SZAKDOLGOZAT

A LOGIKAI PROGRAMOZÁS ÉS ALKALMAZÁSAI A PROLOG PROGRAMOZÁSI NYELVBEN

Készítette: **Túri József Attila**

programtervező informatikus (MSc)
szakos hallgató

Témavezető: **Dr. Herendi Tamás**

egyetemi adjunktus

Debreceni Egyetem, 2009

Köszönetet mondok dr. Herendi Tamás egyetemi adjunktusnak a munkámhoz nyújtott sokoldalú segítségéért és hasznos tanácsaiért, amelyekkel elősegítette szakdolgozatom megírását.

Tartalomjegyzék

BEVEZETÉS

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.

A hagyományos (imperatív) programozási nyelvekben, a program procedurális leírása, specifikációja annak, hogy a problémát hogyan kell megoldani.

Ezzel szemben a logikai program deklaratív specifikációja annak, hogy mi a megoldandó probléma. A logikai programozás egy egészen másfajta, a megszokottól eltérő gondolkodásmódot követel meg, mint amit megszokhattunk az imperatív paradigmában.

Ez igaz például a programbeli változó fogalmára. 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. Valójában a változó elnevezés ebben az esetben helytelen is, mivel egy minden pillanatban jól meghatározott értékre utal. Ezzel szemben a logikai program változója egy matematikai értelemben vett változó, ami bármilyen értéket felvehet. Ebből a szempontból a logikai programozás sokkal közelebb áll a matematikai érzésvilághoz, mint az imperatív programozás.

Az imperatív és a logikai programozás a feltételezett végrehajtó gépmodell tekintetében is eltér. A gépmodell a programok futtatására alkalmazott számítógép absztrakciója. Az imperatív paradigma egy dinamikus állapot alapú gépmodellt feltételez, ahol a gép állapotát a memóriatartalom határozza meg. A programutasítás hatása átmenet az egyik állapotból egy másikba. A logikai programozás nem feltételez ilyen dinamikus gépmodellt. A számítógép és a program együtt meghatározott mennyiségű tudást testesít meg a világról, amit kérdések megválaszolására használunk.

A logikai programozás sajátosságait a Prolog programozási nyelv példáján keresztül mutatjuk be.

A Prolog matematikai logikán alapuló magas szintű programozási nyelv. Célja, hogy a számítógép alkalmazkodjon az ember igényeihez és ne megfordítva. A Prolog beépített

következtetési rendszere olyan intelligens és korszerű programozási elveket enged meg, amelyek merőben eltérnek a hagyományos gyakorlattól.

A Prolog sokéves kutatómunka eredménye. Az első hivatalos Prolog verziót a marseilles-i egyetemen fejlesztették ki Alain Colmerauer vezetésével az 1970-es évek elején, ők adták a nevét is (PROgramming in LOGic). Ekkorra már nagyon felerősödött az az igény, hogy minél több programfejlesztési munkát bízzanak a gépre, kiváltva a szellemi munka kevésbé hatékony területeit. Olyan nyelv gondolatai körvonalazódtak, amely szabályok és tények alapján dolgozik, látszólag önállóan oldja meg a problémát. 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. A 80-as években a nyelv alkalmazási területe rohamosan bővült: építészeti és gépészeti tervező rendszereket, szimulációs programokat és rendkívül sokfajta ún. szakértői rendszert írnak Prologban Magyarországon és a világon. Ma a Prolog az egyik legfontosabb eszköze a mesterséges intelligenciát alkalmazó programozásnak és a szakértői rendszereknek. A Prolog felhasználóbarát és intelligens programozási nyelv.

