójában ezekre a korongokra a Hanoi algoritmust rekurzívan alkalmazva átrakjuk az SR segédrúdra
2. Az átrakni kívánt, most már rajta lévõ korongok nélküli alsó korongot simán áttesszük a CR célrúdra
3. Végül az SR segédrúdról képzeletben egyben, valójában a Hanoi algoritmus rekurzív alkalmazásával a korongokat áttesszük a CR célrúdra.

A LISP programozási példa

- **A Hanoi tornyai feladat megoldását elvégzõ rekurzív függvény**

```
* (DEFUN HANOI (KR CR SR)
    (COND
      ((= (LENGTH KR) 1) ; a rekurzív hívások visszafordítója,
                         ; ezt az aktuális feladatbeli legalsót már simán
                         ; áttesszük:
        (LIST () (APPEND CR (LIST (CAR KR))) SR)
      )
      (T  ; egynél több korong van,  az alsó felettieket rekurzív technikával
          ;áttesszük CR-re:
        (SETQ SR (CAR (CDR (HANOI (CDR KR)  ()  () ))))   ; 1. lépés:
          ; a segédrúdon van az eggyel kisebb torony
        (SETQ CR (APPEND CR (LIST (CAR KR))))             ; 2. lépés:
          ; a résztorony legalsó korongját áttettük CR célrúdra
        (SETQ CR (CAR (CDR (HANOI SR CR () ) )))          ;  3. lépés:
          ; mintha SR lenne a kezdõrúd a résztoronnyal, melyet most
          ; átraktunk a CR célrúdra
        (LIST () CR () ) ; a függvény visszaadott értéke
      )
    )
  )
HANOI
```

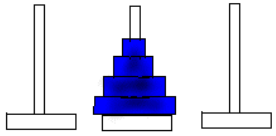


A LISP programozási példa

- **A Hanoi tornyai függvény használata**

  * (HANOI '(K4 K3 K2 K1) () ())
  (NIL (K4 K3 K2 K1) NIL)

- Az átrakási folyamat nyomonkövetéséhez PRINT függvényhívások beillesztésére, vagy a nyomkövetéssel történõ futtatásra van szükség:
  * (TRACE HANOI)
  HANOI
  * (HANOI '(K4 K3 K2 K1) () ())
  ; itt nyomon követhetjük a függvényhívásokat és az aktuális paramétereket.



Forrás: Futó Iván: Mesterséges intelligencia Aula Kiadó,1999.

Bevezetés a PROLOG programozási nyelvbe

- Jellemzõi:

  Hasonló a PLANNER-hez a következõkben:

  - Az eljárások és a tények a **mintaillesztés** során kerülnek elõ. Eltér viszont abban, hogy betartja a bemenõállítások **sorrendjét**.
  - A keresést automatikus **visszalépéssel** (back track) hajtja végre, amelyet azonban a programozó a ! (cut, **vágás**) operátorral módosíthat. Ez a lehetõség **ellentmond a logikai programozás ideális koncepciójának**: a programozó feladata csak a logikai probléma megadása, a problémaleírás algoritmikus kiértékelését pedig a gépnek kell elvégeznie.
    Kowalski képlete:

    algoritmus = logika + vezérlés

    Jelentése a következõ: egy algoritmus mindig két komponenst tartalmaz implicit formában:

    – a logikai összetevõt, amely megadja a kérdéses problémára vonatkozó tudást,

    – a vezérlési összetevõt, mely megtestesíti a megoldási stratégiát az adott probléma esetére.

A PROLOG jellemzõi

A Prolog további jellemzõi:

- Bevezetésre került a **symbol** típus (domain), hogy a predikátumlogika individuumkonstansai, -változói és -függvényei a valóság egy részletét leképezhessék.
- Fontos adatszerkezet a **lista**. A listafeldolgozás a **mintaillesztésen** és a **rekurzión** alapul.
- A függvények a predikátumfüggvényeken alapulnak, és **nincs különbség a bemenõ és a kimenõ változók között.**
- A bemeneti állítások (**szabályok**, clauses) formátuma korlátozott:

P **if** **Q and R and S.**

fej *törzs*

A fej egyetlen predikátum, a törzs pedig atomi formulák (többnyire predikátumok, azaz logikai függvények) negációt nem tartalmazó konjunkciója (csak *and* lehet).

A PROLOG jellemzõi ..

