SZAKDOLGOZAT

A PROLOG PROGRAMOZÁSI NYELV A KÖNYVTÁROSI MUNKA ASPEKTUSÁBÓL

*Készí*tette: **Túri József Attila**

informatikus könyvtáros szakos hallgató

Témavezető: **Dr. habil Boda István**

tanszékvezető egyetemi docens

Debreceni Egyetem, 2010

Ezúton szeretnék köszönetet mondani Dr. Boda Istvánnak a munkámhoz nyújtott segítségéért és hasznos tanácsaiért, amelyekkel elősegítette szakdolgozatom megírását.

Tartalomjegyzék

BEVEZETÉS

Jelen szakdolgozat célja, hogy bemutassa, hogyan használható a Prolog programozási nyelv a könyvtárosi munka során, milyen jellegzetességei és alkalmazási lehetőségei vannak ennek a logikai programozási nyelvnek.

A Prolog programozási nyelvet Alain Colmerauer fejlesztette ki 1972-ben. Röviddel Colmauer után Szeredi Péter is kifejlesztett egy Prolog interpretert Magyarországon.
A Prolog név a francia eredetű programmation en logique (logikai programozás) kifejezés rövidítése. A Prolog az első logikai programozási nyelvnek tekinthető.

A logikai programozás egy programozási paradigma elnevezése, amelyet az 1970-es években fejlesztettek ki. Ahelyett, hogy egy számítógépes programot egy algoritmus lépésenkénti leírásának tekintenénk, a programot egy logikai elméletként fogjuk fel, az eljárást pedig egy tételnek tekintjük, amelynek igazságát igazolni kell. Ezért egy program végrehajtása egy bizonyítás keresésének felel meg.
Míg a hagyományos (imperatív) programozási nyelvekben a program procedurális leírása, specifikációja annak, hogy a problémát hogyan kell megoldani, a logikai program deklaratív specifikációja annak, hogy mi a megoldandó probléma. A logikai programozás teljesen másfajta gondolkodásmódot feltételez, mint amit megszokhattunk az imperatív paradigmában.
Tekintsük például a programbeli változó fogalmát. Az imperatív nyelvekben a változó egy elnevezés, amely egy meghatározott típusú adatok tárolására alkalmas memóriarekeszre hivatkozik. Miközben a rekesz tartalma időről időre változhat, a változó mindig ugyanarra a rekeszre mutat. Ezzel szemben a logikai program változója egy matematikai értelemben vett változó, ami bármilyen értéket felvehet.

A Prolog sokéves kutatómunka eredménye. A 70-es évek óta a Prologot sikeresen alkalmazzák a robotkutatás, a természetes nyelv megértése, a programhelyesség és a tételbizonyítás területein. Japánban az ötödik generációs számítógép kutatásokat is nagy részben a Prologra alapozzák. Az 1980-as években a nyelv alkalmazási területe rohamosan bővült: építészeti és gépészeti tervező rendszereket, szimulációs programokat és rendkívül sokfajta ún. szakértői rendszert írnak Prologban világszerte. Jelenleg a Prolog az egyik

legfontosabb eszköze a mesterséges intelligenciát alkalmazó programozásnak és a szakértői rendszereknek, hiszen felhasználóbarát és intelligens programozási nyelv. Sokkal közelebb áll az emberi gondolkodáshoz, logikához, ezért a természetes nyelvhez is, mint a többi ún. algoritmikus hagyományos programozási nyelv. Ezért a Prologot gyakran használják mesterségesintelligencia-alkalmazások megvalósítására, illetve a számítógépes nyelvészet eszközeként (különös tekintettel a természetes nyelvfeldolgozásra, melyre eredetileg tervezték), valamint adatbáziskezelő programként.

A szakdolgozat bemutatja a Prolog programozás legfőbb sajátosságait, alapelveit, és alkalmazási lehetőségeit.
A szakdolgozat első része a Prolog programozási nyelv elméleti alapjait ismerteti. Igyekeztem olyan példákon keresztül bemutatni a nyelv sajátosságait, amelyek kapcsolódnak a könyvtárosi munkához. A dolgozat második része a könyvtári munkával kapcsolatos feladatokat és azok megoldásait tartalmazza. Minden feladathoz részletes magyarázat tartozik, így azok számára is megérthető, akik még nem találkoztak korábban ezzel a programozási nyelvvel, de mivel minden feladat tartalmaz új ötleteket, megoldási lehetőségeket, azon olvasók is profitálhatnak belőle, akik már tájékozottabbak a Prolog programozásban. A szakdolgozat harmadik fejezete bemutatja, hogyan lehet logikai fejtörőket megoldani a Prolog segítségével.

1. FEJEZET

1.1. A Prolog programozási nyelv alapjai

A Prolog eredetileg egyszerű felépítésű és könnyen használható nyelv volt. A szintaxis és szemantika leginkább a matematikai logika egy ágának, az elsőrendű logikának a szimbólum- és gondolatvilágára épül, azaz a Prolog nyelv kifejezései általában elsőrendű logikai kifejezéseknek feleltethetőek meg. A Prolog azonban nem tekinthető teljes mértékben az elsőrendű logika számítógépes megvalósításának, azzal csak átfedésben van.

Az informatikában használt nyelvosztályzás szerint a Prolog deklaratív (nem imperatív, nem procedurális) nyelv. Itt a programozónak azt kell leírnia, hogy a megadott adatokból mit kellene kiszámítani (és a Program azt jelzi vissza, lehetséges-e ez); és nem azt az algoritmust, hogy az adatokból hogyan kellene kiszámítani valamit.
Sem eljárások, sem objektumok, sem adattípusok nem vagy nem olyan kifejezetten találhatóak benne, mint más nyelvekben.

A Prolognak is megvan a maga jelkészlete, ábécéje, az ezekből képezett kifejezéseket (némileg pontatlanul) termeknek nevezzük. A Prolog forrásprogramok legtöbb sora egy olyan logikai kifejezés (formula, a Prolog nyelvezetében összetett term – avagy (program)klóz); melyet a program alkalmazása során általában speciális implikációs formulának értelmezünk. A Prolog verziók legtöbbje képes ezeken kívül numerikus és szöveges kifejezések kezelésére is, azonban a logikai (forrás)programok „lelkét” a programklózok alkotják; maga a logikai (forrás)program tehát felfogható úgy, mint egy rendezett elsőrendű klózhalmaz (illetve mint e halmaz klózaiból képezett egyetlen konjunktív normálforma).

1.2. A Prolog program jellemzői

A továbbiakban bemutatjuk a Prolog program legfőbb sajátosságait:
Egy Prolog program egy leszűkített világban működik: azokat és csak azokat a körülményeket veszi figyelembe, amelyeket "megmondtunk" programnak.

A Prolog program mondatok sorozatából áll, amelyeket pont zár le. E mondatokat Prolog **állításoknak** is szoktuk nevezni. Minden mondat egy logikai állítás (vagy formula), amelyeknek igazságértéke van. Kétféle igazságérték létezik: igaz és hamis.

A nagybetűvel írt azonosítók **változót** (például Valaki, X, Y), a kisbetűvel írottak **konstanst** jelölnek (például jokai, petofi). Ezek az állítások **objektumai** vagy tagjai. Amennyiben valamely konstanst nagybetűvel szeretnénk kezdeni, vagy ékezetes betűt szeretnénk alkalmazni benne, akkor a konstanst idézőjelek közé kell tenni (pl. "Jókai", "Petőfi").

A **következtetés** (idegen szóval **implikáció**) tulajdonképpen egy feltételes igazságot rögzít.

A logikai következtetés formájú Prolog állításokat **szabályoknak** nevezzük. A szabályok kisbetűvel kezdődő állításból, egy vagy több konstans vagy változó tagból állnak, pont zárja le őket. A szabályok viszonyt írnak le a tagok között: az első tag van adott viszonyban a másodikkal, azonban ez fordítva nem teljesül. A tagok, vagy **objektumok** számát a kijelentések **argumentumának** nevezzük. A szabályok segítségével a feladatot olyan részfeladatokra bontjuk le, amelyek megoldásából következik az eredeti feladat megoldása. A részfeladatok így a megoldandó feladat (elő)feltételei. A szabály a szabály fejével, a **predikátum névvel** kezdődik. A részfeladatok argumentumlistáiban egy vagy több változó szerepelhet. Az azonos nevű változónevek egy szabályon belül mindig ugyanazt a behelyettesítést jelentik. A szabály teljesülése feltételektől függ. A feltételek az **if** kulcsszó vagy a vele megegyező :- jel után állnak, és több szabály esetén logikai kapcsolatukat is meg kell jelölnünk. Ilyen logikai alapszavak és a vele megegyező írásjelek a következők:

and	illetve	, (vessző)
or	illetve	; (pontosvessző)
not		

Általában a kulcsszavas írásmódot részesítjük előnyben, mert a szavak kifejezőbbek és kevésbé téveszthetők, mint az írásjelek.

Néhány példa:

nagyapja(Nagyapa,Unoka) if apja(Nagyapa,Apa) and apja(Apa,Unoka).

Valaki akkor nagyapja egy személynek, ha apja a személy apjának.

kolcsonzunk if bent(Konyv).

Akkor kölcsönözünk, ha bent van a könyv.

Fontos tudnivaló, hogy a programozó által leírt sorrend szerint értékelődnek ki a tények és szabályok, ezért kötelező az azonos „fejhez” tartozó utasításokat egymásután (egy csoportba, ún. blokkba) írni.

A feltétel nélküli Prolog állításokat **tényállításoknak** (vagy egyszerűen **tényeknek**) nevezzük. A tények olyan egyszerű állítások, amelyek a vizsgált világ dolgai között fennálló kapcsolatokat, összefüggéseket írják le. A tény a kapcsolat, a predikátum nevével kezdődik, ezt a zárójelbe tett argumentumok követik. Az argumentumokban azok az objektumok szerepelnek, amelyek között a kapcsolat fennáll. Az argumentum lehet változó is, amelyet nagybetűvel kezdünk. A tényt pont zárja le.

Tények például a következők:

iro(jokai).

Jókai író.

szereti(juli,krimi).

Juli szereti a krimit.

olvas(pisti,konyv).

Pisti könyvet olvas.

keres(ili,folyoirat).

Ili folyóiratot keres.

tart(janos,ora).

János órát tart.

Az utóbbi példa úgy olvasandó, hogy az első objektum (János) 'tart' relációban van a második objektummal, azaz az órá-val. Ebben az esetben a 'tart' reláció két objektum között teremt kapcsolatot, de ezek száma bármely nemnegatív szám lehet.

Itt mindenképp meg kell jegyeznünk azt, hogy ezeket a szabályokat csak egyféleképpen olvashatjuk: az első objektum áll kapcsolatban a másodikkal. Így a

tart(janos,ora).

nem ugyanaz, mint:

tart(ora,janos).

Ha már vannak tényeink, **kérdéseket** tehetünk fel velük kapcsolatban. Egy Prolog program egy kérdés feltevésével indul. Szintaktikailag a kérdés a tényekhez hasonlít, azonban ez a tényállítás változókat is tartalmazha, amelyek mindig nagybetűvel kezdődnek, például: X, Nev, Tipus, ...
A kérdésre adott válasz vagy Igen/Nem [Yes/No] vagy egy objektumhalmaz azon objektumokkal, amelyek szerepelhetnek az adott relációban.

Nézzük meg e fogalmak megfelelőit egy egyszerű példán! A következő Prolog-programrészlet megadja, hogy az Arany családban kik voltak Arany János szülei.

(1) *apja("Arany Gyorgy","Arany Janos").*

(2) *anyja("Megyeri Sara","Arany Janos").*

(3) *szuloje(X,Y) if apja(X,Y) or anyja(X,Y).*

(4) *szuloje(Valaki,"Arany Janos")*

Az (1)-es, (2)-es és (3)-as a **tudásbázis**, amely a vizsgált problémakört írja le (konkrét apa-gyerek és anya-gyerek kapcsolat, a "szülőség" egy **definíciója**), míg a (4)-es a megoldandó feladat.
Az (1)-es és (2)-es tény, amely kimondja, hogy Arany János apja Arany György és anyja Megyeri Sára.
A (3)-as egy szabály a szülőség definíciójának is tekinthető. Ez az "X szülője Y-nak" állítás teljesülését két feltételre vezeti vissza, amelyek előfeltételnek tekinthetők. A két előfeltétel ezek szerint a következőképp fogalmazható meg:
X akkor szülője Y-nak, ha vagy apja vagy pedig anyja Y-nak.
X,Y és Valaki nem konkrét, hanem tetszőleges elemek, amelyek között az előzőekben leírt kapcsolatok fennállnak. Ezt úgy hozzuk a rendszer tudomására, hogy ezeket nagybetűvel kezdjük.
A (4)-es a feladat, vagyis a megválaszolandó kérdés: kik voltak Arany János szülei? Nyilvánvaló, hogy ez Arany Györgyre és Megyeri Sárára, azaz az őket képviselő "Arany Gyorgy" és "Megyeri Sara" objektumra igaz.

A Prologban egy feladat kitűzése egy **célállítás** megfogalmazását jelenti. Egy célállítást megadhatunk kérdőjellel vagy anélkül.

Ezzel megismerkedtünk a Prolog állítások három osztályával, melyek a következők: szabály, tényállítás, célállítás.

Tehát

Prolog program=tények+szabályok+célállítás

A program végrehajtása egy célállítás bizonyítása.

1.3. Prolog programfutások eredményei

A célkifejezésről előbb-utóbb kideríti a Prolog rendszer, hogy az adott körülmények között teljesül-e vagy nem, illetve milyen feltétellel teljesülhet.

Egy Prolog program **futásának** három eredménye lehet:

1. A feladatot sikerült megoldani. Ha a célállítás változókat is tartalmaz, a Prolog rendszer kiírja a változók értékeit, valamint hogy hány megoldást talált és leáll. Ha a célkifejezésben lévő szabály csak konstansokat tartalmaz, akkor a rendszer addig helyettesítgeti a megengedett tényállításokat a beépített **backtrack** algoritmus segítségével, amíg a szabály nem teljesül, amikoris válaszként közli, hogy Yes. Ellenkező esetben, tehát ha az összes lehetőség kimerült, és még mindig nem volt sikeres a keresés, a válasz No.
2. A feladatot nem sikerült megoldani. A Prolog rendszer kiírja, hogy No és leáll.
3. A program nem áll le. Ekkor vagy végtelen ciklusba esett a Prolog futtató rendszere, vagy olyan bonyolult a feladat, hogy még hosszabb idejű futásra volna szükség a sikeres megoldáshoz.

1.4. A Prolog program szerkezete

A Prolog szerkezetileg három szekcióra bontható:

1. **Deklarációs szekció**, amely két részből áll:
   domains: Elemi (atomi) objektumok típusainak deklarációja.
   predicates: Tények és szabályok neveinek deklarációja a bennük levő elemi objektumokkal.
2. **clauses**: A tények és szabályok felsorolása blokkonként. Lényeges, hogy az azonos nevűek együtt legyenek.

3. **goal**: A cél-szekció. A program céljának rögzítésére szolgál. Itt kell a kérdést megfogalmazni, amiért a program fut.

domains

(az elemi objektumok típusainak deklarációja)

Ez a rész nem kötelező. Ha a szabályok leírásánál csak a rendszer által biztosított alaptípusokkal definiálunk, akkor kihagyható. Ellenkező esetben, különösen a „beszédes nevek elve" betartásakor még inkább nem kerülhetjük meg. Az sem árt, hogy a paraméterek átadásakor ezáltal elkerülhetjük a nem kívánt adatkeveredést is.

Az elemi objektumok nevei kisbetűkkel kezdődnek. A név és egy egyenlőségjel után vagy

- egy már létező (standard)változótípus nevét, vagy
- egy konstanskifejezést

kell tenni, ahogyan az alábbi példák mutatják.

A **standard** típusok a következők:

integer	*egész*
real	*valós*
char	*karakter*
symbol	*szimbólum*
string	*szöveg*

Például:

adat=symbol
darab=integer
szam=real
i,j,k=integer
oldal=bal

Ezek után például az adat szó felhasználható a szabályok megfogalmazásánál olyan pozíciókban, ahol szimbólumokat kívánunk elhelyezni, illetve ahol szimbólumokat várunk eredményként.

Az elmondottak szerint az oldal olyan típus, ahol az egyetlen behelyettesíthető érték a „bal" szó. Ennek persze nem sok értelme lenne, létezik egy olyan lehetőség, ahol egy

típusnév többféle típust jelenthet. Ezt **alternatív deklarációnak** nevezzük. Legegyszerűbben ezt konstansokkal szemléltethetjük:

oldal=bal; jobb

Most az oldal típus kétféle értéket kaphat, ezeket pontosvessző választja el egymástól.