Miért is érdekes ez a nyelv? Mert sokkal közelebb áll az emberi gondolkodáshoz, logikához, ezért a természetes nyelvhez is, mint a többi ún. algoritmikus hagyományos programozási nyelv. A Prolog a probléma leírását adja meg a számítógépnek, kellő számú tény és szabály felsorolásával, majd kéri a rendszert, hogy keresse meg a probléma összes lehetséges megoldását. Így nem kell egy probléma megoldásához a megoldó algoritmust részletesen leírni. Ha tehát egyszer a Prolog programozó leírta, hogy mit kell kiszámítani, a Prolog rendszer maga szervezi meg, hogy hogyan kell a számítást végrehajtani, azaz intelligens. Az emberi logika szabályaiból áll, amelyeket már a régi görögök is ismertek, nem pedig a számítógépre szabott speciális utasítások sorozatából.

A szakdolgozat tartalmazza a logikai programozás legfőbbb sajátosságait, Prolog programozás alapelveit, és bemutatja a Prologgal megoldható feladatok megoldásait.

A szakdolgozat három részből áll. Az első a nyelv elméleti alapjait tárgyalja. Minden tudományos bevezetés (jelölések, definíciók, tételek) helyett az elméleti alapokat példaprogramokon keresztül magyarázza el. A dolgozat második és harmadik fejezete feladatokat és azok megoldásait foglalja magában. Valamennyi feladathoz nyújt segítséget, a megoldáshoz szükséges legfontosabb ötleteket mutatja be, majd magát a teljes programot is közli. A programok egyszerűek, egyik sem hosszabb három oldalnál és úgy vannak

összeválogatva, hogy felöleljék a Prolog programozás legfontosabb és legjellemzőbb tulajdonságait, illetve eszközeit. Mindegyik tartalmaz egy-két új programozási módszert vagy trükköt.

A második fejezet a Prolog programozás kezdő lépéseinek elsajátításához nyújt segítséget, a legegyszerűbb programoktól a nehezebbek felé haladva.

A harmadik fejezet kifejezetten logikai fejtörőkkel foglalkozik. Ezek megoldása már több ismeretet igényel, de a feladatok sokszínűsége és érdekessége mindenért kárpótolja a programozót.

1. FEJEZET

A logikai programozás és a Prolog program alapelemei

1.1. A logikai programozás hazai története

A hazai logikai programozás a nemzetközi fejlődés menetét követte, időnként a konkrét programozási eszközök megvalósításában azt meg is előzve.
A kezdetek a hatvanas évek végére, a hetvenes évek elejére nyúlnak vissza, amikor virágkorukat élték a logikai alapú tételbizonyítók. A tételbizonyítók elve a következő: tudásunkat a világról logikai formulák segítségével leírjuk, ezeket a formulákat tekintjük világunk axiómáinak, majd megfelelő következtetési szabályokkal levezetjük azokat az állításokat, melyek ezekből az axiómákból következnek. Ezeket az állításokat a továbbiakban rendszerünk tételeinek tekintjük. Amennyiben levezetési szabályaink megfelelőek és használati sorrendjük is megfelelő, vagyis jó a stratégiánk, akkor elvileg minden olyan állítást (tételt) le tudunk vezetni segítségükkel, mely az axiómáinkból következik. Ebben az esetben teljes logikai rendszerről beszélhetünk.
A fentiek egy klasszikus példa alapján szemléltethetők a legegyszerűbben: abból az axiómából, hogy minden ember halandó és Arisztotelész ember, következik: Arisztotelész is halandó (a modus ponens szabály alkalmazása).
Axiómáink megfogalmazásánál itt az elsőrendű predikátum kalkulust használtuk, amely a későbbiek során kitüntetett szerepet játszott a logikai programozásban.

A logikai programozás történetében a következő feladat az volt, hogyan lehet olyan logikai rendszert konstruálni, amely alkalmas a számítógépes feldolgozásra. Ebben áttörő eredményt jelentett J. A. Robinson rezolúciónak nevezett módszere. Az elv lényege: annak bizonyítása helyett, hogy egy állítás logikai következménye (tétele) axiómáinknak, bizonyítsuk be, hogy negáltja (vagyis tagadása) nem logikai következmény (nem tétele), vagyis a tétel negáltjával bővített axiómahalmaz ellentmondásos. Vagyis a feladat arra módosul, hogy konstruáljunk olyan algoritmust, amely egy formula halmazról bebizonyítja, hogy ellentmondásos. Ilyen algoritmus a rezolúció. A rezolúció olyan matematikai módszer, amelynek első lépéseként egy elsőrendű formulahalmazt átalakítunk

