

Deklaratív programozás 7. előadás

Kabódi László¹ Marussy Kristóf²

BME Számítástudományi és Információelméleti Tanszék
Mesterséges Intelligencia és Rendszertervezés Tanszék

¹`kabodi.laszlo@vik.bme.hu`

²`marussy@mit.bme.hu`

2026. ősz

I. rész

Programozás Prologban

1 Programozás Prologban

A kurzus Logikai Programozás (LP) része

- A Prolog LP nyelv alapjai
 - Szintaxis
 - Deklaratív szemantika
 - Procedurális szemantika (végrehajtási mechanizmus)
 - A legfontosabb beépített eljárások
 - Prolog programozási módszerek
- Haladó deklaratív programozás Prologban
 - Fejlettebb nyelvi és rendszerelemek
 - Kitekintés: új irányzatok a logikai programozásban

A logikai programozás alap gondolata

- Logikai programozás (LP):
 - Programozás a matematikai logika segítségével
 - egy logikai program nem más mint **logikai állítások** halmaza
 - egy logikai program futása egy **következtetési folyamat**
 - De: a logikai következtetés óriási keresési tér bejárását jelenti. . .
 - Szorítsuk meg a logika nyelvét!
 - Válasszunk egyszerű, ember által is követhető következtetési algoritmust!
 - Az LP máig legelterjedtebb megvalósítása a már több mint 50 éves **Prolog** = Programozás **logikában** (**P**rogramming in **l**ogic)
 - az elsőrendű logika (First Order Logic, **FOL**) egy erősen megszorított résznyelve az ún. **definit**- vagy más néven **Horn-klózek**
 - végrehajtása (következtetés): **mintaillesztéses** eljáráshíváson alapuló **visszalépéses** keresés.
 - A Prolog nyelv mottója: „**WHAT** rather than **HOW**”

A Prolog/LP rövid történeti áttekintése

1960-as évek	Első tételbizonyító programok
1970-72	A logikai programozás elméleti alapjai (R A Kowalski)
1972	Az első Prolog interpreter (A Colmerauer)
1975	A második Prolog interpreter (Szeredi P)
1977	Az első Prolog fordítóprogram (D H D Warren)
1977–79	Számos kísérleti Prolog alkalmazás Magyarországon
1981	A japán 5. generációs projekt a logikai programozást választja
1982	A magyar MProlog az egyik első kereskedelmi forgalomba kerülő Prolog megvalósítás
1983	Egy új fordítási modell és absztrakt Prolog gép (WAM) megjelenése (D H D Warren)
1986	Prolog szabványosítás kezdete
1987–89	Új logikai programozási nyelvek (CLP, Gödel stb.)
1990–...	Prolog megjelenése párhuzamos számítógépeken Nagyhatékonyágú Prolog fordítóprogramok

.....

Tartalom

- 1 Programozás Prologban
 - A funkcionális és logikai megközelítés összevetése
 - Prolog bevezetés – példák

A logikai és a funkcionális programozás összehasonlítása

Mi a közös a funkcionális (FP) és logikai (LP) nyelvekben?

- 1 A matematikai változó fogalma: **egyetlen** még ismeretlen értéket jelöl, nem mutábilis (nem változtatható)
- 2 Ciklusok helyett: rekurzió, vagy más eszközök (pl. listanézet)

Miben más az LP (ill. a Prolog) mint az FP megközelítés?

- 3 Az FP a lambda-kalkuluson alapul, az LP alapja az **elsőrendű logika** (FOL)
 - Az LP-ben függvények helyett **predikátumok** (relációkat, vö. adatbáziskezelés) kell definiálnunk, predikátum $\equiv$ **eljárás**
 - Egy LP program futása 0, 1, vagy több eredményt adhat – **nemdeterminisztikus keresés**, vö. SQL lekérdezések
- 4 A Prolog szintaxisa kiterjeszthető – saját **új operátorokat** definiálhatunk
- 5 LP-ben a változók egy adatstruktúra (pl. lista) belsejében is előfordulhatnak – ezek a **logikai változók** pointerként működnek, pl. $[x, x, x]$ egy olyan 3-elemű listát jelöl, amely csupa azonos (de egyelőre még ismeretlen) elemből áll
- 6 Az LP különösen jó **szimbolikus** feladatokra (pl. szimbolikus deriválás)
- 7 Sok FP nyelv van, de praktikusán a Prolog az egyetlen LP nyelv

Ismétlés: listák összefűzése

- A Prolog lista szintaxisa megegyezik az Elixir szintaxissal, de Prologban a változók kötelezően nagybetűvel vagy aláhúzásjellel (_) kezdődnek