A Prolog további jellemzõi .. :

- Mivel a törzs atomi formulák konjunkciója, azaz Horn-halmaz, így azt mondhatjuk, hogy a PROLOG **a Horn halmazokra a rezolúció elvét alkalmazza.**

- **A felhasználóra hagyja a rezolúciós stratégia megvalósítását**: néhány megszorítást tett a bemenõállításokra, és az illesztés a bemenõállítások (clauses, tények és szabályok) megadott sorrendjében hajtódik végre. A rendszer így tömörebb lett és gyorsabb végrehajtást tesz lehetõvé.

- A predikátum logikára épül, de vannak eltérések:
  - Csak az elemváltozók kezdõdnek **nagybetûvel**.
  - Ítéletkonstansok közül a False (fail) **ritkán**, a True és ítéletváltozók még ritkábban fordulnak elõ.
  - **Rendszerpredikátumok** léteznek, melyek valamit elvégeznek és eközben igaz értéket vesznek fel. Pl.: read(). write(), circle(), stb.
  - **Infix** (közbeírt) alakú relációoperátorok (<,>, <=,>=,<>,=,><) és aritmetikai operátorok (+,-,*,/) a predikátumos alak helyett. Pl.: kisebb(X,Y) helyett X<Y írható.

A PROLOG jellemzõi ..

A Prolog további eltérései a predikátum logikától:

- Egyenlõséges reláció egyetlen hiányzó értékû változójának képes **kiszámítani** az értékét. Pl.: 5*7= X ⇒ X=35
- Aritmetikai kifejezéseket, mûveleti operátorokat végrehajtja.
- A szabályokban szereplõ ***if*** (:-) logikai szimbólumot úgy kezeli, mint egy infix alakban használt speciális rendszer-predikátumot, amely a PROLOG-tól egy speciális kiértékelõ algoritmust kíván.
- A szabályok és az atomok (fõként predikátumok) ugyanolyan módon kezelendõ objektumoknak minõsülnek.
- Megengedi **ítéletváltozók** alkalmazását a szabályokban és megengedi **helyettesítésüket szabályokkal**.
- Képes manipulálni saját "adatbázisát" (az **assert** és a **retract** rendszerpredikátumok útján.)
- Kényelmes **[Fej | Törzs]** listaalak. [ ] az üres lista jele.
- A megoldási folyamat programozó általi vezérlésére alkalmazza a **! jelet**, mely formailag egy atomi formula. Pl.: a if b and c and ! and d. A ! (**cut**) megváltoztatja, lerövidíti a PROLOG Depth-First (mélységi bejárás) algoritmusát a felesleges ágak bejárásának letiltásával, az ágak levágásával.

A PROLOG jellemzõi ..

A PROLOG programok felépítésének és eszköz-rendszerének részletezése elõtt tekintsünk egy egyszerû feladatot és PROLOG programját!

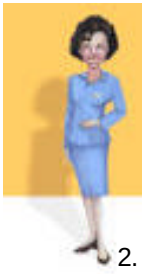
Ismertek az alábbi **tények**:

Cleopátra szebb Ginánál.

Gina szebb Ursulánál.

Kérdés: *Ki sokkal szebb Ursulánál ?*


1.


2.


3.

- A feladat megválaszolására alkalmas **PROLOG program**:

```
domains    hölgy = symbol
predicates      szebb(hölgy, hölgy)
                sokkal_szebb(hölgy,hölgy)
goal            sokkal_szebb(Valaki, ursula)
clauses    szebb(cleopátra,gina).
           szebb(gina, ursula).
           sokkal_szebb(A,C ) if szebb(A,B ) and szebb(B,C ).
```

1. www.dollsinmini.com/ showcase.htm 2. www.meetingadvice.com/columnists/ index.php3?Columnist=2
3. www.theatlantic.com/issues/ 98may/ursula.htm

A PROLOG programok részei és funkciójuk

- **domains**

  Ez a programrész - hasonlóan egy procedurális nyelv, pl. Pascal típusdeklarációs részéhez - a programban alkalmazott **term**ek lehetséges interpretációinak, értékeinek **tartományát** adja meg. Ez a tartomány, ill. típus lehet a rendszer által adott, vagy a programozó által deklarált.