egy vele ekvivalens speciális alakú formulahalmazzá az ún. klózok halmazává (konjunkciójává). Ennek megfelelően axiómáinkat klóz alakra hozzuk, majd bizonyítandó tétel szintén klóz alakú negáltjával kiegészítjük axióma halmazunkat. A rezolúció pedig, mint következtetési szabály (valójában a modus ponens átfogalmazása) pedig mindössze annyit mond, hogy amennyiben egy klóz tartalmaz egy atomi állítást, egy másik pedig ennek negáltját, akkor változóik megfelelő behelyettesítése után ezek a predikátumok elhagyhatók a klózokból, a megmaradt részek pedig egyetlen klózzá olvaszthatók. Amennyiben ezt az eljárást folytatva eljutunk az ún. üres klózig, ami a „hamis" állításnak felel meg, úgy formulahalmazunk ellentmondásos, az eredeti tétel klóz alakjával bővített halmaz viszont ellentmondásmentes, így a bizonyítandó tétel logikai következménye axiómáinknak. A rezolúció egy további előnye, hogy konstruktív bizonyítási eljárás, tehát nem csak azt bizonyítja be, hogy egy feladatnak van megoldása (tétel), hanem meg is konstruálja ezt a megoldást. A konstruktív bizonyítási folyamat jelenti a logikai program futását: a tételbizonyító által felépített „ellenpélda" nem más, mint a logikai program kimenete.

Magyarországon, a hetvenes évek elején - akár csak máshol a világon, ahol erős logikai háttér állt rendelkezésre - igen kutatott témának számított a tételbizonyítás és a programhelyesség-bizonyítás. Az elmélettel elsősorban az MTA Matematikai Kutatóintézetében és a KFKIban foglalkoztak, míg a Nehézipari Minisztérium Ipargazdasági és Üzemszervezési Intézete (NIM IGÜSZI) az elmélet mellett ilyen rendszerek implementációjával is foglalkozott. Az első működő hazai rezolúciós tételbizonyító 1973-ban készült, az abban az időben a NIM IGÜSZI-ben alapszoftver-készítésre használt CDL-Compiler Description Language nyelvben.
Mint már a bevezetőben említettük, 1972-ben, Marseille-ben, Alain Colmerauer és Robert Kowalski munkásságára támaszkodva elkészítettek egy interpretert, amely speciális klózfajtára, az ún. Horn-klózokra épülő, rezolúciós tételbizonyítót tartalmazó programozási nyelv megvalósítása volt és amelyet PROLOG-nak (PROgramming in LOGic) neveztek el. A korábbi tételbizonyítók nagy hibája volt, hogy a feladatmegoldás során nagyon hamar fellépett az ún. kombinatorikus robbanás, vagyis a keresési tér rendkívül megnőtt és gyakorlatilag kivárhatatlanná vált a megoldás megtalálása. A Horn-formulák és ennek megfelelően a Prolog egyik előnye, hogy bár megszorítják az alkalmazható formulákat, az ún. LUSH rezolúcióval (a rezolúcióban részt vevő klózok egyike mindig az újonnan előálló klóz) hatékonyabb feladatmegoldást tesznek lehetővé. Egy másik tulajdonsága a