- Idézzük fel a két listát összefűző `app` Elixir függvényt (`app/2`):

```
# app(l1, l2): l1 és l2 listák összefűzöttje (l1⊕l2)
def app([], b) do b end # [] ⊕ b = b
def app([x|a], b) do [x|app(a,b)] end # [x|a] ⊕ b = [x|a⊕b]
```

- Ennek egy 3-argumentumú Prolog predikátum felel meg, (`app/3`), itt a 3. argumentum lesz az összefűzött lista (az Elixir függvény eredménye):

```
% app(L1, L2, L12): L1 és L2 listák összefűzöttje L12 (L1⊕L2=L12)
app([], B, B). % [] ⊕ B = B
app([X|A], B, [X|C]) :- % [X|A] ⊕ B = [X|C] ha
    app(A, B, C). % A ⊕ B = C
```

- A `c` segédváltozó az $A \oplus B$ részeredményt tárolja.

- Példa az `app/3` predikátum használatára:

```
| ?- app([1,2], [3,4], L).    ⇒    L = [1,2,3,4] ? ;
                             no
```

- A fenti `app/3` eljárás `append` néven beépített predikátumként is elérhető.

Be- és kimenő argumentumok

- Az `app/3` predikátum az `app/2` Elixir függvény átírásával állt elő
- A Prolog predikátum azonban használható más **módon** is, pl:


```
| ?- app(L1, L2, [1,2]).
L1 = [], L2 = [1,2] ? ;
L1 = [1], L2 = [2] ? ;
L1 = [1,2], L2 = [] ? ; no
```
- Az ún. **I/O mód** jelölésrendszer a különböző módú hívások leírására:
 - **+**: bemenő (input) arg., a hívás pillanatában behelyettesített
 - **-**: kimenő (output) arg., a hívás pillanatában behelyettesítetlen
 - **?**: be- és kimenő arg., tetszőleges Prolog kifejezés lehet
- Példák az `app(L1, L2, L3)` hívás különböző módú hívásaira:
 - `(+,+,+)`: konkatenálás ellenőrzése, pl. `app([1], [2], [1,2]) ⇒ yes`
 - `(+,+,-)`: konkatenálás, pl. `app([1], [2], L3) ⇒ L3 = [1,2]`
 - `(+,-,+)`: két lista „különbsége”, pl. `app([1], L2, [1,2]) ⇒ L2 = [2]`
 - `(+,-,-)`: nyílt végű lista előállítás, pl. `app([1], L2, L3) ⇒ L3 = [1|L2]`
 - `(-,-,+)`: lista szétszedése, pl. `app(L1, L2, [1,2]) ⇒ lásd fent`
 - `(-,?,-)`: ∞ keresési tér: pl. `app(L1, [1], L3) ⇒ L1 = [], L3 = [1]? ;`
`L1 = [A], L3 = [A,1]? ; L1 = [A,B], L3 = [A,B,1]? ; ...`
- Az eredményekben logikai változók is lehetnek, lásd fenn, pl. **L2, A, B** stb.

Tartalom

- 1 Programozás Prologban
 - A funkcionális és logikai megközelítés összevetése
 - Prolog bevezetés – példák

A Prolog alapelemei: a családi kapcsolatok példája

- Adottak személyekre vonatkozó, adatbázis-szerű állítások, pl.

„gyerek–szülő” tábla

gyerek	szülő
Imre	István
Imre	Gizella
István	Géza
István	Sarolt
Gizella	C. Henrik
Gizella	B. Gizella

„férfiak” tábla

férfi
Imre
István
Géza
C. Henrik

- Rövidítések feloldása: C. Henrik $\Rightarrow$ Civakodó Henrik,
B. Gizella $\Rightarrow$ Burgundi Gizella
- Definiáljuk az unoka–nagyszülő kapcsolatot, azaz hozzunk létre egy származtatott (virtuális) „unoka–nagyszülő” táblát!

A nagyszülő feladat — Prolog megoldás

- Egy Prolog program állításokból, ún. **klózokból** (**clause**) áll
- A legegyszerűbb klóz a **tényállítás** (**fact**), formája:

$$\text{relációnév}(\text{Arg}_1, \dots, \text{Arg}_n).$$
 (ez egy **klózfej**)
- A relációnév egy **névkonstans** (**atom**): kisbetűvel kezdődő azonosító vagy aposztrófok közé zárt tetsz. karaktersorozat (első közelítésben)
- Az argumentumok lehetnek névkonstansok, változók, stb.
- A változókat nagybetűvel kezdődő azonosítókkal – pl. **Gy**, **Sz** – jelöljük
- Az Imre herceg őseit leíró adatbázis-táblák Prolog alakja:

```
% sz(Gy, Sz): Gy szülője Sz.
```

```
sz('Imre', 'Gizella'). % (sz1)
```

```
sz('Imre', 'István'). % (sz2)
```

```
sz('István', 'Sarolt'). % (sz3)
```

```
sz('István', 'Géza'). % (sz4)
```

```
sz('Gizella', 'B. Gizella'). % (sz5)
```

```
sz('Gizella', 'C. Henrik'). % (sz6)
```

```
% ffi(Személy): Személy férfi.
```

```
ffi('Imre'). % (f1)
```

```
ffi('István'). % (f2)
```

```
ffi('Géza'). % (f3)
```

```
ffi('C. Henrik'). % (f4)
```