  - A rendszer által nyújtott tartományok (típusok):

  - **symbol** pl.: elem = symbol
    - A symbol típusú konstansok kisbetûvel kezdõdõ szavak, pl. ha X elem típusú individuumváltozó, értékül kaphatja a következõket: **X:= a; X:= alma; X:= peter**, stb.
  - **char** pl.: betu = char
    - A char típusú indivíduumkonstans egy karakter, melyet értékül kaphat egy char típusú indivíduumváltozó. Pl.: **Kisbetu := 'k'**

Forrás: Makány György: Prologika ELTE Mikrológia sorozat 7. Saki Shop, 1994

A PROLOG programok részei és funkciójuk ..

- A rendszer által nyújtott tartományok (típusok) .. :
  - **string** **pl.: mondat = string**
    - A string típusú indivíduumkonstans egy ASCII karakterlánc, melyet értékül felvehet egy string típusú individuumváltozó. Pl.: **Input := "Kérem a következõt”**
  - **integer** **pl.: szam = integer**
    - Az integer típusú individuumkonstansok egy korlátos számtartományba esõ egész számok (pl.: -32768 - +32767), melyeket értékül felvehet egy integer típusú individuumváltozó. Pl.: **Egesz := -42**
  - **real** **pl.: valos = real**
    - A real típusú indivíduumkonstansok egy korlátos számtartományba esõ valós számok, melyeket felvehet értékül egy valós típusú indivíduumváltozó. Pl.: **Tort := 32.43**

A PROLOG programok részei és funkciójuk ..

- Programozó által deklarált tartományok:
  - A tartomány **értékkészletének** felsorolásával.
    Pl.: **irány = bal; egyenes; jobb**
    Egy irány típusú indivíduumváltozó a három individuumkonstans egyikét veheti fel értékként.
    Pl.: **Mutat:= bal**
  - **Individuumfüggvénnyel** megadott tartomány.
    Pl.: **xt, yt = symbol**
    fuggotartomany = fuggoen(xt,yt)

www.elementarykids.org/gym/boxhis.htm

- Megjegyzések:
  - A deklarált típusokból **összetett típusok** is létrehozhatók.
  - A domains rész **elhagyható**, ha csak a rendszer által nyújtott típusokat használjuk.
  - **Lista** megadása *-gal: pl.: lista = integer*
  - Az individuumváltozók és -függvények **típusát nem kell deklarálni.** Az, hogy milyen típussal bírnak, az alkalmazás helye által van meghatározva.
  - Stringek manipulálására rendszerpredikátumok léteznek.

A PROLOG programok részei és funkciójuk ..

- **predicates**

  Ez a programrész a programban alkalmazott **predikátumok** és az azokban alkalmazott **termek típusának** megadására szolgál.

  Pl.: **testvérek(gyerek, gyerek)**

  A **predikátum** objektumok sajátságainak ill. az objektumok közötti kapcsolatok, viszonyok, relációk szimbolikus formában való megadására szolgáló állítás.

- **goal**

  Ebben a programrészben a programban megadott tényekbõl és szabályokból levezethetõ állítás fogalmazható meg, melynek igaz, vagy hamis voltára akarunk a rendszertõl választ kapni. Amennyiben változót tartalmaz, a rendszer olyan változókat ad meg megoldásként, melyekkel az állítás igaz, vagy ha ez lehetetlen, akkor jelzi, hogy nincs megoldás.

A PROLOG programok részei és funkciójuk ..

- **goal ..**

  Formája: egyszerû esetben egy predikátum, összetettebb esetben predikátumokból álló formula. A predikátumok összekapcsolására használható az *és*, *vagy*, valamint a *nem* logikai mûvelet.
  Fej nélküli szabálynak is tekinthetjük.

  Pl.: **a and b or not c**

  vagy:

  sokkal_szebb(Valaki, ursula)

  vagy:

  clearwindow and megold([cs,fa],[ur],[ur],[cs,fa])

  A goal programrész **elmaradhat**, ha a DIALOG ablakban akarjuk az eldöntendõ állításunkat feltenni.

A PROLOG programok részei és funkciójuk ..

- **clauses**

  Ez a programrész szolgál a **szabályok és tények felsorolására**.

  A szabály formája: Fej if Törzs.

§

  - A szabály végén pont áll.
  - A Fej egyetlen predikátum.
  - A Törzs egy formula, mely predikátumok *and* logikai operátorokkal összekapcsolt sorozata.
  - A Fej állítása a Törzs állításainak, feltételeinek függvényében lehet igaz, vagy hamis. Felfogható úgy is, hogy a Fej cél a Törzsbeli részcélok megvalósulásának függvénye.
  - Amennyiben Törzs nincs, azaz a Fej-beli állítás nincs feltételekhez kötve, **Tényrõl** beszélünk.
  - A tények igaz értékûek.
  - Az azonos Fej-jel rendelkezõ szabályok csak **egymást követõ sorokban** állhatnak. Értelemszerûen az azonos nevû tények is csak egymást követõ sorokban állhatnak.