Horn-formuláknak, hogy a logikában szokásos deklaratív szemantika mellett, létezik procedurális szemantikájuk is, tehát egy általános Hornformula interpretálható eljárásdefinícióként is. Ennek megfelelően a Prologot tekinthetjük egy nagyon magas szintű programozási nyelvnek, ahol az eljáráshívás mintaillesztéssel történik az egyesítési algoritmuson keresztül.
1974-ben, egy angliai tanulmányút egyik eredményeképp, a NIM IGÜSZI-be behozták a Prolog leírását. 1975 első felében készült el CDL-ben egy Prolog-interpreter, amely gyakorlatilag megfelelt a Marseille-i Egyetemen készült Prolog-interpreternek. Az interpreter 20 logikai következtetést végzett másodpercenként az ICL 1903A-n, amely akkor Magyarország egyik legnagyobb teljesítményû számítógépe volt. Még ebben az évben, júniusban, a NIM IGÜSZI szervezésében egyhetes Prolog tanfolyam volt, melynek során elkészült az első hazai Prolog alkalmazói program. A következő hónapokban, a NIM IGÜSZI kedvező adottságait kihasználva, még további alkalmazások születtek: egy gyógyszerkölcsönhatást kimutató program, melyet ma szakértői rendszernek neveznénk, majd egy gyógyszerhatás előre becslő program és egy lakástervező program. Ezeknek az alkalmazásoknak jelentős nemzetközi visszhangjuk volt, mivel abban az időben a Prolog-programozást elsősorban egyetemi környezetben művelték, és gyakorlati alkalmazások nem voltak. 1976-ban Budapestre látogatott Robert Kowalski és David Warrenis, az azóta Prolog körökben alapfogalommá vált Warren-gép (absztrakt gép) megalkotója.

Az ezt követő húsz évben a hazai Prolog-fejlesztések két nagy ágra bomlottak: az MProlog moduláris Prolog rendszerre és alkalmazásaira, valamint a későbbi CS-PROLOG (TProlog) sokprocesszoros Prolog rendszerre és alkalmazásaira.
Ennek legfontosabb előzménye az volt, hogy 1979-ben a Prolog -fejlesztők több lépésben a NIM IGÜSZI-ből átkerültek a Számítástechnikai Koordinációs Intézet (SZKI) két részlegébe, az Elméleti Laborba (ELL) és a Szoftver Alkalmazásfejlesztési Laborba (SOL). Az Elméleti Laborban elkezdődött a professzionális, nagyméretű alkalmazások készítésére is alkalmas, moduláris M-Prolog rendszer fejlesztése. Ezt a rendszert lehet az első kereskedelmi forgalmi kritériumoknak is eleget tevő Prolog rendszernek tekinteni. A SOL-ban megkezdődött egy hosszú fejlesztési folyamat, mely először az M-Prologon alapuló T-Prolog új elvű szimulációs rendszer, majd a Cs-Prolog különböző változatainak elkészítését jelentette. Ezek a rendszerek két alapvető újdonságot tartalmaztak: megengedték több virtuális Prolog program (processz) egyidejű futását és azok üzeneteken keresztül történő kommunikációját, valamint bevezették a szimulációs idő fogalmát és

lehetővé tették a szimulált időben történő visszalépést. Ez utóbbinak egyik érdekes tulajdonsága olyan „jövőutazások" megvalósíthatósága volt, melynek segítségével a szimulált „jövőből" a modell információt tudott tárolni, melyet az időben való visszalépés után a „jelenben" is fel tudott használni.

1981-ben bejelentették az 5. generációs japán projektet. Ez a 10 éves, 10 milliárd dolláros projekt alapvetően az intelligens számítástechnikai rendszerek megvalósítását tűzte ki céljául, és hardver platformként a nagy teljesítményû, sokprocesszoros Prolog-gépeket választotta. A projekt egy másik nem titkolt céja a japán dominancia megteremtése volt az információtechnológiában. A projektet titokban már 1979-ben elkezdték szervezni és ennek során begyűjtöttek minden logikai programozással kapcsolatos eredményt. Így került be az eredeti anyagba több helyen is az SZKI TProlog rendszere, mint egy potenciális jelölt a sokprocesszoros Prolog-gépek programozására. A projekt egyik fő célját sem érte el maradéktalanul, azonban sok hasznos közbülső eredménye volt. Egyik sajátos „mellékhatása" a nagy európai K+F keretprogramnak, az ESPRIT-nek a létrejötte lett. Mind az Egyesült Államok, mind pedig az akkori Közös Piac ugyanis komolyan vette a japán kihívást és ellenlépésként olyan szervezeti kereteket hozott létre, melyek lehetővé tették nagyméretű (akár soknemzetű) projektek indítását és menedzselését.