- A predikátumok **jelentését** egy **% fejkomment**-tel írjuk le, **/* ez is komment */**
- Azonos nevű és argumentumszámú klózok sorozata egy **predikátumot** alkot, pl. a fenti klózok az **sz/2** ill. **ffi/1** predikátumokat

A nagyszülő feladat — Prolog megoldás (folyt.)

- A klózok másik fajtája az ún. **szabály** (rule), formája:

```
klózfej :- cél1, ..., célk.           % klózfej ← cél1 ∧ ... ∧ célk
      % ^-klóztörzs-^
```

- A **cél** (goal), más néven **hívás** (call) szintaxisa (azonos a **klózfej**-ével):
relációnév(Arg₁, ..., Arg_n)

- A „nagyszülője” kapcsolatot definiáló szabály:

```
% Gyerek nagyszülője Nagyszulo.
nsz(Gyerek, Nagyszulo) :-           % Gyerek nagyszülője Nagyszulo ha ∃ Szulo
    sz(Gyerek, Szulo),              % Gyerek szülője Szulo és
    sz(Szulo, Nagyszulo).            % Szulo szülője Nagyszulo           (nsz)
```

- Egy program futtatásához egy **célsorozat**ot (lekérdezést) kell megadni:

```
% Ki Imre nagyapja? (pontosabban Ki Imre férfi nagyszülője?)
| ?- nsz('Imre', NA), ffi(NA).      NA = 'C. Henrik' ? ;
                                     NA = 'Géza' ? ; no

% Ki Géza unokája?
| ?- nsz(U, 'Géza').                U = 'Imre' ? ; no

% Ki Imre nagyszülője?
| ?- nsz('Imre', NSz).              NSz = 'B. Gizella' ? ;
                                     NSz = 'C. Henrik' ? ;
                                     NSz = 'Sarolt' ? ; NSz = 'Géza' ? ; no
```

Deklaratív szemantika – klózik logikai alakja

- A **szabály** jelentése egy implikáció: a törzsbeli célok **konjunkciójából** **következik** a fej.
 - Példa: $\text{nsz}(\text{Gy}, \text{NSz}) \text{ :- } \text{sz}(\text{Gy}, \text{Sz}), \text{sz}(\text{Sz}, \text{NSz}).$
 - Logikai alak: $\forall \text{Gy}, \text{NSz}, \text{Sz} (\text{nsz}(\text{Gy}, \text{NSz}) \leftarrow \text{sz}(\text{Gy}, \text{Sz}) \wedge \text{sz}(\text{Sz}, \text{NSz}))$
 - Ekvivalens alak: $\forall \text{Gy}, \text{NSz} (\text{nsz}(\text{Gy}, \text{NSz}) \leftarrow \exists \text{Sz} (\text{sz}(\text{Gy}, \text{Sz}) \wedge \text{sz}(\text{Sz}, \text{NSz})))$
- A **tényállítás** feltétel nélküli állítás, pl.
 - Példa: $\text{sz}(\text{'Imre'}, \text{'István'}).$
 - Logikai alakja változatlan
 - Ebben is lehetnek változók, ezeket is univerzálisan kell kvantálni
- A **célsorozat** jelentése: keressük azokat a változó-behelyettesítéseket amelyek esetén a célok konjunkciója igaz
- Egy célsorozatra kapott válasz **helyes**, ha az adott behelyettesítésekkel a célsorozat következménye a program logikai alakjának – **WHAT**
- A Prolog garantálja a helyességet, de a **teljességet** nem: nem biztos, hogy minden megoldást megkapunk (kaphatunk hibajelzést, végtelen ciklust, végtelen keresési teret stb.) – **HOW**

Procedurális szemantika: az ún. redukciós modell

Redukciós lépés: egy CS_i célsorozat **visszavezetése** CS_{i+1} -re úgy, hogy

$$CS_i \leftarrow \text{Program} \wedge CS_{i+1}$$

Pl. az (1) célsorozat **redukciója** az (nsz) programklózzal (2)-t eredményezi:

$:-$ nsz('Imre', NA), ffi(NA). (kiinduló célsorozat) (1)

$:-$ sz('Imre', Sz1), sz(Sz1, NA), ffi(NA). (redukált célsorozat) (2)

- 1 A klózt **lemásoljuk**, a változókat szisztematikusan újakra cserélve

nsz(Gy1, NSz1) :- sz(Gy1, Sz1), sz(Sz1, NSz1). (nsz')

- 2 (1)-et szétbontjuk, első cél: nsz('Imre', NA), maradék célsor.: ffi(NA).