A PROLOG mûködése

A mûködés két alapvetõ összetevõje az

- **illesztés** (matching) és a
- **visszalépés** (backtracking).

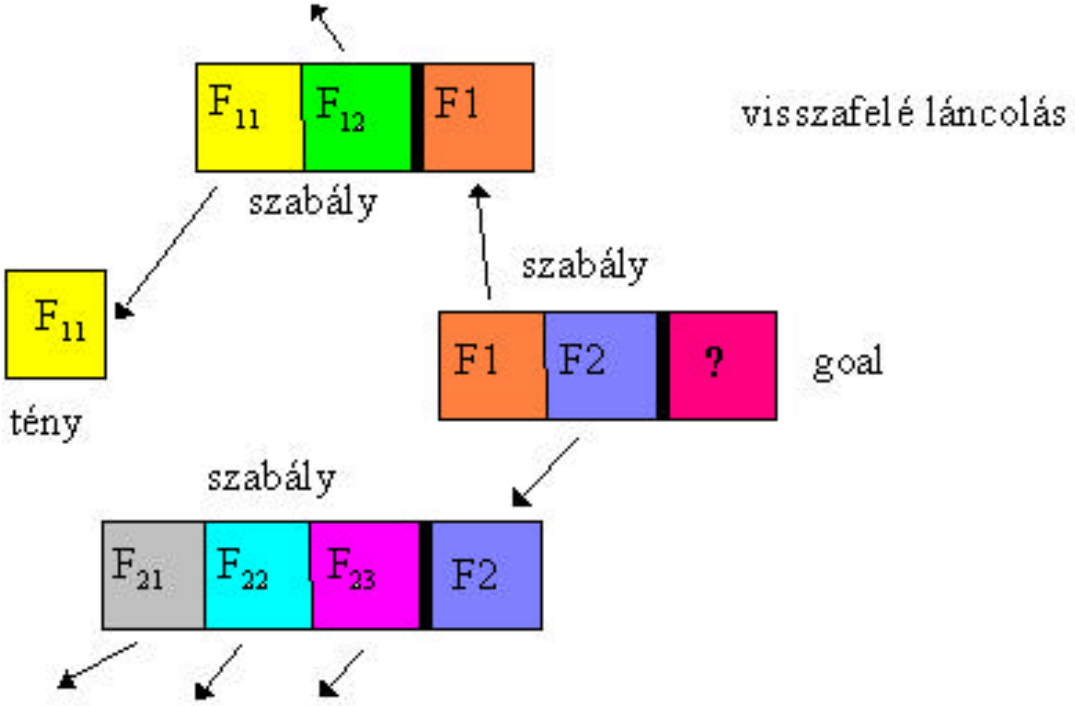
- Lényege: a goal-ban feltett kérdést megpróbálja **tényállításokra visszavezetni**. Amennyiben nem létezik illeszkedõ - nevében és a megadott paramétereiben egyezõ - tény, akkor illeszkedõ szabályt keres. Ha van illeszkedõ szabály, akkor az elõbbi lépések ismétlésével megpróbálja a szabály Törzsében lévõ feltételek kiértékelésével meghatározni a szabály Fejének igazságértékét (Hátraláncolás).

- Az **összes** lehetõség végigpróbálásával megállapítja, hogy a kérdés teljesülhet-e, és ha igen, hányféle módon.

- Ha próbálkozás közben zsákutcába jut - nincs illeszkedõ tény, vagy szabály -, **visszalép** a legutolsó elágazási pontig, ahol a több illesztési lehetõség közül a következõt választja.