A japán 5. generációs program közvetlen hatással volt a hazai fejlesztésekre is. Nagymértékben megnőtt a Prolog rendszerek iránti igény, és ebben a helyzetben jól startolt az M-Prolog. A következő években Kanadában létrejött az M-Prolog fejlesztését és értékesítését támogató LOGICWARE, amelynek segítségével mintegy 1500 M-Prolog rendszer került értékesítésre szerte a világon. Ez volt a hazai szoftverfejlesztés történetében az első eset, hogy tisztán magyar fejlesztésű szoftver jelentős tételben eladásra került Nyugat-Európában és az Egyesült Államokban.
Az M-Prolog egy „igazi" Prolog rendszer, amelynél a változóknak, ahogy az a logikában szokásos, nincs típusuk. Ezt azért érdemes megjegyezni, mert compiler készítés esetén ennek a tulajdonságnak a megtartása hatékonyságcsökkenést okoz. 1984-ban a Borland cég megjelentette a Turbo Prologot, mint a „Turbo" család egyik tagját, és két hónapon belül 300 000 példányt adott el a többi Prolognál egy nagyságrenddel gyorsabb termékéből. A gyorsulás annak volt köszönhető, hogy a Turbo Prolognál nem tartották be a típusmentes implementáció szokását, azonban mint az eladások mutatták, a piacot ez nem zavarta. Az SZKI-ban eközben, a nagyméretű Prolog-alkalmazásokhoz, elkészült egy speciális PC-be

betehető „Prolog processzor" kártya, mely 2MB memóriával rendelkezett, az akkor szokásos 640Kb helyett. Az M-Prolog alaprendszerre a későbbiekben további kiegészítések készültek, melyek közül a legfontosabb az M-Prolog Shell volt. Az M-Prolog Shell egy szakértő keretrendszer jellegű kiterjesztés, melyben több konkrét szakértő rendszert is implementáltak. Az 1990-es évek elején azonban az MPrologNak, mint önálló terméknek a tényleges továbbfejlesztése gyakorlatilag megszűnt.

1990-ben az Elméleti Labor kivált az SZKI-ból és IQSOFT Rt. néven alakult újjá, ahol a megalakulás óta folyik logikai programozással kapcsolatos kutatás-fejlesztési munka. Az 1990-es évek elején a Bank Austria számára készítettek egy banki szakértői rendszert, majd ennek általánosításaként, egy tudás alapú alkalmazásokat támogató eszközcsomagot dolgoztak ki, az EU Copernicus CUBIQ projektje keretében. A cégnél több logikai programozási alkalmazás is készült az elmúlt években, például egy nagy adatbázis intelligens származtatási részét Prologban írták meg. Az IQSOFT 1996 óta részt vesz a napjainkban egyik legnépszerûbb Prolog rendszer, a Sicstus Prolog fejlesztésében. Legújabb K+F tevékenységük a logikai programozás egy új ágához, a CLP-hez kapcsolódik: a TRACE Copernicus projekt keretében C++ nyelven készítettek egy CLP könyvtárat, míg a TACIT ESPRIT projektben a CLP segítségével egy gyártásütemezési feladatot oldottak meg.