- 3 Az első célt **egyesítjük** a klózfejjel, azaz változók behelyettesítésével a klózfejjel azonos alakra hozzuk (**kétirányú** mintaillesztés):

behelyettesítés: Gy1 = 'Imre', NSz1 = NA, közös alak: nsz('Imre', NA)

- 4 Ha az egyesítés sikertelen, akkor a redukciós lépés megghiúsul, egyébként behelyettesítjük a klóztörzset: sz('Imre', Sz1), sz(Sz1, NA) és a maradék célsorozatot is (ebben most nincs változás): ffi(NA)

- 5 Új célsorozat = klóztörzs és utána a maradék célsorozat (lásd fenn, (2))

Az (1) $\rightarrow$ (2) redukciós lépés értelmezhető az (nsz) „makró” kifejtéseként. . .

További redukciós lépések

A (2) célsorozat **redukciója** az (sz1) programklózzal:

```
:- sz('Imre', Sz1), sz(Sz1, NA), ffi(NA). (2)
```

- 1 Az (sz1) klóz nem tartalmaz változót, így nem szükséges lemásolni:

```
sz('Imre', 'Gizella') /* :- (üres klóztörzs) */. (sz1)
```

- 2 (2) első célja: sz('Imre', Sz1), maradék célsor.: sz(Sz1, NA), ffi(NA).

- 3 Az első célt **egyesítjük** a klózfejjel
behelyettesítés: Sz1 = 'Gizella', közös alak: sz('Imre', 'Gizella')

- 4 A behelyettesített maradék: sz('Gizella', NA), ffi(NA).

- 5 Az új célsorozat: az (sz1) klóz (üres) törzse + a maradék célsorozat:

```
:- sz('Gizella', NA), ffi(NA). (3)
```

(Tényállítással redukálva 1-gyel csökken a célsorozat hossza!)

(3)-at redukálva (sz6)-tal (sz('Gizella', 'C. Henrik').) a NA = 'C. Henrik' behelyettesítést kapjuk, az új célsorozat:

```
:- ffi('C. Henrik'). (4)
```

(4)-et redukálva (f4)-gyel (ffi('C. Henrik').) üres célsorozatot (□) kapunk. Ezzel megállapítottuk, hogy az NA = 'C. Henrik' egy megoldás (1)-re.

A nagyszülő példa végrehajtása – egy teljes levezetés

```

% sz(Gy, Sz): Gy szülője Sz.
sz('Imre', 'Gizella').    % (sz1)
sz('Imre', 'István').    % (sz2)
sz('István', 'Sarolt').   % (sz3)
sz('István', 'Géza').     % (sz4)
sz('Gizella', 'B. Gizella'). % (sz5)
sz('Gizella', 'C. Henrik'). % (sz6)

% ffi(Személy): Személy férfi.
ffi('Imre').    % (f1)
ffi('István').  % (f2)
ffi('Géza').    % (f3)
ffi('C. Henrik'). % (f4)

% Gy nagyszülője NSz.
nsz(Gy, NSz) :-          sz(Gy, Sz), sz(Sz, NSz).    % (nsz)

(1) :- nsz('Imre', NA), ffi(NA).    (nsz): (1) ← (2)

(2) :- sz('Imre', Sz1), sz(Sz1, NA), ffi(NA).    (sz1): (2) ← (3)
      Sz1 = 'Gizella'

(3) :- sz('Gizella', NA), ffi(NA).    (sz6): (3) ← (4)
      NA = 'C. Henrik'

(4) :- ffi('C. Henrik').    (f4): (4) ← (5)

(5) □    (5) azonosan igaz

```

Bebizonyítottuk, hogy (1) teljesül az `NA = 'C. Henrik'` behelyettesítés esetén

A nagyszülő példa – a válasz követése az `answer` cél segítségével

```
% sz(Gy, Sz): Gy szülője Sz.
sz('Imre', 'Gizella').    % (sz1)
sz('Imre', 'István').     % (sz2)
sz('István', 'Sarolt').   % (sz3)
sz('István', 'Géza').     % (sz4)
sz('Gizella', 'B. Gizella'). % (sz5)
sz('Gizella', 'C. Henrik'). % (sz6)

% ffi(Személy): Személy férfi.
ffi('Imre').             % (f1)
ffi('István').           % (f2)
ffi('Géza').             % (f3)
ffi('C. Henrik').        % (f4)
```

```
% Gy nagyszülője NSz.
```

```
nsz(Gy, NSz) :-          sz(Gy, Sz), sz(Sz, NSz).          % (nsz)
```

```
(1) :- nsz('Imre', NA), ffi(NA),          answer(NA).          (nsz): (1) ← (2)
```

```
(2) :- sz('Imre', Sz1), sz(Sz1, NA), ffi(NA), answer(NA).    (sz1): (2) ← (3)
```

```
(3) :- sz('Gizella', NA), ffi(NA),          answer(NA).          (sz6): (3) ← (4)
```