A meglévő típusokkal újabb deklarációk építhetők fel, mint azt az alábbi példa mutatja:

adat=symbol

ujadat=adat

A Prolog nyelvben mód van arra is, hogy egy típust felhasználjunk összetettebb adatszerkezet előállítására. Ennek egyik leggyakrabban használt módja a **lista** definiálása. A lista típust a * segítségével definiáljuk.

*elem=symbol**

Az elem nevű típus olyan adatszerkezet, amelyben symbol típusú elemi objektumok vannak listába rendezve. Ha az a,b,c,d betűkből áll a lista, akkor a következőképp adhatjuk meg:

["a","b","c","d"] vagy ["a"|["b","c","d"]]

Az üres lista jele a [], amely minden lista végére odaértendő.

predikates

(a szabályok felsorolása a formalizmus rögzítésével)

Ebben a részben azokat a szabályokat kell bevezetnünk, amelyeket alkalmazni fogunk. Ezzel rögzítjük az alkalmazott szabályok neveit és a használható paraméter típusait. Legalább egy ilyen szabálynak kell lennie, máskülönben nem beszélhetünk Prolog programról.

Néhány példa:

szabalya(real,integer)

szabalyb(symbol)

szabalyc(real,integer,real)

clauses

(tények és szabályok)

A Prolog program utolsó része a tények és szabályok felsorolása blokkonként. A tények és szabályok felépítésével, megfogalmazásával már korábban foglalkoztunk.

goal

(a program addig fut, amíg ez a cél nem teljesül, vagy meg nem állapítja, hogy sosem teljesülhet)

A goal rész tartalmazza a megoldandó feladatot, azaz azokat a kérdéseket, amelyekre a program futása során kívánunk választ kapni. Ez a rész is elmaradhat, de a léte vagy nemléte erősen befolyásolhatja a program működését.

Ha van, akkor pontosan azt teszi, amire az ezt követő utasítások kényszerítik, és az első kielégítő eset után leáll (hacsak másra nem utasítjuk). Ha nincs, akkor a menüből választott **RUN** parancsra a **DIALOG** ablakban megjelenik a **GOAL**: kiírás, ami után az aktuális cél, más néven a kérdés begépelése következhet. Maga a program tehát csak egy bizonyos kérdés megválaszolásáért fog futni. Ebben az esetben minden megoldást kiír.

Például:

apja(X,"Arany Janos") and apja(X,"Arany Sara").

Ennek a feladatnak a jelentése: van-e olyan X személy, aki egy személyben Arany János és Arany Sára apja.

1.5. A Prolog működése

A Prolog a kérdések megválaszolása során ún. **mintaillesztést** és **visszalépést** alkalmaz, ami azt jelenti, hogy a kérdést és ismert argumentumait (olvas(pisti,_), ahol „_” bármi lehet), mint mintát keresi a tények és szabályok között. A szabályokat lexikális sorrendben veszi figyelembe, és mindig a legelső illeszkedőt választja. Ha nem talál ilyet, visszalép a legutóbbi (jó) elágazásra, mivel azt ezt megjegyezte és onnan kezdi egy újabb ágon az illesztést mindaddig, míg végig nem ment. Ha nem sikerült az illesztés, kiírja, hogy nincs válasz („No”), különben megkapjuk a(z összes) jó illesztést. Tehát olyan „értéket” ad az X-nek, ami a célállítást igazzá teszi.

Ha a célállításban nem változó (X) szerepel, hanem konstans, akkor az egy eldöntendő kérdés.

Például:

goal: olvas(pisti,folyoirat)

Ekkor, ha a mintaillesztés sikeres (vagyis talál ilyen tényt), akkor kiírja, hogy

Yes.

Különben:

No.

Azt az esetet, amikor a célállításban szerepel változó, az alábbi példán mutatjuk be:

domains

s=symbol

predicates

kolto(s)

vanneki(s,s)

clauses

kolto(petofi).

kolto(radnoti).

vanneki(kolto,vers).

E szabályok alapján feltehetjük a kérdést:

goal: kolto(neumann)

goal: kolto(petofi)

goal: kolto(X)

Az első kérdésre a válasz No, a másodikra Yes, a harmadikra X=petofi, X=radnoti.

A válasz a következőképpen született: a Prolog az első esetben megvizsgált minden kolto(...) szabályt. Sehol nem talált egyezést, így a válasz hamis. A második esetben talált egyezést, a válasz igaz. A harmadik esetben a kérdés változót tartalmazott (olyan paraméter, amely még nem kapott értéket). A kolto(...) szabályok vizsgálatánál a Prolog meghatározta, hogy X mely értékeire teljesülhet az állítás. Ez a Prolog működésének egyik kulcsa, a **mintaillesztés**. Ennek során a változók értéket kapnak.

A _ (aláhúzás) olyan érték, amely mindenre illeszthető („akármi"). A kolto(_) értéke Yes.

A tényeken kívül megadhatunk összetettebb szabályokat. Pl. az utolsó szabály helyett:

vanneki(X,vers) if kolto(X).

Ez a szabály azt mondja ki, hogy valaminek akkor van verse, ha arra a valamire teljesül, hogy kolto. A vanneki szabály kiértékeléséhez a Prolog megvizsgálja a kolto szabályokat, és mintaillesztés segítségével egyezést keres.

vanneki(neumann,vers) kiértékelésekor vanneki(X,vers) szabálynál illesztési lehetőséget vesz észre, elvégzi az X=neumann illesztést, majd megvizsgálja kolto(X) feltételt. X most már nem változó, tehát a Prolog a kolto(neumann) állítást értékeli ki. Nem talál egyezést, így a válasz No.

vanneki(A,vers) kiértékelésekor vanneki(X,vers) ismét egyezést mutat, X=A illesztéssel. Így X továbbra is változó marad (mivel A változó), és hal(A) vizsgálata következik. Két megoldást kapunk A-ra.

Az = operátor a következőképp működik:

- konstans = konstans esetben megvizsgálja a két érték egyezését, és igaz vagy hamis eredményt ad.
- változó = konstans esetben mintaillesztőként működik, és a konstanst értékül adja a változónak.
- változó = változó esetben a baloldali paraméternek értékül adja a jobboldalit, mégpedig úgy, hogy a továbbiakban a baloldali helyett a jobboldalit használja (név szerinti paraméterátadás), ld. X=A illesztést előbb. Az = operátor működése nem szimmetrikus, és nem használható konstans = változó formában, ugyanis mindig a baloldalnak adja értékül a jobboldalt.

Látható, hogy az összes jó megoldás meghatározásához a Prolog a **visszalépéses keresés**, azaz a **backtrack** algoritmusát használta: ha egy lépés jó, jobbra lép a következő feltételhez, ha nem, visszalép eggyel, és keresi a következõ jó megoldást.

Az egymás után következő, azonos nevű szabályok egymással "vagy" kapcsolatban állnak. Ha egy szabály kiértékelését a Prolog backtrack kereséssel befejezte, a következő szabályra lép (hogyan teljesülhet még az állítás).

1.6. Tagadás a Prologban

Előfordulhat, hogy úgy találjuk, jobb lenne, ha egy feltétel NEM teljesülne. Ezt a Prologban a not meta-kijelentéssel valósíthatjuk meg. A Prologban a **not** azt jelenti, hogy: "amennyiben a program tényei és szabályai szerint az, hogy ezt a feltételt NEM lehet teljesíteni: IGAZ, akkor a feltétel minden bizonnyal HAMIS". Ez a zárt világ feltételezése.
A **tagadásnál** ha teljesül a belső kifejezés, akkor hamissá válik az érték, míg a belső kifejezés kudarca esetén lesz kielégítve a szabály.

Tekintsünk egy egyszerű példát a tagadásra!

kolcsonzott(X) if konyv(X) and not(bentvan(X)).

Vagyis akkor kölcsönözték ki a könyvet, a nincs bent a könyvtárban.

A **not**-on belül már nem működik a mintaillesztés, csak helyben kiértékelhető állításokat tartalmazhat, vagyis nem tartalmazhat változókat. Ezért a két feltétel sorrendje nem cserélhető fel, mire a kiértékelés a not-ra kerül, X már értéket kapott.

1.7. Levágás és kudarc

A **levágás** (**cut**, jele:"!") azt jelenti, hogy a lehetséges megoldások keresését az adott úton már nem engedjük meg.

Használhatjuk akkor is, ha csak az első megoldásra vagyunk kíváncsiak:

egyetirki(X) if bentvan(X) and !.

Ekkor a könyvtárban lévő könyvek csoportjából csak az elsőt írja ki a program.

A ! egyéb használata: hiányzó feltételek pótlása. Az alábbi példában a kétargumentumú abszolútérték kifejezést definiáljuk.

abs(X,Y): } X>0 esetén: Y=X
egyébként: Y= –X

Ezt Prologban így írhatjuk le:

absz(X,Y) if X>=0 and X=Y or –X=Y.

Ha elkezdjük kipróbálni ezt a definíciót, első eredményeink kedvezők lesznek:
az absz(5,Y) és az abs(–5,Y) kérdésekre a válasz: Y= –5.
Ha azonban folytatjuk a vizsgálódást, és feltesszük az absz(5,–5) kérdést, vagyis hogy az 5 abszolút értéke megegyezik-e az –5-tel, akkor azt látjuk, hogy a válasz igen, ami nem helyes eredmény. Hogyan működött ebben az esetben a Prolog?

A Prolog végrehajtási módszere szerint először az első változat értékelődik ki, és ha itt zsákutcába jutunk, akkor rátérünk a következő változatra. Esetünkben az első változat első feltétele teljesül (5>=0), de a második (miszerint az 5 megegyezik a –5-tel) már nem, tehát semmi akadálya sincs, hogy megpróbáljuk a következő változatot, ami teljesül is, hiszen –5 megegyezik –5-tel. Észrevehetjük, hogy az okozza a problémát, hogy ez a két változatból álló Prolog állítás nem egyenértékű a ha–akkor–egyébként szerkezettel: hiszen a Prologban nemcsak akkor térünk át a második változatra, ha a feltétel, a „ha" rész nem teljesül, hanem akkor is, ha az „akkor" részt sikertelenül hajtottuk végre. Nemcsak akkor van azonban baj az első definícióval, ha az „akkor" rész nem teljesül. Ennek értelmében a pozitív számoknak két abszolút értékük van.

Próbáljuk meg kijavítani ezt a definíciót! Első ötletünk az lehet, hogy a második változatot csak akkor szabad használni, ha X<0, és ezt a változat elején ellenőrizni kell:

absz(X,Y) if X>=0 and X=Y or X<0 and –X=Y.

Ezzel a javítással azonban a rossz megoldást úgy szűrtük ki, hogy miután az első változat során X-ről megállapítottuk, hogy pozitív szám, áttértünk a második változatra, és megvizsgáltuk, vajon X negatív szám-e. Ez felesleges többletmunkát jelent. Meg kellene mondani a rendszernek, hogy ha az X>=0 feltétel teljesül, akkor ezzel már megtaláltuk a jó változatot és nincs már értelme a többi változattal foglalkozni. Itt használjuk a ! (cut) eljárást.
Helyesen:

absz(X,Y) if X>=0 and ! and X=Y or –X=Y.

A **kudarc** (**fail**) – végrehajtása esetén – hamis logikai értéket ad. Ezért a Prolog visszatér az ezt megelőző esethez és újabb jó megoldást keres. Ezt használjuk, ha az összes megoldást ki akarjuk íratni (nemcsak az első helyeset, ahogy Prolog-ban szokás).

Legyen a következő program:

olvas(benedek,ujsag).
olvas(cecil,ujsag).
olvas(denes,ujsag).

Ha a kérdés a következő: (Ki készít házi feladatot?)
goal
olvas(X,ujsag) and write(X) and nl.
akkor a Prolog kiírja a listában az első illeszkedőt: Benedek.

Ha az összes helyes válaszra kíváncsiak vagyunk, akkor „tekintsük" a választ hamisnak, - tehát a megoldást kudarcnak - így visszalép (újabb) jó megoldást keresni. Arra az esetre is fel kell készülnünk, hogy senki sem olvas újságot.
goal
olvas(X,ujsag) and write(X) and nl and fail or
write("Senki sem olvas újságot").

A program az összes megoldást kiírja:
X=benedek
X=cecil
X=denes.

1.8. Rekurzió

Az előzőekben kétféle logikai formulával találkoztunk: tényekkel és szabályokkal. Van egy speciális szabály, ami külön figyelmet érdemel. Az a szabály, amely saját magára vonatkozóan definiál relációt. Az „önhivatkozás" ötlete, amit rekurziónak neveznek, a legtöbb procedurális programnyelvben szintén megtalálható.
A Pascal-szerű procedurális nyelven megírt programrészlet

IF N=0
THEN FAC:=1
ELSE FAC:=N*FAC(N-1)
nem más, mint egy adott szám faktoriálisának kiszámítását végző rekurzív eljárás. Ezekben a nyelvekben azonban az iterációt (az utasítássorozat ciklikus, többszöri, előre definiált számú végrehajtását) általában előnyben részesítik a rekurzióval szemben, mert az iteráció memóriakihasználása gazdaságosabb. A Prologban azonban a rekurzió az egyetlen ciklikus struktúra.

Egy **rekurzív definícióra** tekintsük az alábbi példát:

utod(X,Os) if apa(Os,X).
utod(X,Os) if apa(Valaki,X) and utod(Valaki,Os).

A definíció két szabályból áll, tehát utódnak lenni kétfajta módon lehet: vagy úgy, hogy az ősöm az apám, vagy úgy, hogy az apám egy olyan személy, aki egy ősöm utódja. Azaz az utod feladatot visszavezettük az apa részfeladatra és saját magára, az utod részfeladatra.

Mire kell vigyázni a rekurzív definícióknál?
Elsősorban arra, hogy a rekurzív definíció alapján működő feladatmegoldó algoritmusnak ne csak egy önmagába visszatérő ága legyen, hanem legyen egy ún. leállító ága is.

1.9. Hogyan tanul a Prolog program?

Az eddigiek során láttuk, hogy a Prolog nyelv ahhoz nyújt segítséget, hogy bizonyos tudás, ismeretanyag birtokában következtetéseket vonhassunk le. A tudást állítások formájában adtuk meg, amelyek vagy tények voltak vagy szabályok. Ezeket a programban megadtuk és futása alatt változatlan formában rendelkezésre álltak. Erős megszorítás lenne azonban, ha a program futása során a program tudásanyaga nem bővülhetne, azaz nem tanulhatna a program.

A tanulás kizárólag akkor jöhet létre, ha a domains, predicates és clauses részekhez hozzávesszük a **database** szekciót, amely adatbázis keletkezését írja elő. Az adatbázis egy eleme egy szabály, annak is egy konstans behelyettesítési értékkel bíró esete, vagyis egy tényállítás. Itt az adatbázis formális paramétereit kell megadnunk.

Például:

domains

elem=symbol

database

volt(elem)

Az adatbázis kezdeti tényállításait nem itt, hanem a clauses szekcióban, a szabályok és tények között adjuk meg. Maga az adatbázis folyamatosan változhat, például az első eleme elé vagy az utolsó eleme mögé új tény kerülhet vagy onnan kitörlődhet.

A tanulás során a program az **assert, asserta**, **assertz**, **retract**, **retractall** kifejezéseket használja.

Az **assert**(kijelentés) beilleszti a kijelentést az adatbázisba. Ennek egyik speciális esete az **asserta**(kijelentés), mely hatására a program beilleszti a kijelentést az adatbázisba az azonos nevű megfelelő kijelentések elé. Az **assertz**(kijelentés) szintén beilleszti a kijelentést az adatbázisba, azonban az azonos nevű megfelelő kijelentések mögé. Fontos, hogy a kijelentést mindhárom eljárás használata előtt definiáljuk a **database** adatbázisban.

A **retract** törli az első tényállítást az adatbázisból, ami illeszkedik a retract után zárójelben feltűntetett tényállítással. A **retractall** valamennyi tényállítást törli, amelyek megegyeznek a zárójelben levő tényállítással.

Ez a fajta tanulás, több, mint az adattárolás, mert nem csak tényt, hanem új szabályt is felvehetünk e szabályok paramétereként.

1.10. Hogyan számol a Prolog?

A Prolog számolni is tud. Az adott x tulajdonsággal rendelkező elemek összeszámolását mutatja az alábbi példa:

darab(A) if x(B) and szabad(B) and darab(C) and A=C+1 or A=0.

szabad(B) if volt(B) and ! and fail or assert(volt(B)).

x tulajdonságú elemet veszünk, és megnézzük, hogy szabad-e, vagyis szerepelt-e már az összeszámlálásnál (természetes, hogy egy valamit nem akarunk kétszer megszámolni). Ha volt már, akkor ezt az ágat nem folytatjuk (! and fail), hanem áttérünk a következőre. Ha

még nem volt (azaz szabad(B) igaz), akkor felvesszük az adatbázisba (assert(volt(B)), majd a darab paraméterét megnöveljük eggyel (A=C+1). A paraméter kiindulási értéke 0.