A sokprocesszoros Prolog rendszerek fejlesztése 1986-ban részben átkerült az Alkalmazott Logikai Laboratórium Kisszövetkezetbe (ALL), majd 1988-ban, megalakulását követően, az SZKISZÁMALK közös vállalataként létrejött MULTILOGIC Kft.-be. 1988-1991 között elkészült a Transputereken futó CS-PROLOG interpreter és compiler, amely egyprocesszoros környezetben DOS, OS2 és UNIX alatt is futott. A Transputerek PC-kbe, munkaállomásokba helyezhető sokprocesszoros kártyák voltak (max. 10 processzor kártyánként, max. 8Mbyte memória processzoronként). A Transputereket az angol INMOS cég gyártotta, és sokáig az amerikai processzorok európai alternatívájának tekintették, azonban a 90-es évek közepére gyártásuk gyakorlatilag megszűnt, miután a gyártási jogokat átvette az SGS-THOMSON. 1992-ben megszûnt a MULTILOGIC és két éven keresztül a CS-PROLOG rendszerek továbbfejlesztése átkerült az ALL kisszövetkezetbe. Ekkor készült el egy real-time kiterjesztése a CS-PROLOGnak a francia hadügyminisztérium megrendelésére. 1994-ben megalakult az ML Tanácsadó és Informatikai Kft., melynek nevében az ML a korábbi MULTILOGIC-ra utal. A korábbi

CS-PROLOG-fejlesztők által létrehozott ML-ben, a régi felhasználók érdeklődése nyomán, 1995-ben felmerült a CS-PROLOG továbbfejlesztésének gondolata. A lehetőséget 1997-1998-ban egy PHARE-COPERNICUS projekt biztosította, melynek keretében elkészült a CS-PROLOG II UNIX alapú, hálózati elosztott Prolog rendszer. Ez a megvalósítás már hatékony SQL adatbázis és WEB browser-es interfésszel is rendelkezett. A rendszer egy nagyméretű elosztott szakértőrendszer- alkalmazáson lett kitesztelve. A CS-PROLOG- ban is készült szakértői keretrendszer, az ALL Kisszövetkezettel közösen készített ALLEX shell. Lérdekesebb alkalmazása a tervezett 1996-os világkiállítás kockázatbecslő szakértő rendszere volt.

Az előzőekben leírt Prolog-fejlesztéseken kívül még további, elsősorban akadémiai jellegű logikai programozási tevékenység is folyt Magyarországon. Itt kell megemlíteni az ALL Kisszövetkezetben készült LOBO - nem Horn-formulákon és rezolúciós elven alapuló - logikai programozási nyelvet, valamint a KFKI-ban készült data flow alapú 3DPAM absztrakt Prolog gépet. Ezenkívül a JATE-n folyt Prolog programok példák alapján történő generálásával kapcsolatos induktív logikai programozás. Az MProlog- és CS-PROLOG-fejlesztéseket mind itthon, mind pedig külföldön megbecsülték. Ezt mutatja a fejlesztőknek adományozott Akadémiai Díj (1983) és Állami Díj (1988), valamint az a nagyszámú publikáció, amely ezekről a fejlesztésekről készült. A rendszereket a világ sok intézményében használták a nyolcvanas évek közepén, kilencvenes évek elején és használóik még ma is szívesen emlékeznek rájuk.

A logikai programozás eredményei ma már ritkábban jelennek meg önálló programnyelvként, ám a számítástudomány más jelentős eredményeihez hasonlóan beépülnek azokba az általános eszközkészletekbe, programnyelvekbe, amelyeket mindennaposan használunk.

1.2. A feladat világa

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 micimacko, mez, aladar). 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. "Micimackó", "méz", "Aladár").

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.

```
allat(X) if emlos(X).
```

X akkor állat, ha emlős.

```
kirandulunk if jo_az_ido and van_penzunk.
```

Akkor kirándulunk, ha jó az idő és van pénzünk.

```
huzza(ag,X) if ember(X) and szegny(X).
```

Az utolsó szabály értelmezése: az ág is húzza azt a valakit akkor, ha ember és szegény. "Magyarra" lefordítva: szegény embert az ág is húzza.

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:

```
allat(macska).
```

A macska állat.

```
egyenlo(X,X).
```

X egyenlő X-szel.

```
szereti(micimacko,mez).
```

Micimackó szereti a mézet.

```
foz(anya,lecso).
```

Anya lecsót főz.

```
eszik(bloki,csont).
```

Blöki csontot eszik.

A megoldandó **kérdést** leíró programrészt **feladatnak** nevezzük.