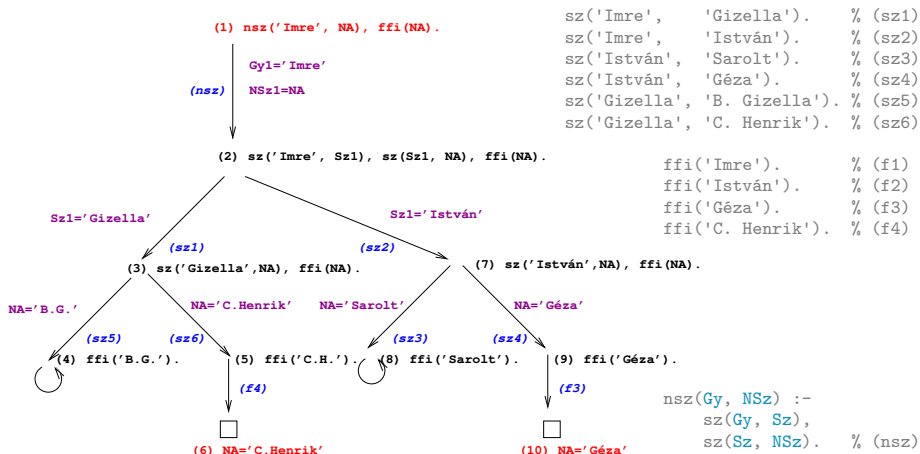
```
(4) :- ffi('C. Henrik'),          answer('C. Henrik').    (f4): (4) ← (5)
```

```
(5) :-          answer('C. Henrik').    A lekérdezés sikeres
```

A futás végén az `answer` „virtuális” cél tartalmazza a választ.

A nagyszülő példa végrehajtása – keresési tér

- A Prolog minden lehetséges redukciót szisztematikusan végigpróbál,
- balról jobbra haladó mélységi keresés formájában.



A Prolog végrehajtási algoritmus – első közelítés

Egy célsorozat végrehajtása

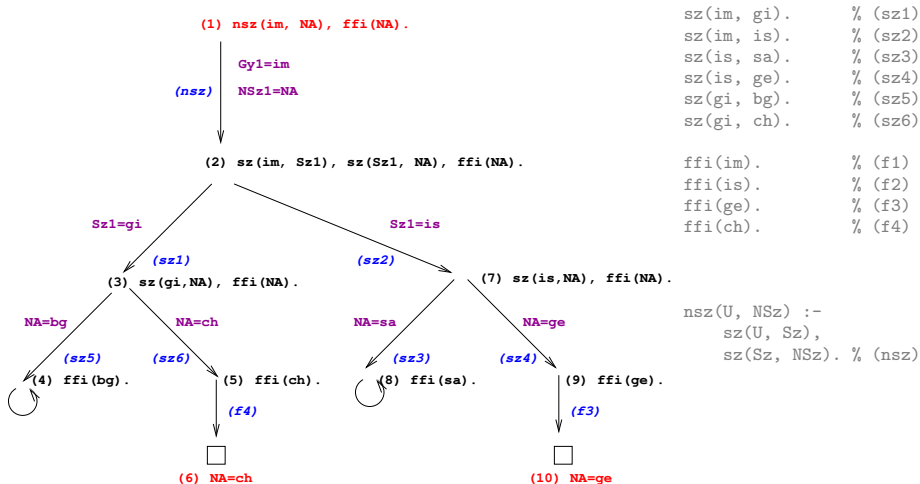
1. Ha az **első** cél beépített eljárást (BIP) hív, végrehajtjuk a BIP-et.
2. Ha az **első** cél felhasználói eljárásra vonatkozik, akkor megkeressük az eljárás **első** (visszalépés után: következő) olyan klózat, amelynek feje egyesíthető a hívással, és végrehajtjuk a redukciót.
3. Ha a redukció sikeres (találunk egyesíthető fejű klózt), folytatjuk a végrehajtást 1.-től az új célsorozattal.
4. Ha a redukció megghiúsul, akkor visszalépés következik:
 - visszatérünk a legutolsó, felhasználói eljárással történt (sikeres) redukciós lépéshez,
 - annak *bemeneti* célsorozatát megpróbáljuk *újabb* klózzal redukálni – ugrás a 2. lépésre
(Ennek megghiúsulása értelemszerűen újabb visszalépést okoz.)