Adott tények összegzése

Ha nem csupán megszámlálni akarunk bizonyos elemeket, hanem adott tényeket akarunk összegezni, akkor is úgy járunk el, mint a fenti esetben, csak nem eggyel, hanem egy adott változó értékével növeljük a paraméter értékét.

osszeg(A) if x(B,Y) and szabad(B) and osszeg(C) and A=C+Y or A=0.

A legtöbb adott tulajdonsággal rendelkező meghatározása

A legtöbb adott tulajdonsággal rendelkező meghatározásához először meg kell határozni a "több" kapcsolatát. Az a maximális, amelynél nincs több adott tulajdonsággal rendelkező elem.

tobb(A) if darab(Z,B) and B>A.
maxdarab(Y) if darab(Y,A) and not(tobb(A)).

1.11. Adatok beolvasása

Az adatok beolvasása a **readln**, a **readint**, a **readchar** és a **readreal** utasításokkal történik.
A **readchar**(változó) egyetlen karaktert olvas be a billentyűzetről a paraméterbe.
A **readint**(változó) egy egész számot olvas be a billentyűzetről a paraméterbe.
A **readln**(változó) karaktereket olvas be a billentyűzetről a paraméterbe.
A **readreal**(változó) egy valós számot olvas be a billentyűzetről a paraméterbe.

1.12. Az eredmény kijelzése

Az eredmények kiíratására a **write** kifejezést használhatjuk, néhány „segédeszközzel”. Ezek a beépített (már ismertetett) kifejezések: levágás vagy **cut**(**!**), kudarc vagy **fail**, az **nl** (newline), ami soremelést hajt végre. Soremelést eredményez a **\n** is, melyet a **write** után zárójelen és idézőjelen belül, a kiírandó szöveg után alkalmazunk.
A **clearwindow** utasítással töröljük az éppen aktív szöveges ablakot.

A **makewindow** kifejezéssel és pozíció megadásával akár teljes képernyős szövegfelírást alkalmazhatunk. A **makewindow** a képernyő egy területét ablaknak definiálja. Az első paraméter az ablak megkülönböztető számát jelöli, amellyel lehet választani az ablakok között. A második paraméter adja meg az ablakon belül megjelenő szöveg színének kódját. Ha a harmadik paraméter értéke nem 0, akkor keretet rajzol a jelölt terület köré, és a felső vonalba foglalja a fejléc szövegét, amit idézőjel között adunk meg. A következő két paraméter értékei az ablak bal felső sarkának pozíciói a teljes képernyőterületen, míg az utolsó két paraméter az ablak magassága és szélessége. Fontos, hogy ezek a paraméterek összhangban legyenek a képernyőmérettel.
Klasszikus példa: a program számokat kér be, kiírja a reciprokukat, kivéve a 0-t, mert akkor leáll.

```
predicates
        megold
clauses
        megold if readint(X) and not(X=0) and Z=1/X and write(Z) and nl and megold.
goal
        clearwindow and makewindow(1,11,75,"Reciprok",1,0,25,80) and megold.
```

1.13. Nyomkövetés

Ha lépésenként szeretnénk futtatni a programot, akkor írjuk be a program legelejére, a domains rész elé a **trace** utasítást. A futtatás az F10 billentyűvel történik, amelyet valamennyi lépésnél le kell nyomnunk. Az egyes lépések eredményét a trace ablakban láthatjuk.

2. FEJEZET

A második fejezet olyan feladatokat tartalmaz, melyek már kevés elméleti ismeret birtokában is megoldhatók, amelyekben egyszerű adatszerkezetekkel, tényekkel, szabályokkal találkozunk.

2.1. Könyv

A feladat

Kati könyvtárba indul. Milyen könyvet kölcsönözzön ki? Természetesen csak olyan könyvet kölcsönözhet, ami bent van a könyvtarban és érdekes. Ha nem tud megfelelő könyvet kölcsönözni, akkor otthon marad és leveszi a polcról azt a könyvet, ami van.

A feladat megoldása

Ebben a programban Kati kétféleképp juthat olvasnivalóhoz: vagy elmegy a könyvtárba és kiválaszt egy könyvet, vagy otthon marad és azt a könyvet olvasa, ami van.

A két lehetőséget a következőképp írhatjuk le Prolog nyelven:

```
olvasas if kolcsonoz(Konyv) and nl and write("Menj a konyvtarba es kolcsonozz ")
 and  write(Konyv) and write("t!") and nl.
olvasas if otthon_marad and nl and write("Maradj otthon es ") and
 write("olvasd, ami van!") and nl.
```

Ahhoz, hogy ki tudjuk választani a megfelelő könyvet, tudnunk kell, melyik könyv az, ami érdekes és bent is van a könyvtárban.

```
kolcsonoz(Konyv) if erdekes(Konyv) and bentvan(Konyv).
```

A program ismeri Kati ízlését, és számon tartja a számára érdekes könyveket.

```
erdekes(krimi).
```

erdekes(scifi).

erdekes(regeny).

erdekes(novella).

Ezenkívül azt is tudja, hogy pontosan mely könyvek találhatók meg a könyvtárban.

bentvan(regeny).

bentvan(verses).

bentvan(novella).

A program tudásához az is hozzátartozik, hogy otthon maradni minden feltétel nélkül lehet. Ebben a programban egy változó van, ez a Konyv. Ne felejtsük el, hogy nagybetűvel írjuk!

A program tartalmaz goal részt, ami azt jelenti, hogy a futtatás során ezt hajtja végre a rendszer. Először a **clearwindow** utasítás hatására letörli a képernyőt, majd a **makewindow** ablakot rajzol a megadott paraméterekkel.

clearwindow and makewindow(1,11,15,"Konyvtar",1,0,23,80) and
write("Kati konyvtarba indul. Milyen konyvet kolcsonozzon ki? ") and nl and
write("Megmondja a program.") and nl and
write("Termeszetesen csak olyan konyvet kolcsonozhet, ami") and nl and
write("bent van a konyvtarban es erdekes.") and nl and
write("Ha nem tud megfelelo konyvet kolcsonozni, akkor otthon marad") and nl and
write("es leveszi a polcrol azt a konyvet, ami van.") and nl and
readchar(W) and olvasas and fail.

A teljes program

*/*Kati konyvtarba indul.*
Milyen konyvet kolcsonozzon ki? Megmondja a program.
Termeszetesen csak olyan konyvet kolcsonozhet, ami bent van a konyvtarban es erdekes.
Ha nem tud megfelelo konyvet kolcsonozni,
akkor otthon marad es leveszi a polcrol azt a konyvet, ami van./*

domains

mi=string.

```
predicates
        kolcsonoz(mi).
        erdekes(mi).
        bentvan(mi).
        otthon_marad.
        olvasas.

clauses
/*vagy elmegy a konyvtarba konyvet kolcsonozni, vagy otthon marad*/
        olvasas if kolcsonoz(Konyv) and nl and write("Menj a konyvtarba es kolcsonozz ")
         and  write(Konyv) and write("t!") and nl.
        olvasas if otthon_marad and nl and write("Maradj otthon es ") and
         write("olvasd, ami van!") and nl.

/*akkor kolcsonoz konyvet, ha erdekes es bent van*/
        kolcsonoz(Konyv) if erdekes(Konyv) and bentvan(Konyv).

/*az erdekes konyvek:*/
        erdekes(krimi).
        erdekes(scifi).
        erdekes(regeny).
        erdekes(novella).

/*a bent levo konyvek:*/
        bentvan(regeny).
        bentvan(verses).
        bentvan(novella).
        otthon_marad.

goal
        clearwindow and makewindow(1,11,15,"Konyvtar",1,0,23,80) and
        write("Kati konyvtarba indul. Milyen konyvet kolcsonozzon ki? ") and nl and
        write("Megmondja a program.") and nl and
```

write("Termeszetesen csak olyan konyvet kolcsonozhet, ami") and nl and
write("bent van a konyvtarban es erdekes.") and nl and
write("Ha nem tud megfelelo konyvet kolcsonozni, akkor otthon marad") and nl and
write("es leveszi a polcrol azt a konyvet, ami van.") and nl and
readchar(W) and olvasas and fail.

A program működése: mintaillesztés, visszalépés

Nézzük meg, hogyan történik a könyvtáras feladat megoldása Prologban! Nyomon fogjuk követni azt az utat, amelyet a Prolog rendszer automatikusan bejár.

A Prolog rendszer betöltése után be kell olvasni a programot. A beolvasás rendszerint szintaktikai ellenőrzéssel folyik, ami azt jelenti, hogy a rendszer nyelvileg ellenőrzi a programot. Hibaüzeneteket kapunk, ha a Prolog állítások formája nem felel meg az éppen aktuális megvalósítás szintaktikai követelményeinek. A program indítása az alt+R billentyűkombinációval (**RUN**) történik. Mivel van goal rész, ezzel kezdődik a program. A **clearwindow** hatására letörli a képernyőt, a **makewindow** utasítás segítségével ablakot hoz létre és a **write** paranccsal kiírja az általunk idézőjelek között megadott szöveget és az **nl** hatására sort emel.

Ezután következik a feladat, az olvasas. Ha a program nem rendelkezi goal résszel, nekünk kell begépelnünk egy célállítást, jelen esetben ez a
goal: olvasas

A Prolog következtetési mechanizmusa megkeresi az olvasas definícióját, és megpróbálja a feladatot egyeztetni a definícióval. Az egyeztetést az ún. **mintaillesztéssel** hajtja végre. A feladatunkban az olvasas nulla argumentumú predikátum, tehát az egyeztetés problémamentes. Felmerülhet a kérdés, hogy egy definíció két állítása közül melyiket válasszuk, hiszen vannak olyan definíciók, melyek még több Prolog állításból állnak. A válasz egyszerű: A Prolog a lehetséges állítások közül mindig az elsőt próbálja illeszteni, és ha ez a mintaillesztés szabályai szerint nem sikerül, akkor megpróbálja a következőt illeszteni és így tovább. Esetünkben tehát a
goal: olvasas
célállítás és az

olvasas if kolcsonoz(Konyv) and nl and write("Menj a konyvtarba es kolcsonozz ") and write(Konyv) and write("t!") and nl.

szabály automatikusan illeszkedik. Ennek a szabálynak a jobb oldala feltételként hat elemi állítást tartalmaz. Ezt úgy is mondhatjuk, hogy a csokolade feladatot csak úgy tudjuk megoldani, ha megoldjuk a kolcsonoz(Konyv) részfeladatot, és megoldjuk a soremelés (**nl**), és a kiírás (**write**) részfeladatait az adott sorrendben. Tehát egy feladat megoldását részfeladatok megoldására vezettük vissza.

Először az első, a kolcsonoz(Konyv) részfeladattal kell foglalkoznunk. Eljárásunk ugyanaz, mint az eredeti feladatnál, keresünk hozzá egy definíciót:

kolcsonoz(Konyv) if erdekes(Konyv) and bentvan(Konyv).

amely megint egy szabály két feltétellel. Ez a feladat egy egy-argumentumú predikátumot tartalmaz, így itt a mintaillesztés már lényeges lehet. A mintaillesztés itt természetesen sikerül, hiszen a vasarol predikátumnak egy argumentuma van és az mindkét elemi állításban változó. Így a kolcsonoz(Konyv) feladatot két részfeladatra bontottuk:
az első: erdekes(Konyv) és
a második: bentvan(Konyv).

Az erdekes(Konyv) részfeladatot a megfelelő definíció első Prolog állításával, az

erdekes(krimi).

tényállítással illeszthetjük. A krimi konstans, a Konyv változó, így az illesztés sikerülni fog. Ilyen esetekben, amikor egy változó és egy konstans illeszkedik, az illesztés után a változó felveszi a konstans értékét. Így történik a Prologban az értékátadás, hiszen értékadó utasítás – a hagyományos nyelvekkel ellentétben – nincs. Tehát a Konyv változó felveszi a krimi értéket. Azonban a Konyv változónak további előfordulásai is vannak a kolcsonoz definíciójában:

kolcsonoz(Konyv) if erdekes(Konyv) and bentvan(Konyv).

szerepel a bal oldalon, továbbá a jobb oldalon mindkét elemi állításban. A változók hatásköre a Prologban egy Prolog állítás. Ez azt jelenti, hogy ha a program futása során egy változó egyik előfordulása felvesz egy értéket, akkor ugyanazon az állításon belüli összes többi előfordulás is automatikusan felveszi ugyanazt az értéket.
Így tehát, amikor vesszük a következő részfeladatot, az ott szereplő Konyv változó helyett már a krimi értéket kell figyelembe vennünk, azaz ezt a részfeladatot kell megoldanunk:

bentvan(krimi).

Ez azonban nem fog sikerülni, mivel a bentvan definíciójában sehol sem szerepel a krimi, és konstans csak ugyanazzal a konstanssal illeszkedik. Márpedig ha egy feladat egyik részfeladatát nem sikerül megoldani, akkor a teljes feladat sem oldható meg. Itt le is állna a Prolog következtetési rendszere, és kiírná, hogy nem sikerült a feladatot megoldani, ha nem rendelkezne a **visszalépés** tulajdonságával.

Visszalépésre akkor kerül sor, ha az egyik részfeladat megoldása nem sikerül, de egy másik, előzőleg megoldott részfeladatot több módon is meg lehet oldani, és még van ki nem próbált megoldási lehetőség. Ehhez persze az kell, hogy visszalépjünk a feladat megoldásának folyamatában, állítsuk vissza a programot egy korábbi állapotára, felejtsük el a közben történt értékátadásokat, és próbáljunk ki egy új utat.

A második esetben hasonlóképp járunk. A harmadikban viszont sikerünk lesz:
Az

erdekes(Konyv)

részfeladat megoldására most próbálkozzunk a definíció harmadik tényállításával:

erdekes(regeny).

Ekkor a mintaillesztés után a Konyv felveszi a regeny értéket, és a

bentvan(regeny).

részfeladat is megoldható.
Ezzel teljesítettük a kolcsonoz definíció mindkét feltételét, így a kolcsonoz(Konyv) is megoldódott, mégpedig úgy, hogy a Konyv felvette a regeny értéket.

Az **nl** beépített eljárás vagy predikátum hatására soremelés történik, a **write**(„Menj a konyvtarba es kolcsonozz”) hatására a rendszer kiírja az idézőjelek közötti szöveget. A **write**(Konyv) kiírja a rendszer a Konyv változó aktuális értékét (most a regeny). A **write**(„t!”) kiírja az idézőjelek közötti szöveget, az **nl** hatására pedig sort emel.

Ezzel még mindig nem vagyunk kész, mivel a **goal** részben ezután még szerepe a **fail** szó, amely visszalépést eredményez. Miután a Prolog rendszer sikeresen megoldotta a feladatot, kiírja, hogy regényt kell kölcsönöznünk, majd pedig megpróbálja megoldani a **fail** részfeladatot. Ez nem sikerül neki, és a rendszer úgy érzi, hogy a teljes feladat megoldása

is kudarcba fulladt. Ekkor visszalép, és megpróbálja a részfeladatot egy másik úton megoldani. Ez megint sikerül, és ki is írja, hogy novellát is kölcsönöözzünk, de utána a **fail**-lel találkozva újból kudarc éri. A Prolog rendszer kétségbeesetten állandóan visszalép, és új meg új megoldásokat állít elő, amíg csak lehetséges. Úgy is mondjuk, hogy bejárja a teljes keresési fát, keresi azt a megoldást, amitől a rákövetkező részfeladat esetleg megoldható lesz. Esetünkben, miután a rendszer nem talál több olyan könyvet, amit kikölcsönözhetünk, kiírja, hogy „Maradj otthon es olvasd, ami van!”. Ezután kimerült az összes lehetőség és a rendszer leáll.

2.2. Családfa

A feladat

Az informatikus könyvtáros munkája során gyakran találkozik történelmi könyvekkel. A következő program szintén történelmi témájú: az Árpád-házon belüli kapcsolatokat rögzíti, illetve definiálja. A program segítségével a megadott apa-gyermek és anya-gyermek viszonyon kívül további családi kapcsolatokra lehet fényt deríteni.