dr.Dudás László

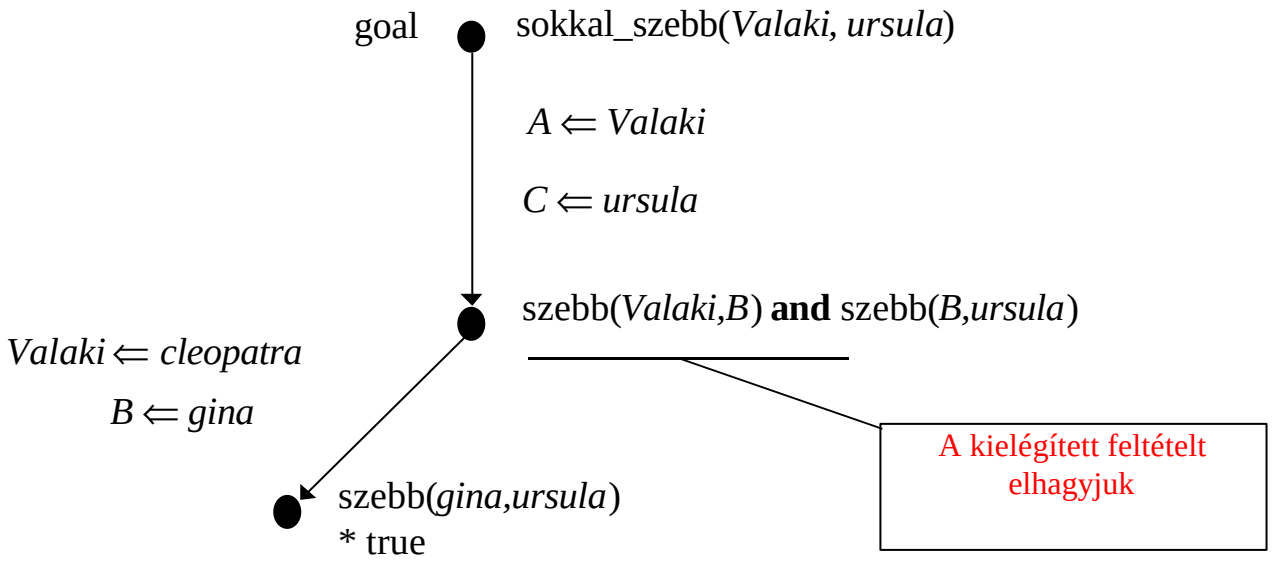
A PROLOG mûködése ..



A PROLOG mûködése ..

Szemléltetés fagráffal:

domains hölgy = **symbol**
predicates szebb(*hölgy, hölgy*)
sokkal_szebb(*hölgy, hölgy*)
goal sokkal_szebb(*Valaki, ursula*)
clauses szebb(*cleopátra,gina*).
szebb(*gina, ursula*).
sokkal_szebb(*A,C*) **if** szebb(*A,B*) **and** szebb(*B,C*).

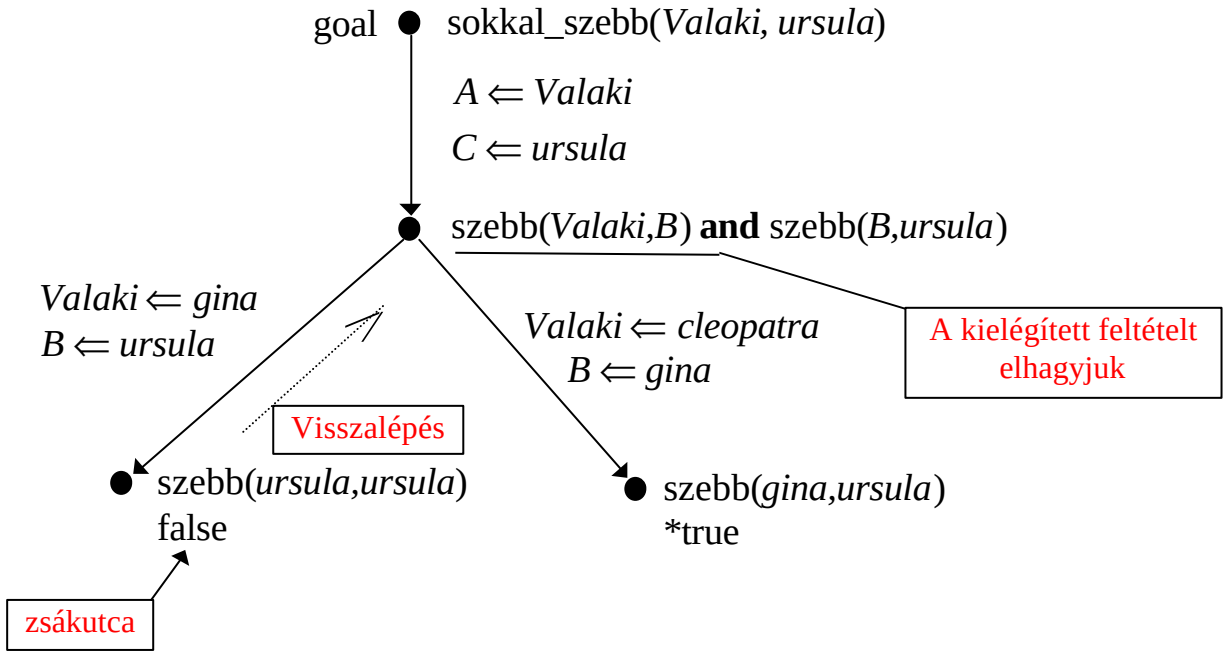


A PROLOG mûködése ..