A végrehajtás nem „intelligens”

- Pl. :- nsz(**Gy**, 'Géza'). hatékonyabb lenne ha a klóz törzsét **jobbról balra** hajtánánk végre
- DE: így a **végrehajtás a program írója számára** átlátható; a Prolog nem **tételbizonyító**, hanem programozási nyelv (**WHAT** rather than **HOW**)

A nagyszülő példa „tömörített” változata

- Imre herceget és felmenőit kétbetűs atomokkal jelöljük:
Imre $\Rightarrow$ im, Gizella $\Rightarrow$ gi, István $\Rightarrow$ is, Sarolt $\Rightarrow$ sa, Géza $\Rightarrow$ ge,
Burgundi Gizella $\Rightarrow$ bg, Civakodó Henrik $\Rightarrow$ ch.



A cél-redukciós modell alapfogalmai

- A végrehajtás bemenete:
 - egy Prolog program (klózek sorozata), pl. a „nagyyszülő” program, és
 - egy célsorozat, pl. `:- nsz(im, Sz).`
 a megoldás meghatározása érdekében ezt egy utolsó,
`answer(Megoldás)` fiktív céllal bővítjük ki, pl.

```
:- nsz(im, NSz), answer(NSz).      % Kik Imre nagyszülei?
:- sz(Gy, Sz), answer(Gy-Sz).    % Mik a gyerek-szülő párok?
```
- Az `answer(...)` cél segítségével követhetjük a megoldás felépülését
- Ha a célsorozat már csak az `answer` célt tartalmazza, akkor eljutottunk egy megoldáshoz (ezt a szerepet korábban az üres célsorozat játszotta)
- Az `answer` csak egy elméleti eszköz, nem beépített elj., de definálhatjuk, így: `answer(M) :- write(M), nl, fail.`
- A végrehajtásnak többféle kimenetele lehetséges:
 - Hiba (kivétel, exception), pl. `:- Y = alma, X is Y+1.`
 (Ezzel most nem foglalkozunk részletesebben.)
 - Meghiúsulás (nincs megoldás), pl. `:- sz(ge, Sz), answer(Sz).`
 - Siker (1 vagy több megoldás), pl. `:- sz(im, Sz), answer(Sz).`

A redukciós végrehajtás alapfogalmai (folyt.)

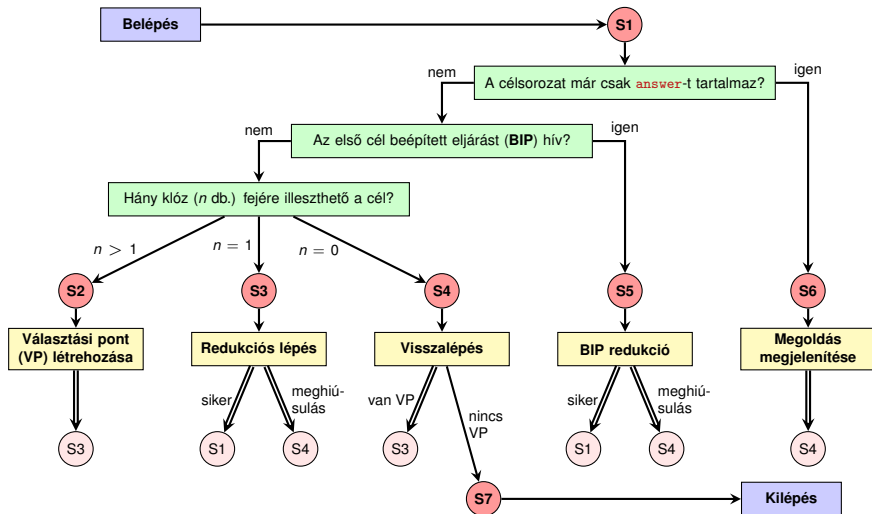
- A végrehajtás által használt (imperatív!) adatstruktúrák:
 - a jelenlegi célsorozatot tartalmazó változó (Goal)
 - a választási pontokat (VP) tartalmazó verem (Choice point stack)
- Például a `nsz(im, NA)`, `ffi(NA)`, `answer(NA)` célsorozat végrehajtásakor az alábbi VP verem jön létre:

Choice point stack

ChPoint name	Clause list	Goal	
CHP2	[sz5,sz6]	(3)	<code>sz(gi, NA), ffi(NA), answer(NA).</code>
CHP1	[sz1,sz2]	(2)	<code>sz(im, Sz), sz(Sz, NA), ffi(NA), answer(NA).</code>

- A VP verem akkor mélyül, ha 2 vagy több klózzal lehet redukálni
 - a redukció előtt a veremre elmentjük a célsorozatot és a redukcióban használható klózek listáját, majd folytatjuk a végrehajtást
 - ennek megghiúsulása esetén
 - a verem tetején levő klózlistából elhagyjuk az első elemet,
 - a klózlistában most első klózzal folytatjuk a redukciót,
 - ezt megelőzően, ha egyelemű a klózlista, megszüntetjük a VP-t
 - ha megghiúsuláskor üres a VP-verem $\Rightarrow$ kimerítettük a keresési teret

A redukciós modell folyamatábrája



A kettős nyilak jelentése: ugrás a halványlila körben megadott lépésre (folytatás a megfelelő sötétlila körnél).