A feladat megoldása

Először rögzítjük, hogy ki kinek az apja. Például:

```
apja(almos,arpad).
apja(arpad,levente).
apja(arpad,tarhos).
apja(arpad,solt).
```

Felsoroljuk, hogy ki kinek a felesége. Például:

```
felesege(geza,sarolta).
felesege(istvan,gizella).
felesege(orseolo,"leany 2").
felesege(laszlo,premiszlava).
```

Beírjuk a fejedelmek nevét és uralkodásuk idejét.

```
fejedelem(almos,819,895).
fejedelem(arpad,889,907).
fejedelem(solt,907,947).
fejedelem(taksony,947,970).
fejedelem(geza,970,997).
fejedelem(istvan,997,1001).
```

Ugyanezt tesszük a királyokkal, például:

kiraly(istvan,1001,1038).

kiraly(peter,1038,1041).

kiraly(peter,1044,1046).

kiraly(endre,1046,1060).

kiraly(bela,1060,1063).

Minden más kapcsolatot ezek segítségével határozunk meg:

Valaki akkor anyja egy gyermeknek, ha az apjának a felesége.

anyja(Anya,Gyerek) if apja(Apa,Gyerek) and felesege(Apa,Anya).

Valaki akkor szülője egy gyermeknek, ha vagy anyja vagy apja a gyermeknek.

szuloje(Szulo,Gyerek) if apja(Szulo,Gyerek) or anyja(Szulo,Gyerek).

Valaki nagyszülője egy gyermeknek, ha szülője valakinek, aki szülője a gyermeknek.

nagyszulo(Nagyszulo,Gyerek) if szuloje(Nagyszulo,Szulo) and szuloje(Szulo,Gyerek).

A leszármazott-ős kapcsolat a legnehezebb. Valaki leszármazottja egy személynek, ha vagy az illető személy a szülője, vagy a szülője leszármazottja az illető személynek.

ose(Os,Utod) if szuloje(Os,Utod) or szuloje(Os,Valaki) and ose(Valaki,Utod).

Valaki akkor királyné, ha a férje király.

kiralyne(No) if felesege(Ferfi,No) and kiraly(Ferfi,_,_).

Valaki testvére egy személynek, ha van közös szülőjük.

testvere(X,Y) if szuloje(Szulo,X) and szuloje(Szulo,Y) and not(X=Y).

A teljes program

```
/*A program az Arpad-hazon beluli  kapcsolatokat mutatja be.*/

domains
        valaki=string.
        evszam=integer.
```

```
predicates

apja(valaki,valaki).
felesege(valaki,valaki).
fejedelem(valaki,evszam,evszam).
 kiraly(valaki,evszam,evszam).

anyja(valaki,valaki).
szuloje(valaki,valaki).
nagyszuloje(valaki,valaki).
ose(valaki,valaki).
kiralyne(valaki).
testvere(valaki,valaki).

clauses

/*ki kinek az apja*/
        apja(almos,arpad).
        apja(arpad,levente).
        apja(arpad,tarhos).
        apja(arpad,solt).
        apja(solt,taksony).
        apja(taksony,geza).
        apja(geza,istvan).
        apja(istvan,imre).
        apja(geza,judit).
        apja(geza,"leany1").
        apja(geza,"leany2").
        apja(orseolo,peter).
        apja(taksony,mihaly).
        apja(mihaly,vazsoly).
        apja(vazsoly,endre).
        apja(endre,salamon).
```

```
        apja(endre,adelhaid).
        apja(vazsoly,bela).
        apja(bela,"I. geza").
        apja("I. geza",kalman).
        apja(bela,"I. laszlo").
        apja(bela,eufemia).
        apja(vazsoly,levente).
        apja(mihaly,laszlo).

/*ki kinek a felesege*/
        felesege(geza,sarolta).
        felesege(istvan,gizella).
        felesege(orseolo,"leany 2").
        felesege(laszlo,premiszlava).
        felesege(endre,anasztazia).
        felesege(bela,rixa).
        felesege(salamon,judit).
        felesege(vratislav,adelhaid).
        felesege("I. geza","görög nő").
        felesege(kalman,buzilla).
        felesege(kalman,eufemia).
        fejedelem(almos,819,895).
        fejedelem(arpad,889,907).
        fejedelem(solt,907,947).
        fejedelem(taksony,947,970).
        fejedelem(geza,970,997).
        fejedelem(istvan,997,1001).

        kiraly(istvan,1001,1038).
        kiraly(peter,1038,1041).
        kiraly(peter,1044,1046).
        kiraly(endre,1046,1060).
        kiraly(bela,1060,1063).
        kiraly(salamon,1063,1074).
```

kiraly("I. geza",1074,1077).

kiraly("I. laszlo",1077,1095).

kiraly(kalman,1095,1116).

*/*anyja, ha az apjanak a felesege*/*

anyja(Anya,Gyerek) if apja(Apa,Gyerek) and felesege(Apa,Anya).

*/*akkor szuloje, ha apja vagy anyja*/*

szuloje(Szulo,Gyerek) if apja(Szulo,Gyerek) or anyja(Szulo,Gyerek).

*/*akkor nagyszuloje, ha van olyan ember, akinek a nagyszulo szuloje es aki a gyerek szuloje*/*

nagyszuloje(Nagyszulo,Gyerek) if szuloje(Nagyszulo,Szulo) and

szuloje(Szulo,Gyerek).

*/*ose, ha szuloje vagy ha a szuloje leszarmazottja az illeto szemelynek*/*

ose(Os,Utod) if szuloje(Os,Utod) or szuloje(Os,Valaki) and ose(Valaki,Utod).

*/*kiralyne, ha a kiraly felesege*/*

kiralyne(No) if felesege(Ferfi,No) and kiraly(Ferfi,_,_).

*/*testvere, ha van kozos szulojuk*/*

testvere(X,Y) if szuloje(Szulo,X) and szuloje(Szulo,Y) and not(X=Y).

A program működése

Ez a program nem tartalmaz **goal** részt, így a nekünk kell begépelnünk mindazt, amit tudni szeretnénk az Árpád-házi uralkodókkal kapcsolatban.

Először megnézzük, hogy Géza szülője-e Istvánnak. Csupán ennyit kell begépelnünk:

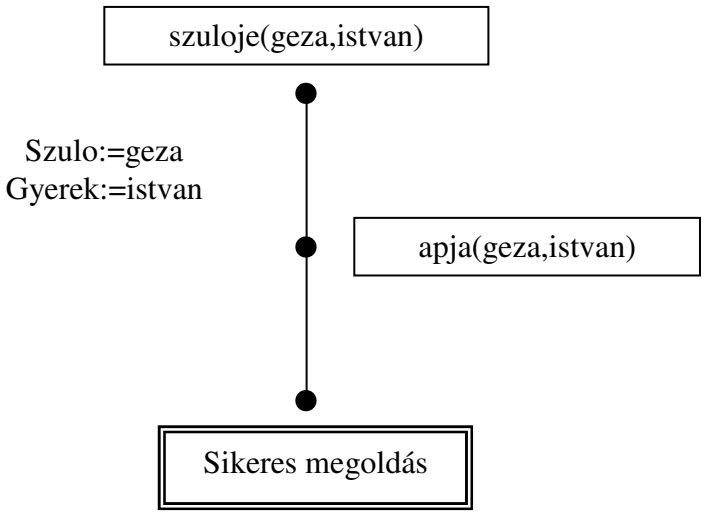
szuloje(geza,istvan)

A szuloje meghatározása

szuloje(Szulo,Gyerek) if apja(Szulo,Gyerek) or anyja(Szulo,Gyerek).

alapján a Prolog rendszer a következőképp fog működni: a Szulo változó illeszkedik a geza konstanssal, a Gyerek az istvan konstanssal. Az első rész meghívásakor – apja(geza,Gyerek) – mivel van olyan tényállításunk, ahol az első argumentum geza –

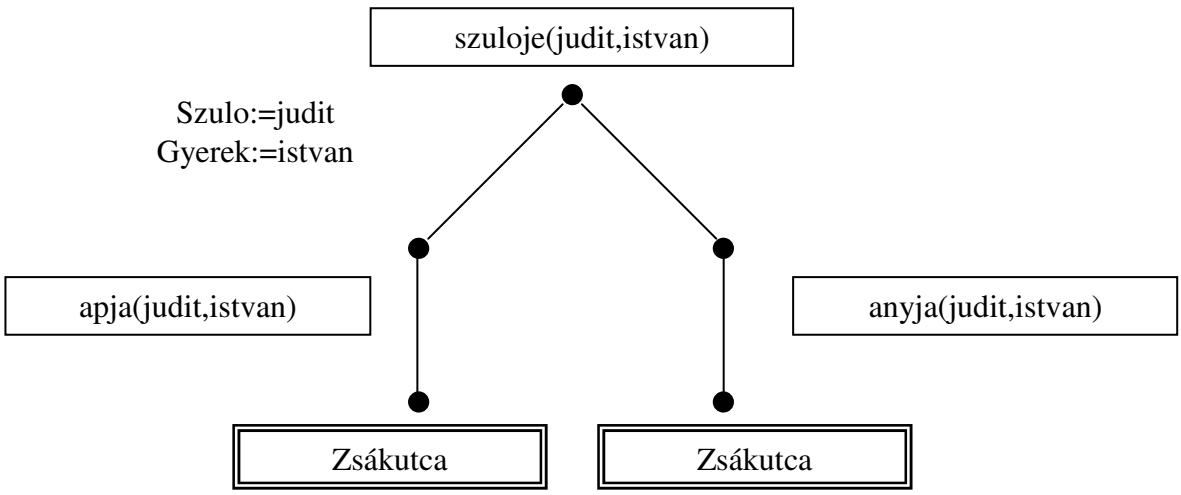
apja(geza,istvan)– az illesztés sikeres, és a Gyerek változó felveszi az istvan értéket. A rendszer úgy érzi, hogy sikeresen oldotta meg a feladatot, kiírja, hogy Yes és leáll.
A megoldás menetét az alábbi ábra szemlélteti:



A

szuloje(judit,istvan)

célállítás ugyanolyan kérdést tesz fel, mint az első, de itt a válasz negatív lesz, azaz Judit nem szülője Istvánnak. A Szulo=judit és a Gyerek=istvan értékadások után sem az apja, sem az anyja részfeladatot nem sikerül megoldani, mivel a kriszta konstans egyiknek sem szerepel az első argumentumában. Így a rendszer azt írja ki, hogy No és leáll.



Ha arra vagyunk kíváncsiak, hogy kik Géza szülei, egyszerűen begépeljük:

szuloje(X,geza)

Ekkor a célállítás egyik argumentumában változó szerepel, a másikban konstans. Ekkor az a feladat, hogy keressük meg azt az értéket (vagy azokat az értékeket), amelyek mellett a célállítás predikátuma igaz lesz. Esetünkben olyan személyt keresünk, akire vonatkozóan a szuloje igaz lesz, ha a második argumentum geza. A válasz:

X=taksony

1 Solution

tehát a program tudása szerint Gézának egy szülője van: Taksony.

Most vegyünk egy olyan állítást, ahol a szuloje első argumentuma konstans, és a második változó:

szuloje(geza,X)

Ki Géza gyermeke? Vagy: Kik Géza gyermekei? – attól függően, hogy hány van neki. Tehát a szuloje definícióval egyszerre határoztuk meg, hogy ki kinek a szülője és ki kinek a gyermeke.

Esetünkben a válasz:

X=istvan

X=judit

X=leany1

X=leany2

4 Solutions

Itt négy megoldása is van a feladatnak.

A következő példánkban arra vagyunk kíváncsiak, hogy ki Géza nagyszülője. Ekkor az alábbiakat kell begépelnünk:

nagyszuloje(X,geza)

Ekkor az alábbi megoldást kapjuk:

X=soltr

1 Solution

Ha viszont arra vagyunk kíváncsiak, hogy Géza kinek a nagyapja, akkor ezt gépeljük be:

nagyszuloje(geza,X)

Ekkor az Imre nevet kapjuk megoldásként.

Valaki ősének a meghatározásakor **rekurzív** definíciót használunk. A definíció két szabályból áll, tehát ősnek lenni kétfajta módon lehet: vagy úgy, hogy az ősöm valamelyik szülőm, vagy úgy, hogy valamelyik szülőm olyan személy, aki egy ősöm utódja.
Arra kérdezünk rá, hogy Álmos őse-e Gézának.

ose(almos,geza)

A program futása úgy kezdődik, hogy a célállítást illesztjük az első szabály bal oldalával. Illesztés után Os=almos és Utod=geza, mivel az ose definíciója így szól:

ose(Os,Utod) if szuloje(Os,Utod) or szuloje(Os,Valaki) and ose(Valaki,Utod).

Az illesztés után a szuloje(almos,geza) részfeladat megoldására kerül sor. Ha igaz lenne, hogy Álmos Géza szülője, akkor a program Yes válasszal leállna. Azonban nem igaz sem az apja(almos,geza), sem az anyja(almos,geza).
Mivel ezeket a részfeladatokat nem sikerült megoldani, a rendszer visszalép, és most az ose definíciójának második szabályával foglalkozik, tehát keres valakit, akinek Álmos a szülője, ehhez megnézi, hogy van-e valaki, akinek Álmos az apja, apja(almos,_). Megtalálja Árpádot, mivel az apja(almos,arpad) a kapcsolatok között szerepel. Ekkor megnézi, hogy Árpád Géza őse-e. Ez akkor teljesül például, ha szuloje(arpad,geza). Ez sem igaz, így keres valakit, akinek Árpád az apja. A program több lépést jár be hasonlóképpen, amíg eljut az apja(taksony,geza) kapcsolatig, ami igaz. Ezzel a feladatot megoldottuk, azaz igazoltuk, hogy Álmos Géza őse. A rendszer kiírja, hogy Yes és leáll.

A továbbiakban megnézzük, hogy Ámos fejedelem mikor uralkodott. Ehhez be kell írnunk, hogy

fejedelem(almos,X,Y)

A rendszer kiírja Álmos uralkodásának kezdetét és végét, azaz X=819, Y=895.

A következő példában a királynékat szeretnénk kiíratni. Ekkor az alábbiakat gépeljük be:

kiralyne(X)

Ekkor a következő megoldást írja ki a rendszer:
X=Gizella
X=Anasztazia
X=rixa

X=judit
X=görög nő
X=buzilla
X=eufemia
7 Solutions

Vajon ki volt István testvére? A kérdés megválaszolásához csupán az alábbiakat kell beírni:

testvere(geza,X)

és rögtön megkapjuk a választ: X=mihaly.

Számos hasonló kérdésre találhatunk választ az Árpád-házi uralkodókkal kapcsolatban a Prolog program segítségével.

2.3. Terv1

A feladat

A következő program segítségével tervezzünk könyvtárat! Először egy olyat, amely két egybenyíló helyiségből áll. A két termet különböző irányokból lehet csatlakoztatni, és az ajtók és ablakok elhelyezésének változtatásával lehet különböző változatokhoz jutni.
Az alábbi igényeket kell figyelembe vennünk a tervezés során:
Északon se ajtó, se ablak ne legyen.
1 falon ne legyen ajtó is és ablak is.
A két ablak ne legyen egymással szemben levő falon.

A feladat megoldása

A feladatban szereplő könyvtárat a bejárati ajtóval, a két terem között elhelyezkedő átjáróval és a két ablakkal jellemezzük. Ezek irányának megadása jelenti a feladat megoldását. Azt a termet, mely bejárati ajtóval rendelkezik, bejaratosnak neveztük el. Ezt a bejárati ajtó, a két terem közötti átjáró és egy ablak jellemez. A másik termet értelemszerűen teremnek nevezzük, ezt egy átjáró és egy ablak jellemez. A szemben(Atjaro,At) azt jelenti, hogy a bejárati ajtóval rendelkező terem átjáró ajtajának iránya legyen szemben a másik átjárójának irányával, vagyis a két terem illeszkedjen egymáshoz. A két terem ablaka pedig ne legyen egymással szemben levő falon, ahogy ezt kikötöttük.

konyvtar(Ajto,Atjaro,Ablak1,Ablak2) if bejaratos(Ajto,Atjaro,Ablak1) and
terem(At,Ablak2) and szemben(Atjaro,At) and
not(szemben(Ablak1,Ablak2)).

A bejárati ajtóval rendelkező terem legyen olyan terem, mint a másik, ezenkívül a bejárati ajtó ne nézzen északra, az ajtó iránya ne egyezzen meg sem az ablakéval, sem az átjáróéval, azaz egy falon ne legyen ajtó is és ablak is.

bejaratos(Ajto,Atjaro,Ablak) if terem(Atjaro,Ablak) and irany(Ajto) and
Ajto<>"eszak" and Ajto<>Ablak and Ajto<>Atjaro.

A terem meghatározása egy előre meghatározott tulajdonságot tartalmaz, miszerint az ablakok nem nézhetnek északra, és egy falon nem lehet ajtó is és ablak is.

terem(Ajto,Ablak) if irany(Ajto) and irany(Ablak) and Ablak<>"eszak"
and Ablak<>Ajto.

Az irányokat a négy égtáj segítségével adjuk meg.

irany("eszak").
irany("kelet").
irany("nyugat").
irany("del").