Szemléltetés fagráffal:

domains hölgy = **symbol**
predicates szebb(*hölgy, hölgy*)
sokkal_szebb(*hölgy, hölgy*)
goal sokkal_szebb(*Valaki, ursula*)
clauses szebb(*gina, ursula*).
szebb(*cleopátra,gina*).
sokkal_szebb(*A,C*) **if** szebb(*A,B*) **and** szebb(*B,C*).

A tényeket felcseréltük



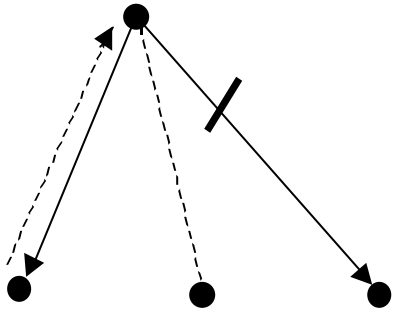
A vágás mûködése a PROLOG-ban

- Fogalmazzuk meg PROLOG nyelven a „szóvégi ú mindig hosszú” szabályt kivételeivel együtt! Megoldást adó szabályok:

```
szóvégi_ú_hosszú(kapu) if ! and fail.      ; kivétel
szóvégi_ú_hosszú(alku)  if ! and fail.     ; kivétel
…                                          ; további kivételek itt
szóvégi_ú_hosszú(X).                       ; általános eset
```

Fail a hamis érték.

goal szóvégi _ú_hosszú(kapu) ⇒ No



! and fail. Nem ill. szóvégi_ú_hosszú(X)

Elõbb levágja az illeszkedõ utolsó állítás ágát, majd fail miatt visszalép. Vágás nélkül átlépne a szóvégi_ú_hosszú(X) állításra, XÜ kapu illesztés után, ami rossz eredményt adna.

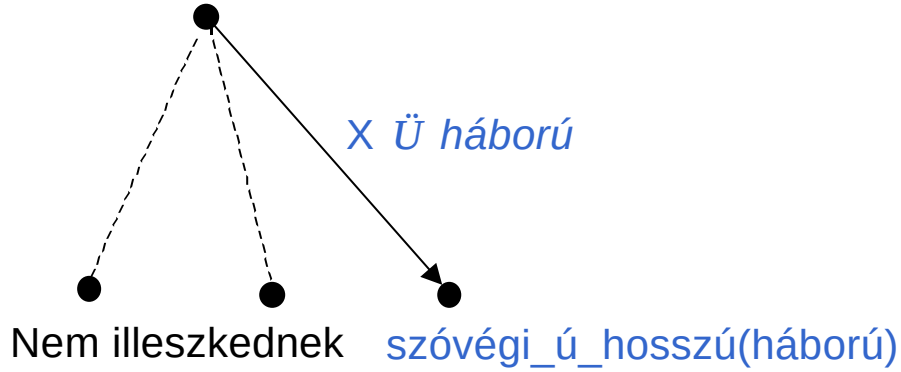
Forrás: Farkas - Futó - Langer - Szeredi: MPROLOG programozási nyelv Mûszaki Könyvkiadó Bp. 1989

A vágás mûködése a PROLOG-ban

- Nézzük meg a *háború* szó esetére a mûködést!

```
szóvégi_ú_hosszú(kapu) if ! and fail.      ; kivétel
szóvégi_ú_hosszú(alku)  if ! and fail.     ; kivétel
…                                          ; további kivételek itt
szóvégi_ú_hosszú(X).                       ; általános eset
```

goal szóvégi _ú_hosszú(háború) ⇒ Yes, X = háború



A kivételek nem illeszkednek, az utolsó tény viszont minden szóval illeszkedik, így a háború szó esetén a szúvégi_ú_hosszú predikátum igaz értéket ad.

Forrás: Farkas - Futó - Langer - Szeredi: MPROLOG programozási nyelv Mûszaki Könyvkiadó Bp. 1989