Megjegyzések a folyamatábrához

- Hétféle végrehajtási lépésünk van: **S1–S7**, ahol **S1** a kiindulási pont (de közbülső is), **S7** a végállapot.
- Az **S1** lépés alapvető feladata az elágaztatás az **S2–S6** lépések egyikére
 - ha `Goal` már csak az `answer(...)` elemet tartalmazza $\Rightarrow$ **S6**;
 - ha az első cél beépített eljárást hív $\Rightarrow$ **S5**;
 - egyébként az első cél felhasználói eljárást hív. Ekkor megvizsgáljuk (általában **csak közelítően**), hogy az eljárás mely klózainak fejére illeszthető az első cél, és ezek száma (n) szerint $\Rightarrow$ **S2**, **S3** vagy **S4**.
- **S2** létrehoz egy VP-t, majd az első klózzal redukál ($\Rightarrow$ **S3**).
- **S3** meghiúsulhat, ha **S1**-ben n csak közelítés volt, ilyenkor $\Rightarrow$ **S4**.
- **S4** a VP-ban eltárolt következő klózzal való redukcióra lép ($\Rightarrow$ **S3**), ha van ilyen; egyébként befejezi a végrehajtást ($\Rightarrow$ **S7**).
- **S5** az **S3** lépéssel analóg módon vagy $\Rightarrow$ **S1**, vagy $\Rightarrow$ **S4**.
- **S6**-ban a megoldás megjelenítése után visszalépéssel folytatjuk ($\Rightarrow$ **S4**, további megoldások keresése).

A Prolog adatfogalma, a Prolog kifejezés (term)

- konstans (az **atomic** beépített eljárásnak felel meg)
 - számkonstans (**number**) – egész/lebegőpontos, pl. 1, -2.3, 3.0e10
 - névkonstans (**atom**), pl. 'István', szuloje, *, ++, tree_sum
 - egy C konstans **funktor** C/0
- összetett- vagy struktúra-kifejezés (**compound**)
 - ún. kanonikus alak: $\langle \text{struktúranév} \rangle (\langle \text{arg}_1 \rangle, \dots, \langle \text{arg}_n \rangle)$
 - a $\langle \text{struktúranév} \rangle$ egy névkonstans, az $\langle \text{arg}_i \rangle$ argumentumok tetszőleges Prolog kifejezések
 - a kifejezés **funktor**: $\langle \text{struktúranév} \rangle / n$
 - példák: `person(ian,smith,2003)`, `<(X,Y), is(X, +(Y,1))`
 - szintaktikus „édesítőszerek”, pl. operátorok és listák:
 $X \text{ is } Y+1 \equiv \text{is}(X, +(Y,1))$, ill. $[1,2,3|X] \equiv .(1,.(2,.(3,X)))$
- változó (**var**), pl. **Valtozo**, **X**, **_var**, **_** (don't care) **(nincs funktor)**
 - a változó alaphelyzetben behelyettesíthetetlen, értékkel nem bír, egyesítés során egy tetszőleges Prolog kifejezést (akár egy másik változót) vehet fel értékül – **dinamikus típusfogalom**
 - a változó „first class citizen”, előfordulhat egy struktúra argumentumaként – **logikai változó** **minta $\equiv$ adat**

Néhány alapvető beépített eljárás (Built-In-Procedure, BIP)

• Kifejezések egyesítése

- $X = Y$: az X és Y **szimbolikus** kifejezések egyesítése $\equiv$ azonos alakra hozása változók esetleges behelyettesítésével, a lehető legáltalánosabb módon
- $X \backslash= Y$: az X és Y kifejezések **nem** egyesíthetőek (nem hozhatók azonos alakra)

| ?- $U+V = 1+(2*3)$. $\implies$ $U = 1, V = 2*3$

| ?- $U-V = (8-4)-2$. $\implies$ $U = 8-4, V = 2$

| ?- $U+1 = 4+V$. $\implies$ $U = 4, V = 1$? % Kétirányú a mintaillesztés!

| ?- $U+1 \backslash= V+4$. $\implies$ yes % **szimbolikus** kifejezések, a $+$ nem kommutatív!