Megadjuk az egymással ellentétes irányokat is.

szemben("eszak","del").
szemben("del","eszak").
szemben("kelet","nyugat").
szemben("nyugat","kelet").

A teljes program

```
domains
        hol=string.
predicates
        konyvtar(hol,hol,hol,hol).  /*a konyvtarat a bejarati ajtoval, a ket terem kozotti atjaroval es a ket ablakkal jellemezzuk*/
        bejaratos(hol,hol,hol).  /*itt van a bejarati ajto*/
        terem(hol,hol).  /*ez a masik terem*/
        szemben(hol,hol).  /*egymassal ellentes iranyok*/
        irany(hol).  /*iranyok megadasa*/
clauses
/*ket terem van, amelyek illeszkednek egymassal, ket ablak nincs egymassal szemben levo falon*/
        konyvtar(Ajto,Atjaro,Ablak1,Ablak2) if
                bejaratos(Ajto,Atjaro,Ablak1) and
                terem(At,Ablak2) and
```

```
        szemben(Atjaro,At) and
        not(szemben(Ablak1,Ablak2)).
/*a bejarati ajtoval rendelkezo terem olyan, mint a masik, a bejarati ajto ne nezzen
eszakra, az ajto iranya ne egyezzen meg sem az ablakeval, sem az atjaroeval, azaz egy
falon ne legyen ajto is es ablak is*/
    bejaratos(Ajto,Atjaro,Ablak) if
        terem(Atjaro,Ablak) and
        irany(Ajto) and
        Ajto<>"eszak" and
        Ajto<>Ablak and
        Ajto<>Atjaro.
/*az ablakok nem nezhetnek eszakra, egy falon nem lehet ajto is es ablak is*/
    terem(Ajto,Ablak) if
        irany(Ajto) and
        irany(Ablak) and
        Ablak<>"eszak" and
        Ablak<>Ajto.
/*iranyok megadasa*/
    irany("eszak").
    irany("kelet").
    irany("nyugat").
    irany("del").
/*ellentetes iranyok*/
    szemben("eszak","del").
    szemben("del","eszak").
    szemben("kelet","nyugat").
    szemben("nyugat","kelet").

goal
    clearwindow and makewindow(1,14,15,"Terv1",1,0,23,80) and
    write("Konyvtarat akarunk tervezni, amely ket terembol all. \n") and
    write("Az alabbi igenyeket kell figyelembe vennunk: \n") and
    write("Eszakon se ajto, se ablak ne legyen.\n") and
    write("1 falon ne legyen ajto is es ablak is.\n") and
```

```
write("A ket ablak ne legyen egymassal szemben levo falon.\n\n") and
readchar(W) and
konyvtar(Ajto,Atjaro,Ablak1,Ablak2) and
write("Megoldas:\n\n") and
write("Az ajto iranya: ",Ajto) and nl and
write("az atjaroe: ",Atjaro) and nl and
write("az egyik ablak iranya:",Ablak1) and nl and
write("a masik ablake:  ",Ablak2).
```

2.4. Terv2

A feladat

A program három egybenyíló helyiségből álló könyvtárat. A három termet különböző irányokból lehet csatlakoztatni, és az ajtók és ablakok elhelyezésének változtatásával lehet különböző változatokhoz jutni.

Az alábbi igényeket kell figyelembe vennünk a tervezés során:
- A termek egyenes mentén helyezkedjenek el,
- a középsőn legyen bejárati ajtó,
- ajtó és ablak nem nézhet északi irányba,
- egy falon csak egy valami (ajtó vagy ablak) lehet,
- a középső termen egy, a másik kettőn két ablak legyen.

A feladat megoldása

A feladat hasonlít az előző példánkhoz. A könyvtárat ezúttal a bejárati ajtóval, a termek között elhelyezkedő átjárókkal és az öt ablakkal jellemezzük. Ezek irányának megadása jelenti a feladat megoldását. Azt a termet, mely bejárati ajtóval rendelkezik, újból bejaratosnak neveztük el. Ezt a bejárati ajtó, a két terem közötti átjáró és egy ablak jellemez. A másik két termet itt is teremnek nevezzük, ezt egy-egy átjáró és két-két ablak jellemez. A szemben(Atjaro1,At1) stb. definíciók azt jelentik, hogy az adott terem átjáró ajtajának iránya legyen szemben a másik terem átjárójának irányával, vagyis a két terem illeszkedjen egymáshoz. Mivel esetünkben három terem van, így a feltételek bővülnek.

konyvtar(Ajto,Atjaro1,Atjaro2,Ablak1,Ablak2,Ablak3,Ablak4,Ablak5) if
bejaratos(Ajto,Atjaro1,Atjaro2,Ablak1) and
terem(At1,Ablak2,Ablak3) and terem(At2,Ablak4,Ablak5) and
szemben(Atjaro1,At1) and szemben(Atjaro2,At2) and
szemben(At1,At2).

A bejárati ajtóval rendelkező teremben a bejárati ajtó és az ablak ne nézzen északra, az ajtó iránya ne egyezzen meg sem az ablakéval, sem az átjárókéval, tehát egy falon csak egy valami lehet.

bejaratos(Ajto,Atjaro1,Atjaro2,Ablak1) if irany(Ajto) and
irany(Ablak1) and irany(Atjaro1) and irany(Atjaro2) and
Ajto<>"eszak" and Ajto<>Ablak1 and Ajto<>Atjaro1 and
Ajto<>Atjaro2 and Atjaro1<>Atjaro2 and Ablak1<>"eszak" and
Ablak1<>Atjaro1 and Ablak1<>Atjaro2.

A terem meghatározása rögzíti, hogy az ablakok nem nézhetnek északra. Az egyik ablak iránya nem egyezhet meg a másik ablakéval, valamint egyik ablak iránya sem egyezhet meg az ajtók irányával, azaz egy falon csak egy valami lehet.

terem(Ajto,Ablak2,Ablak3) if irany(Ajto) and irany(Ablak2) and
irany(Ablak3) and Ablak2<>"eszak" and Ablak3<>"eszak" and
Ablak2<>Ablak3 and Ablak2<>Ajto and Ablak3<>Ajto.

Az irányokat a négy égtáj segítségével adjuk meg.

irany("eszak").
irany("kelet").
irany("nyugat").
irany("del").

Megadjuk az egymással ellentétes irányokat is.

szemben("eszak","del").
szemben("del","eszak").
szemben("kelet","nyugat").
szemben("nyugat","kelet").

A teljes program

```
domains
        hol=string.

predicates
        konyvtar(hol,hol,hol,hol,hol,hol,hol,hol). /*a konyvtarunkat bejarati ajto, a ket atjaro es az ot ablak jellemzi*/
        bejaratos(hol,hol,hol,hol).  /*a bejarati ajtoval rendelkezo terem*/
        terem(hol,hol,hol).  /*a masik ket terem*/
        szemben(hol,hol).  /*egymassal ellentetes iranyok*/
        irany(hol).  /*iranyok megadasa*/

clauses
/*harom termunk van, amelyek illeszkednek egymashoz*/
        konyvtar(Ajto,Atjaro1,Atjaro2,Ablak1,Ablak2,Ablak3,Ablak4,Ablak5) if
                bejaratos(Ajto,Atjaro1,Atjaro2,Ablak1) and
                terem(At1,Ablak2,Ablak3) and
                terem(At2,Ablak4,Ablak5) and
                szemben(Atjaro1,At1) and
                szemben(Atjaro2,At2) and
                szemben(At1,At2).
/*a bejarati ajtos teremben bejarati ajto es az ablak ne nezzen eszakra, egy falon csak egy valami lehet*/
        bejaratos(Ajto,Atjaro1,Atjaro2,Ablak1) if
                irany(Ajto) and
                irany(Ablak1) and
                irany(Atjaro1) and
                irany(Atjaro2) and
                Ajto<>"eszak" and
                Ajto<>Ablak1 and
                Ajto<>Atjaro1 and
                Ajto<>Atjaro2 and
                Atjaro1<>Atjaro2 and
```

```
        Ablak1<>"eszak" and
        Ablak1<>Atjaro1 and
        Ablak1<>Atjaro2.
/*a teremben az ablakok nem nezhetnek eszakra, egy falon csak egy valami lehet*/
    terem(Ajto,Ablak2,Ablak3) if
        irany(Ajto) and
        irany(Ablak2) and
        irany(Ablak3) and
        Ablak2<>"eszak" and
        Ablak3<>"eszak" and
        Ablak2<>Ablak3 and
        Ablak2<>Ajto and
        Ablak3<>Ajto.
/*iranyok megadasa*/
    irany("eszak").
    irany("kelet").
    irany("nyugat").
    irany("del").
/*ellentetes iranyok rogzitese*/
    szemben("eszak","del").
    szemben("del","eszak").
    szemben("kelet","nyugat").
    szemben("nyugat","kelet").

goal
    clearwindow and makewindow(1,15,15,"Terv2",1,0,23,80) and
    write("Konyvtarat akarunk tervezni, amely harom terembol all. \n") and
    write("Az alabbi igenyeket kell figyelembe vennunk: \n") and
    write("A termek egyenes menten helyezkednek el,\n") and
    write("a kozepson van bejarati ajto,\n") and
    write("ajto es ablak nem nezhet eszaki iranyba\n") and
    write("egy falon csak egy valami (ajto vagy ablak) lehet,\n") and
    write("a kozepso teremben egy, a masik ketton ket ablak van.\n\n") and
    readchar(W) and
```

```
konyvtar(Ajto,Atjaro1,Atjaro2,Ablak1,Ablak2,Ablak3,Ablak4,Ablak5) and
write("Megoldas:\n\n") and
write("Az ajto iranya: ",Ajto) and nl and
write("az egyik atjaro iranya: ",Atjaro1) and nl and
write("a masik atjaroe: ",Atjaro2) and nl and
write("az elso ablak iranya: ",Ablak1) and nl and
write("a masodik ablake:  ",Ablak2) and nl and
write("a harmadik ablake: ",Ablak3) and nl and
write("a negyedik ablake: ",Ablak4) and nl and
write("az otodik ablake: ",Ablak5).
```

2.5. Szomszéd

A feladat

Könyvtári adatbázisunkban adott az olvasók neve és címe. A program megadja két ember nevét, akik szomszédjai egymásnak.

A feladat megoldása

Először rögzítjük az egyes személyek adatait: nevüket és címüket: a várost, az utcát és a házszámot.

cim(anita,budapest,kakukk,2).
cim(benjamin,budapest,kakukk,4).
cim(cecil,debrecen,cinke,2).
cim(david,debrecen,kakukk,6).

Ahhoz, hogy áttekinthető legyen a program, szükséges, hogy kilistázzuk az ismert személyek adatait. Ehhez nyújt segítséget a kiir szabály, amely sorra veszi a cim-ben megadott adatokat, kiírja ezeket, sort emel, majd a **fail** hatására a részfeladat megoldása "kudarcba fullad", és a program visszalép, új megoldásokat keresve vagy pedig a **! (cut)** miatt leáll. Ha nem alkalmaznánk a **fail** és a **!** eljárásokat, a rendszer csupán az első nevet írná ki.

kiir if cim(N,V,U,H) and write(N," ",V," ",U," ",H) and nl and fail or !.

Definiáljuk, mikor szomszédja két ember egymásnak! Akkor, ha ugyanabban a városban, ugyanabban az utcában laknak és kettővel tér el a házszámuk (mivel vagy páros vagy páratlan oldalban laknak).

szomszed(X,Y) if cim(X,V1,U1,H1) and cim(Y,V2,U2,H2) and
X<>Y and V1=V2 and U1=U2 and H1=H2-2.
szomszed(X,Y) if cim(X,V1,U1,H1) and cim(Y,V2,U2,H2) and
X<>Y and V1=V2 and U1=U2 and H1=H2+2.

A teljes program

```
domains
        nev,varos,utca=string.
        hazszam=integer.

predicates
        cim(nev,varos,utca,hazszam).  /*az illeto cime, ami tartalmazza a nevet, a varost, az utcat es a hazszamot*/
        kiir.  /*kilistazza az egyes szemelyek nevet es teljes cimet*/
        szomszed(nev,nev).  /*megadja a szomszedokat*/

clauses
        cim(anita,budapest,kakukk,2).
        cim(benjamin,budapest,kakukk,4).
        cim(cecil,debrecen,cinke,2).
        cim(david,debrecen,kakukk,6).
        kiir if cim(N,V,U,H) and write(N,"  ",V,"  ",U,"  ",H) and nl and fail or !.

/*szomszedok, ha ugyanabban a varosban, ugyanabban az utcaban laknak es kettovel ter el a hazszamuk*/
        szomszed(X,Y) if cim(X,V1,U1,H1) and cim(Y,V2,U2,H2) and
        X<>Y and V1=V2 and U1=U2 and H1=H2-2.
        szomszed(X,Y) if cim(X,V1,U1,H1) and cim(Y,V2,U2,H2) and
        X<>Y and V1=V2 and U1=U2 and H1=H2+2.

goal
        makewindow(1,13,15," Szomszed ",0,0,25,80) and
        clearwindow and nl and nl and
        write("Konyvtari adatbazisunkban adott nehany olvaso neve es cime.\n") and
        kiir and nl and
        write("Az alabbi program megadja ket ember nevet, akik szomszedjai egymasnak.\n\n") and
        readchar(W) and
```

szomszed(X,Y) and write("szomszedok: ",X,", ",Y) or
write("nincsenek szomszedok").

2.6. Külföldi

A feladat

Könyvtárunkban lehetőség nyílik könyvtárközi kölcsönzésre is. Külföldről is lehet könyveket rendelni. Programunk a különböző városok közötti közvetlen összeköttetések leírását tartalmazza. A feladat az, hogy az indulási állomásból a célállomásba olyan útvonalat találjunk, amely nem érinti ugyanazt a várost, így a szükséges könyv eljuthat a megfelelő helyre.

A feladat megoldása

A különböző városok közötti közvetlen összeköttetéseket tényekkel írjuk le. Például:

kapcsolat(budapest,varso).

Ez azt jelenti, hogy Budapest és Varsó között közvetlen kapcsolat van.

Az

osszekottetes(X,Y) if kapcsolat(X,Y) or kapcsolat(Y,X).

összefüggés azt mondja ki, hogy X város és Y város között van összeköttetés, ha X-ből el lehet jutni Y-ba, vagy ha Y-ból el lehet jutni X-be.

Ha létezik közvetlen út X és Y város között, akkor készen is vagyunk.

ut(X,Y) if osszekottetes(X,Y).

Ha nem, akkor az is lehet, hogy úgy is el tudunk juttatni a könyvet X városból Y városba, hogy közben más várost, illetve városokat is érintünk. Vigyázni kell azonban arra, hogy a köztes várost (nevezzük ezt Z-nek) ne érintsük többször is. Ezt a problémát oldjuk meg a szabad kifejezéssel, csak akkor engedélyezzük az utat Z városba, ha az szabad, azaz még nem jártunk ott.

ut(X,Y) if osszekottetes(X,Z) and szabad(Z) and ut(Z,Y).

Ha már voltunk Z-ben, akkor hamis eredményt ad ez a részfeladat, ha még nem jártunk ott, akkor a program felveszi Z-t az adatbázisunkba.

szabad(Z) if volt(Z) and ! and fail.

szabad(Z) if assert(volt(Z)).

Fő kérdésünk, hogy eljuthat-e a könyv egyik városból a másikba. A feladat megkezdése előtt fontos, hogy „tiszta lappal” kezdjük, ezért kiürítjük az adatbázist, mielőtt bármivel is foglalkoznánk. Ezután felvesszük az adatbázisba az elindulásunk helyszínét és ezután jöhet a keresés. Fontos, hogy a start és a célállomás ne egyezzen meg, különben nincs értelme a feladatnak.

eljut(X,Y) if retractall(volt(_)) and assert(volt(X)) and ut(X,Y) and

X<>Y.

A programunkat úgy tudjuk tesztelni, hogy lekérdezzük az egyes városok közötti kapcsolatokat. A program nem tartalmaz **goal** részt, ezért nekünk kell begépelni a **DIALOG** ablakba a szükséges utasításokat. Az Alt+R-rel tudunk átlépni ebbe az ablakba.
Ha például arra vagyuk kíváncsiak, hogy el tudjuk-e juttatni a könyvet Moszkvából Budapestre, mindössze csak annyit kell beírnunk, hogy

eljut(moszkva,budapest)

A program kiírja, hogy Yes, azaz lehetséges a könyvtárközi kölcsönzés.
Amennyiben azt szeretnénk megtudni, hogy melyek azok a városok, amelyek Budapesttel állnak összeköttetésben, akkor be kell írni, hogy

eljut(budapest,X),

és a program valamennyi várost felsorolja, amellyel Budapestről könyvtárközi kölcsönzést lehet folytatni.

A teljes program

```
domains
        varos=string.
predicates
        kapcsolat(varos,varos).        /*a kulonbozo varosok kozotti kozvetlen kapcsolatok*/
        ut(varos,varos).
        osszekottetes(varos,varos).
        szabad(varos).
        eljut(varos,varos).            /*eljuthat-e a konyv egyik varosbol a masikba - erre
```

```
                                        vagyunk kivancsiak*/
database
        volt(varos).
clauses
        kapcsolat(budapest,varso).
        kapcsolat(budapest,london).
        kapcsolat(london,peking).
        kapcsolat(london,boston).
        kapcsolat(varso,moszkva).
        kapcsolat(madrid,roma).
/*oda-vissza ervenyes*/
        osszekottetes(X,Y) if kapcsolat(X,Y) or kapcsolat(Y,X).
/*van ut ket varos kozott, ha van kozvetlen osszekottetes vagy ha van olyan varos, amit
erintve el tudjuk juttatni a konyvet oda, ahova szeretnenk*/
        ut(X,Y) if osszekottetes(X,Y).
        ut(X,Y) if osszekottetes(X,Z) and szabad(Z) and ut(Z,Y).
/*Jartunk-e mar az adott varosban? Nem akarunk ketszer ugyanoda visszaterni*/
        szabad(Z) if volt(Z) and ! and fail.
        szabad(Z) if assert(volt(Z)).
/*A kerdes: eljut-e a konyv egyik varosbol a masikba?*/
        eljut(X,Y) if retractall(volt(_)) and assert(volt(X)) and ut(X,Y) and
                X<>Y.
```

2.7. Könyvtár

A feladat

A könyvtári katalógusunk tartalmazza az egyes könyvek adatait. A program elsőként kilistázza a könyveket, majd megadja azokat a szerzőket, akiknek több könyvük is van, azokat, akiknek csak egy könyvük található a könyvtárban, illetve megadja azt a szerzőt, akinek a legtöbb könyve van a katalógus szerint.

A feladat megoldása

Elsőként rögzítjük a könyvek helyrajzi számát, szerzőjét és címét, pl.:

konyv(1,"Jokai Mor","Az arany ember").

konyv(2,"Jokai Mor","Fekete gyemantok").

A kiadás adatait, pl.:

kiadas(1,"Europa K.","Budapest",1996).

kiadas(2,"Szepirodalmi K.","Budapest",2005).

Az egyes könyvekhez tartozó tárgyszavakat, pl.:

targyszo(1,"regeny","szepirodalom").

targyszo(2,"regeny","szepirodalom").

A feladat első része, hogy a program írja ki az egyes könyvek adatait. Ezt Prologban a következőképpen valósíthatjuk meg:

```
listaz if konyv(X,Y,Z) and kiadas(X,A,B,C) and
targyszo(X,D,E) and
write("Helyrajzi szam: ",X,"  ") and
write("Szerzo: ",Y,"  ") and
write("Cim: ",Z,"  ") and
write("Kiado: ",A,"  ") and
write("Kiadas helye: ",B,"  ") and
write("Kiadas eve: ",C,"  ") and
write("Targyszavak: ",D,", ",E) and nl and fail or !.
```

Hogy ne csak egy könyvet írjon ki a rendszer, ezért a **fail** utasítást használjuk, így kudarcot észlel a rendszer és újabb megoldásokat keres. Ha már nem talál újabbat, akkor a **!** (**cut**) hatására befejeződik ez a részfeladat. Ekkor már igaz értékkel tér vissza.

A program kiírja azokat a szerzőket, akiknek több könyvük is van. A könyveket ugyanaz a szerző írta (X), a címük viszont nem egyezik meg (Y nem ugyanaz, mint Z).

tobbkonyv(X) if konyv(_,X,Y) and konyv(_,X,Z)
and Y<>Z.

Megadja azokat a szerzőket, akiknek csak egy könyvük van. Ezt a tobbkonyv(X) tagadásával érhetjük el.

egykonyv(X) if konyv(_,X,_) and not(tobbkonyv(X)).

A program megszámolja, hogy hány könyve van a szerzőnek. Itt az egyes szerzők könyveinek címét vizsgáljuk. Megnézzük, hogy szerepelt-e már a cím az összeszámlálásnál, mivel nem akarjuk kétszer megszámolni. Ha volt már, akkor ezt az ágat nem folytatjuk, hanem áttérünk a következőre. Ha még nem volt, akkor felvesszük az adatbázisba (assert(marvolt(Z)), majd a darab paraméterét megnöveljük eggyel. A paraméter kiindulási értéke 0.

szamol(Y,DB) if szerz(Y) and konyv(_,Y,Z) and
not(marvolt(Z)) and assert(marvolt(Z)) and
szamol(Y,Db2) and Db=Db2+1 and ! or Db=0.
szamfo(Y,DB) if retractall(marvolt(_)) and
szamol(Y,DB).

Megadja azt a szerzőt, akinek a legtöbb könyve van bent a katalógus szerint. Ezt a részfeladatot azonban nem tudjuk közvetlenül megoldani, előbb azt kell meghatároznunk, hogy melyek azok a szerzők, akiknek „nem a legtöbb” könyvük van, majd ezt tagadjuk.

nemlegtobb(Y) if szerz(Y) and szerz(B) and
konyv(_,Y,_) and konyv(_,B,_)
and Y<>B and szamfo(Y,Db1) and szamfo(B,Db2)
and Db1<Db2.

legtobb(Y) if konyv(_,Y,_) and not(nemlegtobb(Y)) and ! or fail.

A teljes program

```
domains
        szerzo,cim,kiado,hely,targy=string
        hszam,evszam,db=integer

predicates
        konyv(hszam,szerzo,cim).
        kiadas(hszam,kiado,hely,evszam).
        targyszo(hszam,targy,targy).
        tobbkonyv(szerzo).
        egykonyv(szerzo).
        listaz.
        szamol(szerzo,db).
        szamfo(szerzo,db).
        legtobb(szerzo).
        nemlegtobb(szerzo).
        szerz(szerzo).

database
        marvolt(cim).

clauses
        /*A konyvek szerzoje es cime*/
        konyv(1,"Jokai Mor","Az arany ember").
        konyv(2,"Jokai Mor","Fekete gyemantok").
        konyv(3,"Petofi Sandor","Janos vitez").
        konyv(4,"Petofi Sandor","A helyseg kalapacsa").
        konyv(5,"Mikszath Kalman","A beszelo kontos").
        konyv(6,"Bolyai Janos","Appendix").
        konyv(7,"Johann Sebastian Bach","Adagio e Fuga").
        konyv(8,"Jokai Mor","A koszivu ember fiai").
```

```
/*A kiadas adatai*/
kiadas(1,"Europa K.","Budapest",1996).
kiadas(2,"Szepirodalmi K.","Budapest",2005).
kiadas(3,"Nagykonyv K.","Nyiregyhaza",2009).
kiadas(4,"Akkord","Talentum",2000).
kiadas(5,"Mozaik","Szeged",2002).
kiadas(6,"Reprint","Szeged",2002).
kiadas(7,"Editio Musica","Budapest",1962).
kiadas(8,"Europa","Budapest",2008).

/*Targyszavak*/
targyszo(1,"regeny","szepirodalom").
targyszo(2,"regeny","szepirodalom").
targyszo(3,"elbeszelo koltemeny","kepeskonyvek gyermekeknek").
targyszo(4,"elbeszelo koltemeny","szepirodalom").
targyszo(5,"regeny","szepirodalom").
targyszo(6,"matematika","geometria").
targyszo(7,"nyomtatott kotta","kamarazene trio").

/*A szerzok*/
szerz("Jokai Mor").
szerz("Petofi Sandor").
szerz("Mikszath Kalman").
szerz("Bolyai Janos").
szerz("Johann Sebastian Bach").

/*Kiirja az egyes konyvek adatait*/
listaz if konyv(X,Y,Z) and kiadas(X,A,B,C) and
targyszo(X,D,E) and
write("Helyrajzi szam: ",X,",  ") and
write("Szerzo: ",Y,",  ") and
write("Cim: ",Z,",  ") and
write("Kiado: ",A,",  ") and
write("Kiadas helye: ",B,",  ") and
```

```
        write("Kiadas eve: ",C,"  ") and
        write("Targyszavak: ",D,", ",E) and nl and fail or !.

        /*Azok a szerzok, akiknek tobb konyvuk is van*/
        tobbkonyv(X) if konyv(_,X,Y) and konyv(_,X,Z)
           and Y<>Z.

        /*Azok a szerzok, akiknek csak egy konyvuk van*/
        egykonyv(X) if konyv(_,X,_) and not(tobbkonyv(X)).

        /*Megszamolja, hany konyve van a szerzonek*/
        szamol(Y,DB) if szerz(Y) and konyv(_,Y,Z) and
                not(marvolt(Z)) and assert(marvolt(Z)) and
                szamol(Y,Db2) and Db=Db2+1 and ! or Db=0.
        szamfo(Y,DB) if retractall(marvolt(_)) and
                        szamol(Y,DB).

        nemlegtobb(Y) if szerz(Y) and szerz(B) and
                        konyv(_,Y,_) and konyv(_,B,_)
                        and Y<>B and szamfo(Y,Db1) and szamfo(B,Db2)
                        and Db1<Db2.

        /*Az a szerzo, akinek a legtobb konyve van*/
        legtobb(Y) if konyv(_,Y,_) and not(nemlegtobb(Y)) and ! or fail.

goal
        makewindow(1,2,15," Konyvtar ",0,0,25,80) and
        clearwindow and nl and nl and
        write("Konyvtarunkban az alabbi konyvek talalhatok meg: \n") and
        listaz and nl and
        write("Azok a szerzok, akiknek tobb konyvuk is van:\n") and
        readchar(W) and
        tobbkonyv(Y) and write(Y) and nl and
        write("Azok a szerzok, akiknek csak egy konyvuk van:\n") and
```

readchar(Q) and
egykonyv(V) and write(V) and nl and
write("Az a szerzo, akinek a legtobb konyve van:\n") and
readchar(T) and
legtobb(Z) and write(Z) and nl and
write("Konyveinek szama:\n") and
readchar(P) and
szamfo(Z,DB) and write(DB).

2.8. Megye

A feladat

Könyvtárunkban műveltségi vetélkedőt tartanak. Az egyik feladat földrajzi jellegű. A program segít a vetélkedő résztvevőinek a feladatok megoldásában. A feladatok a következők:
Magyarország megyéiről tudjuk, hogy melyik melyiknek a szomszédja.

a) Nevezz meg egy megyét, amely nem szomszédos Heves megyével!
b) Mondj egy megyét, amelyik nem szomszédja sem Somogy sem Pest megyének!
c) Adj meg egy megyét, amelyiknek a legkevesebb szomszédja van!

A feladat megoldása

A szomszédos megyéket a következőképp rögzítjük:

szomszedja(nev,nev)

Akkor beszélünk megyéről, ha vagy szomszédja valaminek, vagy valami a szomszédja.

megye(X) if szomszedja(X,_) or szomszedja(_,X).

A kiírás itt bonyolultabb, mint az előzőekben. A korábbi módszert itt nem alkalmazhatjuk, mert akkor egy megyét többször is kiírna a rendszer. Meg kell mondanunk neki, hogy csak akkor írjon ki egy megyét, ha az még nem szerepelt.

kiir if retractall(voltmar(_)) and kiiras.
kiiras if megye(X) and not(voltmar(X)) and write(X," ") and
assert(voltmar(X)) and kiiras or !.

Az a) kérdés megválaszolásához előbb meghatározzuk, hogy mikor szomszédos egymással két megye. Ennek tagadásával kapunk választ arra, hogy mely megyék nem szomszédosak egymással.

szomszedos(X,Y) if szomszedja(X,Y) or szomszedja(Y,X).
kerdesa(X,Y) if megye(X) and megye(Y) and X<>Y and

not(szomszedos(X,Y)).

A b) kérdés két megyéhez megad egy olyat, amelyiknek egyik sem a szomszédja. Ehhez újból az adatbázisunkat használjuk.

elokeszit(X,Y,Z) if megye(Z) and X<>Z and not(szomszedos(X,Z)) and
Y<>Z and not(szomszedos(Y,Z)) and szabad(Z).
szabad(Z) if voltmar(Z) and ! and fail or asserta(voltmar(Z)).
kerdesb(X,Y,Z) if retractall(voltmar(_)) and elokeszit(X,Y,Z).

A harmadik feladathoz meg kell határoznunk egy megyét, amelynek a legkevesebb szomszédja van. Itt előbb minden megyénél meg kell határoznunk a szomszédjai számát, majd meg kell adnunk, hogy mely megyéknek van nem a legkevesebb szomszédja. Ennek tagadásával kapjuk a feladat megoldását.

szomszedszam(M,DB) if szomszedos(M,M1) and not(voltmar(M1)) and
asserta(voltmar(M1)) and szomszedszam(M,DB1) and DB=DB1+1
and ! or DB=0.
foszomszedszam(M,DB) if retractall(voltmar(_)) and
szomszedszam(M,DB).
nemlegkevesebbszomszed(M) if szomszedos(M,_) and szomszedos(M1,_)
and M<>M1 and foszomszedszam(M,DB) and
foszomszedszam(M1,DB1) and DB>DB1.
legkevesebb_szomszedja_van(M) if megye(M) and
not(nemlegkevesebbszomszed(M)).
ckerdes(M) if legkevesebb_szomszedja_van(M).

A goal részben az kerdesa és a kerdesb kérdéseknél konstans paramétert, illetve paramétereket alkalmazunk, így oldjuk meg a megadott feladatokat.

kerdesa(X,heves)

kerdesb(somogy,pest,Y)

A teljes program

```
domains
        nev=string
        szam=integer

predicates
        szomszedja(nev,nev)  /*szomszedos megyek*/
        kiir      /*megyek kiiratasa*/
        kiiras
        megye(nev)
        szomszedos(nev,nev)  /*az a) kerdeshez*/
        kerdesa(nev,nev)
        elokeszit(nev,nev,nev)  /*a b) kerdeshez*/
        szabad(nev)
        kerdesb(nev,nev,nev)
        szomszedszam(nev,szam)  /*a c) kerdeshez*/
        nemlegkevesebbszomszed(nev)
        foszomszedszam(nev,szam)
        legkevesebb_szomszedja_van(nev)
        ckerdes(nev) /*csak az erthetoseg kedveert*/
database
        voltmar(nev)
clauses
        szomszedja(borsod_abauj_zemplen,heves).
        szomszedja(heves,nograd).
        szomszedja(nograd,pest).
        szomszedja(pest,fejer).
        szomszedja(fejer,veszprem).
        szomszedja(veszprem,vas).
        szomszedja(vas,zala).
        szomszedja(zala,somogy).
```

```
szomszedja(baranya,tolna).
szomszedja(hajdu_bihar,bekes).
szomszedja(szabolcs_szatmar_bereg,hajdu_bihar).
szomszedja(szabolcs_szatmar_bereg,borsod_abauj_zemplen).
szomszedja(csongrad,bekes).
szomszedja(somogy,tolna).
szomszedja(hajdu_bihar,jasz_nagykun_szolnok).
szomszedja(jasz_nagykun_szolnok,bekes).
szomszedja(jasz_nagykun_szolnok,pest).
szomszedja(jasz_nagykun_szolnok,heves).
szomszedja(jasz_nagykun_szolnok,csongrad).
szomszedja(pest,komarom_esztergom).
szomszedja(fejer,bacs_kiskun).
szomszedja(pest,bacs_kiskun).
szomszedja(tolna,bacs_kiskun).
szomszedja(baranya,bacs_kiskun).
szomszedja(csongrad,bacs_kiskun).
szomszedja(jasz_nagykun_szolnok,bacs_kiskun).
szomszedja(komarom_esztergom,fejer).
szomszedja(komarom_esztergom,veszprem).
szomszedja(komarom_esztergom,gyor_sopron_moson).
szomszedja(fejer,tolna).
szomszedja(baranya,somogy).
szomszedja(gyor_sopron_moson,veszprem).
szomszedja(zala,veszprem).
szomszedja(gyor_sopron_moson,vas).
megye(X) if szomszedja(X,_) or szomszedja(_,X).
/*a megyek kiirasa*/
kiir if retractall(voltmar(_)) and kiiras.
kiiras if megye(X) and not(voltmar(X)) and write(X,"  ") and
        assert(voltmar(X)) and kiiras or !.
/* a) kerdes: megad egy megyet, ami egy nem szomszedos egy adott megyevel*/
szomszedos(X,Y) if szomszedja(X,Y) or szomszedja(Y,X).
kerdesa(X,Y) if megye(X) and megye(Y) and X<>Y and not(szomszedos(X,Y)).
```

```
/* b) kerdes: ket megyehez megad egy olyat, amelyiknek egyik sem a szomszedja*/
        elokeszit(X,Y,Z) if megye(Z) and X<>Z and not(szomszedos(X,Z)) and
                Y<>Z and not(szomszedos(Y,Z)) and szabad(Z).
        szabad(Z) if voltmar(Z) and ! and fail or asserta(voltmar(Z)).
        kerdesb(X,Y,Z) if retractall(voltmar(_)) and elokeszit(X,Y,Z).
/* c) kerdes: megad egy megyet, amelyiknek a legkevesebb szomszedja van*/
        szomszedszam(M,DB) if szomszedos(M,M1) and not(voltmar(M1)) and
                asserta(voltmar(M1)) and szomszedszam(M,DB1) and DB=DB1+1 and !
                or DB=0.
        foszomszedszam(M,DB) if retractall(voltmar(_)) and szomszedszam(M,DB).
        nemlegkevesebbszomszed(M) if szomszedos(M,_) and szomszedos(M1,_) and
                M<>M1 and foszomszedszam(M,DB) and foszomszedszam(M1,DB1) and
                DB>DB1.
        legkevesebb_szomszedja_van(M) if megye(M) and
                not(nemlegkevesebbszomszed(M)).
        ckerdes(M) if legkevesebb_szomszedja_van(M).

goal
        makewindow(1,14,15," Megye ",0,0,25,80) and
        clearwindow and nl and nl and
        write("Magyarorszag megyei: \n") and
        kiir and nl and nl and
        write("Nevezz meg egy megyet, amely nem szomszedos Heves megyevel!\n") and
        readchar(W) and
        kerdesa(X,heves) and write(X) and nl and
        write("Mondj egy megyet, amelyik nem szomszedja sem Somogy sem Pest
                megyenek!\n") and readchar(Q) and
        kerdesb(somogy,pest,Y) and write(Y) and nl and
        write("Adj meg egy megyet, amelyiknek a legkevesebb szomszedja van!\n") and
        readchar(T) and
        ckerdes(M) and write(M).
```

3. FEJEZET

A harmadik fejezetben olyan logikai fejtörőket mutatunk be, melyek megoldásához a Prolog nagy segítséget nyújt.