• Típusvizsgálat (az elemi, nem bontható típusok aláhúzással jelölve)

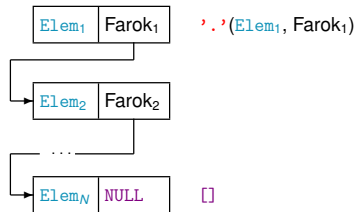
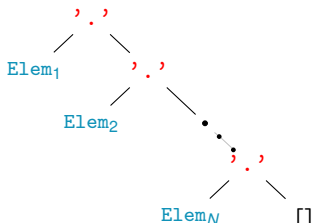
- var(X): X változó, ill. ennek komplementere: nonvar(X): X nem változó
- atomic(X): X konstans
 - number(X): X szám (float(X): X lebegőp., integer(X): X egész)
 - atom(X): X névkonstans
- compound(X): X összetett kifejezés

| ?- $X = 1, \text{atomic}(X), \text{number}(X), \text{integer}(X)$. $\implies$ yes

| ?- $\text{atomic}(X), X = 1$. $\implies$ no (**What rather than How**)

A Prolog lista-fogalma

- A Prolog lista
 - Az üres lista a `[]` névkonstans.
 - A nem-üres lista a `'.' (Fej, Farok)` struktúra:
 - `Fej` a lista feje (első eleme), míg
 - `Farok` a lista farka, azaz a fennmaradó elemekből álló lista.
 - A listákat egyszerűsítve is leírhatjuk („szintaktikus édesítés”).
 - Megvalósításuk optimalizált, időben és helyben is hatékonyabb.
- A listák fastruktúra alakja és megvalósítása



- Az SWI Prolog nem szabványos, a lista-konstruktor nem `'.'`, hanem `'[]'` `:-(((`

Listák jelölése – szintaktikus „édesítőszerek”

- Az alapvető édesítés:
 $.(\text{Fej}, \text{Farok})$ helyett a $[\text{Fej} | \text{Farok}]$ kifejezést írjuk
- Kiterjesztés N darab „fej”-elemre, a skatulyázás kiküszöbölése:
 $[\text{Elem}_1 | \dots | [\text{Elem}_N | \text{Farok}] \dots] \implies [\text{Elem}_1, \dots, \text{Elem}_N | \text{Farok}]$
- Ha a farok $[]$, a „ $| []$ ” jelsorozat elhagyható:
 $[\text{Elem}_1, \dots, \text{Elem}_N | []] \implies [\text{Elem}_1, \dots, \text{Elem}_N]$
- Egy Kif Prolog kifejezés **nyílt végű lista**, ha Kif változó, vagy $\text{Kif} = [_ | \text{Farok}]$ ahol Farok nyílt végű lista (azaz ha előbb-utóbb egy változó található a farokpozíción)

?- $[1,2] = [X Y].$	$\implies$	$X = 1, Y = [2] ?$
?- $[1,2] = [X,Y].$	$\implies$	$X = 1, Y = 2 ?$
?- $[1,2,3] = [X Y].$	$\implies$	$X = 1, Y = [2,3] ?$
?- $[1,2,3] = [X,Y].$	$\implies$	no
?- $[1,2,3,4] = [X,Y Z].$	$\implies$	$X = 1, Y = 2, Z = [3,4] ?$
?- $L = [1 _], L = [_ , 2 _].$	$\implies$	$L = [1,2 _A] ?$ % nyílt végű!
?- $L = .(1, [2,3 []]).$	$\implies$	$L = [1,2,3] ?$
?- $L = [1,2 . (3, [])].$	$\implies$	$L = [1,2,3] ?$

Egy egyszerű listakezelő eljárás

```
% unalmas(Lista, X): Lista minden eleme = X
unalmas([], _X).
unalmas([H|T], X) :-
    H = X,
    unalmas(T, X).
```

```
% Ekvivalens eljárás
unalmas([], _).
unalmas([X|T], X) :-
    unalmas(T, X).
```

```
| ?- unalmas([1,2,3], _).            $\implies$  no
| ?- unalmas([2,2,2], 1).            $\implies$  no
| ?- unalmas([2,2,2], 2).            $\implies$  yes
| ?- unalmas([2,2,2], X).            $\implies$  X = 2 ? ; no
| ?- L=[_,_,_], unalmas(L, 3).       $\implies$  L = [3,3,3] ? ; no
| ?- L=[_,_,_], unalmas(L, X).       $\implies$  L = [X,X,X] ? ; no
| ?- length(L, 10), unalmas(L, X).
     $\implies$  L = [X,X,X,X,X,X,X,X,X,X] ? ; no
| ?- length(L, 10), unalmas(L, X), X = 5.
     $\implies$  L = [5,5,5,5,5,5,5,5,5,5], X = 5 ? ; no
```