3.1. Verseny

A feladat

Egy könyvtárban műveltségi vetélkedőt tartottak, amelyen az olvasók vehettek részt.
A verseny után mindenkit megkérdeztek, melyik helyen végzett. A résztvevők a következő válaszokat adták:
Marika: Nem lettem sem első, sem utolsó.
Feri: Nem lettem első.
Zsiga: Első lettem.
Erzsi: Én lettem az utolsó.
A kérdés az, hogy mi volt a helyezési sorrendjük.

A feladat megoldása

Tudjuk, hogy a négy versenyző között négy helyezést oszthatnak ki:

elso(hely)
masodik(hely)
harmadik(hely)
negyedik(hely)

Elemezzük mondatonként az állításokat!
Marikáról tudjuk, hogy nem lett sem első, sem utolsó, tehát második vagy harmadik lett.

marika(M) if masodik(M) or harmadik(M).

Feri nem lett első, azaz vagy második vagy harmadik vagy negyedik lett.

feri(F) if masodik(F) or harmadik(F) or negyedik(F).

Zsiga nyerte meg a versenyt.

zsiga(Zs) if elso(Zs).

Erzsi pedig utolsó lett.

erzsi(E) if negyedik(E).

A teljes program

```
domains
        nev=string
        hely=string

predicates
        elso(hely)
        masodik(hely)
        harmadik(hely)
        negyedik(hely)
        marika(hely)
        feri(hely)
        zsiga(hely)
        erzsi(hely)
        megoldas(nev,hely,nev,hely,nev,hely,nev,hely)
clauses
        elso(elso).
        masodik(masodik).
        harmadik(harmadik).
        negyedik(negyedik).
/*Marika nem lett sem elso, sem utolso, tehat masodik vagy harmadik lett.*/
        marika(M) if masodik(M) or harmadik(M).
/*Feri nem lett elso, azaz vagy masodik vagy harmadik vagy negyedik lett*/
        feri(F) if masodik(F) or harmadik(F) or negyedik(F).
/*Zsiga nyerte meg a versenyt*/
        zsiga(Zs) if elso(Zs).
/*Erzsi pedig utolso lett*/
```

```
erzsi(E) if negyedik(E).
megoldas("Marika",M,"Feri",F,"Zsiga",Zs,"Erzsi",E) if
        marika(M) and
        feri(F) and zsiga(Zs) and erzsi(E) and M<>F and M<>Zs and
        M<>E and F<>Zs and F<>E and Zs<>E.

goal
        clearwindow and makewindow(1,11,15,"Verseny",1,0,23,80) and nl and
        write("Egy konyvtarban muveltsegi vetelkedot tartottak, \n") and
        write("amelyen olvasok vehettek reszt. \n") and
        write("A verseny utan mindenkit megkerdeztek, melyik helyen vegzett. \n") and
        write(" A kovetkezo valaszokat adtak: \n") and
        write(" Marika:   Nem lettem sem elso, sem utolso.\n ") and
        write("Feri:  Nem lettem elso.\n ") and
        write("Zsiga: Elso lettem.\n")  and
        write(" Erzsi: En lettem az utolso.\n ") and
        write("Mi volt a helyezesi sorrendjuk? \n\n") and readchar(Y) and
        megoldas("Marika",M,"Feri",F,"Zsiga",Zs,"Erzsi",E) and
        write("  Marika: ",M,"   Feri: ",F,"   Zsiga: ",Zs,"   Erzsi: ",E) and
        nl and fail.
```

3.2. Munkatárs

A feladat

Könyvtárunk munkatársai: a könyvtárvezető, a könyvtáros és a rendszergazda. Őket Kissnek, Nagynak és Kovácsnak hívják, azonban nem feltétlenül ebben a sorrendben. A könyvtár három olvasóját is véletlenül pont így hívják (nem csoda, hiszen ezek nagyon gyakori nevek). Őket Dr. Kissnek, Dr. Nagynak és Dr. Kovácsnak szokták szólítani. A következőket tudjuk a könyvtár munkatársairól és az olvasókról:

a) Dr. Kiss Debrecenben lakik.

b) Dr. Kovács Hajdúnánáson lakik.

c) A hajdúböszörményi olvasó névrokona a könyvtárvezető.

d) Kovács úr nem könyvtáros.

Kérdésem: mi a rendszergazda neve?

A feladat megoldása

Először meg kell adnunk, hogy kik a könyvtár munkatársai,

```
munkatars(kiss).
munkatars(nagy).
munkatars(kovacs).
```

és kik az olvasókhoz.

```
olvaso(dr_kiss).
olvaso(dr_nagy).
olvaso(dr_kovacs).
```

Az is rögzítenünk kell, hogy ki kinek a névrokona.

```
nevrokon(kiss,dr_kiss).
nevrokon(nagy,dr_nagy).
nevrokon(kovacs,dr_kovacs).
```

Ezután sorra kell vennünk a feladatban megadott állításokat. Az első szerint Dr. Kiss Debrecenben lakik.

lakik(dr_kiss,debrecen).

A második azt mondja ki, hogy Dr. Kovács Hajdúnánáson lakik.

lakik(dr_kovacs,hajdunanas).

Tudjuk még, hogy a harmadik utas sem Debrecenben, sem Hajdúnánáson nem lakhat, tehát Hajdúböszörményben lakik.

lakik(X,hajduboszormeny) if olvaso(X) and not(lakik(X,debrecen)) and not(lakik(X,hajdunanas)).

A harmadik állítás szerint a hajdúböszörményi olvasó névrokona a könyvtárvezető.

foglalkozas(X,konyvtarvezeto) if nevrokon(X,Y) and lakik(Y,hajduboszormeny).

A negyedik kijelentés szerint Kovács úr nem könyvtáros. Tehát vagy könyvtárvezető vagy rendszergazda.

foglalkozas(kovacs,rendszergazda) if not(foglalkozas(kovacs,konyvtarvezeto)).

Tudjuk még, hogy a könyvtár munkatársai közül aki nem könyvtárvezető és nem rendszergazda, annak feltétlenül könyvtárosnak kell lennie.

foglalkozas(X,konyvtaros) if munkatars(X) and not(foglalkozas(X,konyvtarvezeto))
and not(foglalkozas(X,rendszergazda)).

A teljes program

domains

nev,tevekenyseg,varos=symbol.

predicates

munkatars(nev).
olvaso(nev).
nevrokon(nev,nev).
lakik(nev,varos).
foglalkozas(nev,tevekenyseg).

```
clauses
/*a konyvtar munkatarsai:*/
        munkatars(kiss).
        munkatars(nagy).
        munkatars(kovacs).
/*az olvasok:*/
        olvaso(dr_kiss).
        olvaso(dr_nagy).
        olvaso(dr_kovacs).
/*ki kinek a nevrokona:*/
        nevrokon(kiss,dr_kiss).
        nevrokon(nagy,dr_nagy).
        nevrokon(kovacs,dr_kovacs).
/*a) Dr. Kiss Debrecenben lakik.*/
        lakik(dr_kiss,debrecen).
/*b) Kollegaja, Dr. Kovacs hajdunanason lakik.*/
        lakik(dr_kovacs,hajdunanas).
/*Tudjuk, hogy a harmadik utas sem sem Debrecenben, sem Hajdunanason nem lakhat,
tehát Hajduboszormenyben lakik.*/
        lakik(X,hajduboszormeny) if olvaso(X) and not(lakik(X,debrecen)) and
not(lakik(X,hajdunanas)).
/*c) A hajdúböszörményi olvasó névrokona a könyvtárvezető.*/
        foglalkozas(X,konyvtarvezeto) if nevrokon(X,Y) and lakik(Y,hajduboszormeny).
/*d) Kovacs ur nem konyvtaros.*/
        foglalkozas(kovacs,rendszergazda) if not(foglalkozas(kovacs,konyvtarvezeto)).
/* Tudjuk meg, hogy a konyvtar munkatarsai kozul aki nem konyvtarvezeto es nem
rendszergazda, annak feltetelenul konyvtarosnak kell lennie.*/
        foglalkozas(X,konyvtaros) if munkatars(X) and not(foglalkozas(X,konyvtarvezeto))
                and not(foglalkozas(X,rendszergazda)).

goal
        clearwindow and makewindow(1,11,15,"Munkatars",1,0,23,80) and
        write("Konyvtarunk munkatarsai: a konyvtarvezeto, a konyvtaros ") and
```

write("es a rendszergazda. \n") and
write("Oket Kissnek, Nagynak es Kovacsnak hivjak,\n") and
write("azonban nem feltetlenul ebben a sorrendben.\n") and
write("A konyvtar harom olvasojat is veletlenul pont igy hivjak\n") and
write("(nem csoda, hiszen ezek nagyon gyakori nevek).\n") and
write("Oket Dr. Kissnek, Dr. Nagynak es Dr. Kovacsnak\n") and
write("szoktak szolitani. \n") and
write("A kovetkezoket tudjuk a munkatarsakrol es az olvasokrol:\n") and
write("a) Dr. Kiss Debrecenben lakik.\n") and
write("b) Dr. Kovacs Hajdunanason lakik.\n") and
write("c) A hajduboszormenyi utas nevrokona a konyvtarvezeto.\n") and
write("d) Kovacs ur nem konyvtaros.\n") and
write("Kerdesem: mi a rendszergazda neve?\n") and
readchar(W) and foglalkozas(X,rendszergazda) and nl and write(X).

3.3. Idősebb

A feladat

Könyvtárunk két olvasóját, egy fiút és egy lányt megkérdeztek arról, hogy melyikük az idősebb.
- Én vagyok az idősebb - mondta a fiú.
- Én vagyok a fiatalabb - mondta a lány.
Kiderült azonban, hogy legalább az egyikük hazudott. Ki az idősebb?

A feladat megoldása

Írjuk le Prolog-nyelven az egyes kijelentéseket!
Az illető korát, amely lehet idősebb vagy fiatalabb, a következőképp jelöljük:

kora(idosebb).

kora(fiatalabb).

Az egyik alapfeltétel, hogy a fiú és a lány különböző korú.

allit(F,L) if kora(F) and kora(L) and not(F=L).

Ha a lány igazat mond, akkor a fiú az idősebb és a lány a fiatalabb.

allitLany(idosebb,fiatalabb).

Ha a lány hazudik, akkor a fiú a fiatalabb és a lány az idősebb.

allitLany(fiatalabb,idosebb).

Ha a fiú igazat mond, akkor ő az idősebb és a lány a fiatalabb.

allitFiu(idosebb,fiatalabb).

Ha a fiú hazudik, akkor ő a fiatalabb és a lány az idősebb.

allitFiu(fiatalabb,idosebb).

Tudjuk azonban, hogy legalább egyikük hazudott. Vagyis vagy a fiú, vagy a lány, vagy mindkettő nem mondott igazat. Valamennyi esetet sorra kell vennünk.

```
legalabb1h(I,F) if allitLany(I,fiatalabb) and not(allitFiu(idosebb,F))
        and kora(F) and kora(I) and not(F=I).
legalabb1h(I,F) if kora(I) and kora(F) and allitFiu(idosebb,F)
        and not(allitLany(I,fiatalabb)).
legalabb1h(I,F) if kora(I) and kora(F) and allit(I,F)
        and not(allitFiu(idosebb,F)) and not(allitLany(I,fiatalabb)).
```

A teljes program

```
domains
        milyen=string

predicates
        kora(milyen).
        allit(milyen,milyen).
        allitLany(milyen,milyen).
        allitFiu(milyen,milyen).
        kerdes(milyen,milyen).
        legalabb1h(milyen,milyen).

clauses
        kora(idosebb).
        kora(fiatalabb).
        allit(F,L) if kora(F) and kora(L) and not(F=L).
/*ha a lany igazat mond*/
        allitLany(idosebb,fiatalabb).
/*ha a lany hazudik*/
        allitLany(fiatalabb,idosebb).
/*ha a fiu igazat mond*/
        allitFiu(idosebb,fiatalabb).
/*ha a fiu hazudik*/
        allitFiu(fiatalabb,idosebb).
```

```
/*legalabb az egyikuk hazudott*/
        legalabb1h(I,F) if allitLany(I,fiatalabb) and not(allitFiu(idosebb,F))
                and kora(F) and kora(I)
                        and not(F=I).
        legalabb1h(I,F) if kora(I) and kora(F) and allitFiu(idosebb,F)
                        and not(allitLany(I,fiatalabb)).
        legalabb1h(I,F) if kora(I) and kora(F) and allit(I,F) and not(allitFiu(idosebb,F))
                         and not(allitLany(I,fiatalabb)).
/*a megoldas*/
        kerdes(I,F) if allitFiu(I,F) and allitLany(I,F) and legalabb1h(I,F) .

goal
        clearwindow and makewindow(1,15,15,"Ki az idosebb?",1,0,23,80) and
        write("Konyvtarunk ket olvasojat, egy fiut es egy lanyt megkerdeztek \n") and
        write("arrol, hogy melyikuk az idosebb. \n") and
        write("- En vagyok az idosebb -mondta a fiu. \n") and
        write("- En vagyok a fiatalabb -mondta a lany. \n") and
        write("Kiderult azonban, hogy legalabb az egyikuk hazudott.\n") and
        write("Ki az idosebb?\n") and readchar(W) and
        kerdes(F,L) and write("A fiu ",F,", ","a lany ",L,".") and nl .
```

3.4. Foglalkozás

A feladat

Könyvtárunk négy olvasójának vezetékneve: Kanász, Halász, Vadász és Madarász. Az egyikük foglalkozása kanász, a másiké halász, a harmadiké vadász, a negyedike madarász. Tudjuk, hogy Kanász nem halász, Halász nem vadász, Vadász nem madarász, Madarász nem kanász és nem halász, valamint egyikük foglalkozása sem egyezik meg a vezetéknevével. Melyiküknek mi a foglalkozása?

A feladat megoldása

Tudjuk, hogy egyik személy foglalkozása sem egyezik meg a vezetéknevével. Ezért, ha azt mondjuk Kanászról, hogy nem halász, akkor csak vadász vagy madarász lehet, kanász semmiképp sem.

muvel("Kanasz",vadasz).
muvel("Kanasz",madarasz).

Halász nem vadász, így vagy kanász vagy madarász lehet.

muvel("Halasz",kanasz).
muvel("Halasz",madarasz).

Vadász nem madarász, azaz vagy halász vagy kanász a foglalkozása.

muvel("Vadasz",halasz).
muvel("Vadasz",kanasz).

Madarász nem kanász és nem halász, vagyis ő a vadász.

muvel("Madarasz",vadasz).

A teljes program

```
domains
        fajta=string
        nev=string

predicates
        muvel(nev,fajta)
        megold(nev,fajta,nev,fajta,nev,fajta,nev,fajta)

clauses
/*Kanasz nem halasz, tehat vagy vadasz vagy madarasz*/
/*kanasz nem lehet, mivel egyikuk foglalkozasa sem
egyezik meg a vezeteknevevel*/
        muvel("Kanasz",vadasz).
        muvel("Kanasz",madarasz).
/*Halasz nem vadasz, vagyis vagy kanasz vagy madarasz*/
        muvel("Halasz",kanasz).
        muvel("Halasz",madarasz).
/*Vadasz nem madarasz, azaz vagy halasz vagy kanasz*/
        muvel("Vadasz",halasz).
        muvel("Vadasz",kanasz).
/*Madarasz nem kanasz es nem halasz, vagyis o a vadasz*/
        muvel("Madarasz",vadasz).
        megold("Kanasz",A,"Halasz",B,"Vadasz",C,"Madarasz",D) if muvel("Kanasz",A)
                and muvel("Halasz",B) and muvel("Vadasz",C) and
                muvel("Madarasz",D) and
                A<>B and A<>C and A<>D and
                B<>C and B<>D and C<>D.

goal
        clearwindow and makewindow(1,12,75,"Kinek mi a foglalkozasa?",1,0,23,80) and
         nl and
        write(" Konyvtarunk negy olvasojanak vezetekneve: Kanasz, Halasz, Vadasz es
```

Madarasz. \n") and

write(" Az egyikuk foglalkozasa kanasz, a masike halasz, a harmadike vadasz,\n")

and write(" a negyedike madarasz. \n") and

write(" Tudjuk, hogy Kanasz nem halasz, Halasz nem vadasz, Vadasz nem madarasz, \n") and

write(" Madarasz nem kanasz es nem halasz, valamint egyikuk foglalkozasa \n")

and write(" sem egyezik meg a vezeteknevevel.\n\n") and

write(" Melyikuknek mi a foglalkozasa? \n\n") and readchar(Y) and

megold("Kanasz",A,"Halasz",B,"Vadasz",C,"Madarasz",D) and

write(" Kanasz: ",A,"\n Halasz: ",B,"\n Vadasz: ",C,"\n Madarasz: ",D) and

nl and fail or readchar(Y).

3.5. Olvasó

A feladat

Könyvtárunk egyik olvasója meglehetősen furcsán viselkedik: hétfőn, kedden és szerdán hazudik, a többi napon igazat mond. Az olvasó mond két állítást, ezek alapján a könyvtárosnak meg kell mondania, hogy milyen nap van ma. Az állítások a következők:

– Tegnap hazudtam.
– Holnap megint hazudni fogok.

Milyen nap van ma?

A feladat megoldása

Mivel az olvasó nem minden nap mond igazat, ezért először is meg kell határoznunk az igazmondó napjait.

igazmondo(csutortok).
igazmondo(pentek).
igazmondo(szombat).
igazmondo(vasarnap).

Fontos definiálnunk azt is, hogy mit értünk tegnap alatt. Például a hétfő "tegnapja" vasárnap stb.

tegnap(vasarnap,hetfo).
tegnap(hetfo,kedd).
tegnap(kedd,szerda).
tegnap(szerda,csutortok).
tegnap(csutortok,pentek).
tegnap(pentek,szombat).
tegnap(szombat,vasarnap).

A holnapot ehhez hasonlóan határozzuk meg:

holnap(hetfo,vasarnap).
holnap(kedd,hetfo).
holnap(szerda,kedd).
holnap(csutortok,szerda).
holnap(pentek,csutortok).
holnap(szombat,pentek).
holnap(vasarnap,szombat).

Ezek után már csak az állítások megfogalmazására van szükség. Nem mindegy, hogy az olvasó igazmondó, vagy hazudós napján nyilatkozott, ezért szét kell választanunk az egyes eseteket.

Ha az olvasó ma igazat mond, akkor ez azt jelenti, hogy tegnap hazudott.

allit1(X) if igazmondo(X) and tegnap(Y,X) and not(igazmondo(Y)).

Ha tegnap mondott igazat, akkor ma hazudik.

allit1(X) if tegnap(Y,X) and igazmondo(Y) and
not(igazmondo(X)).

Ha ma igazat mond, akkor holnap hazudni fog.

allit2(X) if igazmondo(X) and holnap(Y,X) and not(igazmondo(Y)).

Ha viszont holnap mond igazat, ez azt jelenti, hogy ma hazudik.

allit2(X) if holnap(Y,X) and igazmondo(Y) and not(igazmondo(X)).

Akkor találtuk meg a megoldást, ha valamennyi állítás teljesül.

megoldas(X) if allit1(X) and allit2(X).

A végeredmény szerint nincs ilyen nap, azaz a feladatnak nincs megoldása. Szerencsére erre az esetre is felkészült a program.

A teljes program

```
domains
        nap=string

predicates
        igazmondo(nap)
        tegnap(nap,nap)
        holnap(nap,nap)
        allit1(nap)
        allit2(nap)
        megoldas(nap)

clauses
/*az olvaso igazmondo napjai*/
        igazmondo(csutortok).
        igazmondo(pentek).
        igazmondo(szombat).
        igazmondo(vasarnap).
/*a tegnap es a holnap definialasa*/
        tegnap(vasarnap,hetfo).
        tegnap(hetfo,kedd).
        tegnap(kedd,szerda).
        tegnap(szerda,csutortok).
        tegnap(csutortok,pentek).
        tegnap(pentek,szombat).
        tegnap(szombat,vasarnap).
        holnap(hetfo,vasarnap).
        holnap(kedd,hetfo).
        holnap(szerda,kedd).
        holnap(csutortok,szerda).
        holnap(pentek,csutortok).
        holnap(szombat,pentek).
        holnap(vasarnap,szombat).
```

```
/*ha ma igazat mond az olvaso, akkor tegnap hazudott*/
        allit1(X) if igazmondo(X) and
                tegnap(Y,X) and not(igazmondo(Y)).
/*ha tegnap igazat mondott, ma hazudik*/
        allit1(X) if tegnap(Y,X) and igazmondo(Y) and
                not(igazmondo(X)).
/*ha ma igazat mond, holnap hazudni fog*/
        allit2(X) if igazmondo(X) and
                holnap(Y,X) and not(igazmondo(Y)).
/*ha holnap mond igazat, ma hazudik*/
        allit2(X) if holnap(Y,X) and igazmondo(Y) and
                not(igazmondo(X)).
/*akkor talaltuk meg a megoldast, ha valamennyi allitas teljesul*/
        megoldas(X) if allit1(X) and allit2(X).

goal
        makewindow(1,2,75,"Milyen nap van ma? ",0,0,25,80) and window_attr(112) and
        clearwindow and nl and nl and
        write("  Konyvtarunk egyik olvasoja meglehetosen furcsan viselkedik:")and nl and
        write("  minden hetfon, kedden es szerdan hazudik,") and nl and
        write("  a tobbi napon igazat mond.") and nl and
        write("  Az olvaso mond ket allitast, ezek alapjan a konyvtarosnak meg kell") and nl
        and
        write("  mondania, hogy milyen nap van ma.") and nl and
        write("  Az allitasok a kovetkezok") and nl and
        write("        Tegnap hazudtam.") and nl and
        write("        Holnap megint hazudni fogok. ") and nl and nl and
        write("  Milyen nap van ma?") and readchar(Y) and nl and nl
        and megoldas(X) and nl and write("          ") and write(X) and fail or
        write("    Nincs ilyen nap").
```

3.6. Tolvaj

A feladat

Könyvtárunkban az egyik nap lopás történt. Szerencsére valamennyi hiányzó könyv megkerült és kiderült, hogy vagy Arnold vagy Ivett vagy Hugó volt a tolvaj.
Megkérdezésükkor a következőket állították:
Arnold: Nem én követtem el a rablást.
Ivett: Nem Hugó volt.
Hugó: De, én voltam.
Később közülük ketten bevallották, hogy hazudtak. Ki volt a tettes?

A feladat megoldása

Az olvasók vagy bűnösek vagy ártatlanok lehetnek.

```
milyen(bunos). /*bunos lehet vagy artatlan*/
milyen(artatlan).
```

Igazat mondanak vagy pedig hazudnak.

```
mitmond(igazatmond). /*igazat mond vagy hazudik*/
mitmond(hazudik).
```

A továbbiakban állításaink úgy épülnek fel, hogy mindhárom személy ártatlanságáról, illetve igazmondásáról állítunk valamit.

Tudjuk, hogy csak az egyikük bűnös. Ezt a következőképp fogalmazzuk meg:

```
allit1(artatlan,P,artatlan,Q,bunos,R) if mitmond(P)
                        and mitmond(Q) and mitmond(R).
allit1(artatlan,P,bunos,Q,artatlan,R) if mitmond(P)
                        and mitmond(Q) and mitmond(R).
allit1(bunos,P,artatlan,Q,artatlan,R) if mitmond(P)
                        and mitmond(Q) and mitmond(R).
```

Azt is tudjuk, hogy a három személy közül ketten hazudnak.

allit2(A,igazatmond,I,hazudik,H,hazudik) if milyen(A)
and milyen(I) and milyen(H).
allit2(A,hazudik,I,igazatmond,H,hazudik) if milyen(A)
and milyen(I) and milyen(H).
allit2(A,hazudik,I,hazudik,H,igazatmond) if milyen(A)
and milyen(I) and milyen(H).

Mindhárman mondhatnak igazat vagy hazudhatnak is. Vegyük sorra az állításokat!
Ha Arnold igazat mond, akkor ő ártatan.

allitA(artatlan,igazatmond,I,Q,H,R) if milyen(I) and milyen(H)
and mitmond(Q) and mitmond(R).

Ha viszont Arnold hazudik, akkor az alábbiakat írhatjuk fel:

allitA(bunos,hazudik,I,Q,H,R) if milyen(I) and milyen(H)
and mitmond(Q) and mitmond(R).

Ha Ivett igazat mond, akkor vallomása szerint Hugó ártatlan.

allitI(A,P,I,igazatmond,artatlan,R) if milyen(A) and milyen(I)
and mitmond(P) and mitmond(R).

Ha Ivett hazudik, akkor ez azt jelenti, hogy Hugó bűnös.

allitI(A,P,I,hazudik,bunos,R) if milyen(A) and milyen(I)
and mitmond(P) and mitmond(R).

Amennyiben Hugó igazat mond, akkor ő biztosan bűnös.

allitH(A,P,I,Q,bunos,igazatmond) if milyen(A) and milyen(I)
and mitmond(P) and mitmond(Q).

Ha viszont hazudik, akkor ártatlan.

allitH(A,P,I,Q,artatlan,hazudik) if milyen(A) and milyen(I)
and mitmond(P) and mitmond(Q).

A megoldást akkor kapjuk meg, ha valamennyi állításunk teljesül.

```
megoldas(A,P,I,Q,H,R) if allit1(A,P,I,Q,H,R) and allit2(A,P,I,Q,H,R)
                and allitA(A,P,I,Q,H,R) and allitI(A,P,I,Q,H,R) and
                allitH(A,P,I,Q,H,R).
```

A teljes program

```
domains
        ki=string

predicates
        milyen(ki)  /*milyen bunosseg szempontjabol*/
        mitmond(ki)  /*igazat mond vagy hazudik*/
        allit1(ki,ki,ki,ki,ki,ki)  /*egy ember kovette el*/
        allit2(ki,ki,ki,ki,ki,ki)  /*ketten hazudnak*/
        allitA(ki,ki,ki,ki,ki,ki)  /* Arnold allitasa */
        allitI(ki,ki,ki,ki,ki,ki)  /* Ivett allitasa */
        allitH(ki,ki,ki,ki,ki,ki)  /* Hugo allitasa*/
        megoldas(ki,ki,ki,ki,ki,ki)  /* a megoldas */

clauses
        milyen(bunos). /*bunos lehet vagy artatlan*/
        milyen(artatlan).
        mitmond(igazatmond). /*igazat mond vagy hazudik*/
        mitmond(hazudik).
/*Csak egyikuk bunos*/
        allit1(artatlan,P,artatlan,Q,bunos,R) if mitmond(P)
                                and mitmond(Q) and mitmond(R).
        allit1(artatlan,P,bunos,Q,artatlan,R) if mitmond(P)
                                and mitmond(Q) and mitmond(R).
        allit1(bunos,P,artatlan,Q,artatlan,R) if mitmond(P)
                                and mitmond(Q) and mitmond(R).
/*Ketten hazudnak*/
        allit2(A,igazatmond,I,hazudik,H,hazudik) if milyen(A)
                                and milyen(I) and milyen(H).
```

```
        allit2(A,hazudik,I,igazatmond,H,hazudik) if milyen(A)
                                and milyen(I) and milyen(H).
        allit2(A,hazudik,I,hazudik,H,igazatmond) if milyen(A)
                                and milyen(I) and milyen(H).
/* Arnold allitasa */
        allitA(artatlan,igazatmond,I,Q,H,R) if milyen(I) and milyen(H)
                        and mitmond(Q) and mitmond(R).
        allitA(bunos,hazudik,I,Q,H,R) if milyen(I) and milyen(H)
                        and mitmond(Q) and mitmond(R).
/*Ivett allitasa*/
        allitI(A,P,I,igazatmond,artatlan,R) if milyen(A) and milyen(I)
                        and mitmond(P) and mitmond(R).
        allitI(A,P,I,hazudik,bunos,R) if milyen(A) and milyen(I)
                        and mitmond(P) and mitmond(R).
/*Hugo allitasa*/
        allitH(A,P,I,Q,bunos,igazatmond) if milyen(A) and milyen(I)
                        and mitmond(P) and mitmond(Q).
        allitH(A,P,I,Q,artatlan,hazudik) if milyen(A) and milyen(I)
                        and mitmond(P) and mitmond(Q).
/* a megoldas: ha allit1, allit2, allitA, allitI es allitH is teljesul */
        megoldas(A,P,I,Q,H,R) if allit1(A,P,I,Q,H,R) and allit2(A,P,I,Q,H,R)
                        and allitA(A,P,I,Q,H,R) and allitI(A,P,I,Q,H,R) and
                        allitH(A,P,I,Q,H,R).

goal
        makewindow(1,7,75,"Tolvaj",0,0,24,80) and clearwindow and nl and
        write("Konyvtarunkban az egyik nap lopas tortent. \n") and
        write("Szerencsere valamennyi hianyzo konyv megkerult es \n") and
        write("kiderult, hogy vagy Arnold vagy Ivett vagy pedig Hugo volt a tolvaj. \n") and
        write("Megkerdezesukkor a kovetkezoket allitottak: \n") and
        write("Arnold: Nem en kovettem el a rablast. \n") and
        write("Ivett: Nem Hugo volt. \n") and
        write("Hugo: De, en voltam. \n") and
        write("Kesobb kozuluk ketten bevallottak, hogy hazudtak. \n") and
```

```
write("Ki volt a tettes? \n\n") and
write("\n\t\t A megoldas: \n\n") and readchar(Y) and
megoldas(A,P,I,Q,H,R) and
write("Arnold ",A," es ",P,".\n") and
write("Ivett ",I," es ",Q,".\n") and
write("Hugo ",H," es ",R,".\n") and nl and fail or nl.
```

IRODALOM

Cawsey, Alison (2002): Mesterséges intelligencia. Alapismeretek. Panem, Budapest

Clocksin, William F. (2003): Programming in Prolog. Springer, Berlin

Csink László (1991): Logikai programozás. KKVMF, Budapest

Flach, Peter (2001): Logikai programozás. Panem, Budapest

Grothaus, Manfred – Gust, Helmar (1987): Turbo Prolog. Einführung · Anwendungen · Vergleich mit anderen Systemen. Vogel-Buchverlag, Würzburg

Justen, Konrad (1988): Turbo Prolog – Einführungen in die Anwendung. Friedr. Vieweg & Sohn, Braunschweig, Wiesbaden

Makány György (1995): Programozási nyelvek: Prologika. ELTE TTK Általános Számítástudományi Tanszék, Budapest

Márkusz Zsuzsanna (1988): Prologban programozni könnyű. Novotrade, Budapest

Prolog: programozás logikában (1982). Számalk, Budapest

Schildt, Herbert (1987): Professionelles Turbo Prolog. McGraw-Hill Book GmbH, Hamburg

Smith, Peter (1988): Expert system development in Prolog and Turbo-Prolog. Sigma Press, Wilmslow

Sterling, Leon – Shapiro, Ehud (1997): The Art of Prolog. The MIT Press Cambridge, Massachusetts, London

Weiskamp, Keith – Hengl, Terry (1989): KI-Programmierung mit Turbo Prolog. McGraw-Hill Book GmbH, Hamburg