Nézzük meg e fogalmak megfelelőit egy egyszerű példán! A következő Prolog-programrészlet megadja, hogy a Mézga családban ki Kriszta testvére.

```
(1)   apja("Geza","Kriszta").
(2)   anyja("Paula","Kriszta").
(3)   szuloje(X,Y) if apja(X,Y) or anyja(X,Y).
(4)   szuloje(Valaki,"Kriszta")
```

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 Kriszta apja Géza és anyja Paula.

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 Kriszta szülei? Nyilvánvaló, hogy ez Gézára és Paulára, azaz az őket képviselő "Geza" és "Paula" 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 Turbo 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
kartya=treff
```

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

Az elmondottak szerint a kartya olyan típus, ahol az egyetlen behelyettesíthető érték a „treff” 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:

kartya=treff; kor; karo; pikk

Most a kartya típus négyféle 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 "A feladat világa" című 1.2. fejezetben 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,"Kriszta") and apja(X,"Aladar").
```

Ennek a feladatnak a jelentése: van-e olyan X személy, aki egy személyben Kriszta és Aladár 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 (szereti(micimacko,_), 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:   szereti(micimacko,mez)
```

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
        hal(s)
        vanneki(s,s)
clauses
        hal(ponty).
        hal(csuka).
        vanneki(ponty,pikkely).
```

E szabályok alapján feltehetjük a kérdést:

```
        goal: hal(macska)
```

```
        goal: hal(ponty)
```

```
        goal: hal(X)
```

Az első kérdésre a válasz No, a másodikra Yes, a harmadikra X=ponty, X=csuka.

A válasz a következőképpen született: a Prolog az első esetben megvizsgált minden hal(...) 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 hal(...) 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 hal(_) értéke Yes.

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

```
vanneki(X,pikkely) if hal(X).
```

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

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

vanneki(A,pikkely) kiértékelésekor vanneki(X,pikkely) 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

A **tagadás** képzése a **not** beépített függvénnyel történhet. Ha teljesül a belső kifejezés, akkor hamissá válik az érték, míg a belső kifejezés kudarca esetén lesz kielégítve a szabály.

Egészítsük ki az előző feladatban szereplő szabályokat a hal(angolna). sorral. Mivel az angolnának nincs pikkelye, a vanneki szabály módosul.
Valaminek van pikkelye, ha hal, és nem angolna. Ebben segít a not(érték) szabály, mely a benne lévő állítás igazság-értékét fordítja meg:

```
vanneki(X,pikkely) if hal(X) and not(X=angolna).
```

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.
Példánkban tegyük fel a vanneki(X, pikkely) kérdést. hal(X)-re talált X=ponty megoldást. Ezután a következő állításra lép: not(X=angolna). Mivel a ponty=angolna tényleg hamis, ez az állítás is teljesül. A szabálynak vége, X=ponty kiírható. Ezután a Prolog visszalép, és keresi a következő jó megoldást. X=csuka is megfelel. Végül X=angolna helyettesítéssel lép tovább, not(angolna=angolna) most nem teljesül. Ezért a Prolog visszalép eggyel, és újabb megoldást keres hal(X)-re. Nincs több, így leáll.

Lássunk egy további egyszerű példát a tagadásra! A programban felsorolunk állatokat, hímeket, nőstényeket, utódokat:

```
gyereke(macska,”kölyök”).
gyereke(kakukk,"fióka").
gyereke("vaddisznó",malac).
gyereke("ló","csikó").
gyereke(„bölény”,"borjú").
gyereke(„veréb”,"fióka").
gyereke(„őz”,gida).
```

Megadjuk a „madárság” feltételét. (Miszerint: madár egy állat akkor, ha a gyereke fióka.)

```
madar(Allat) if allatok(Allat) and gyereke(Allat,"fióka").
```

Ezt a szabályt felhasználva adjuk meg (ezen a halmazon) az emlősök csoportját! A zárójelben tett megjegyzés könnyítés, mert az adott halmazban csak emlősök és madarak szerepelnek (pl. hüllők nem!); tehát amelyik állat nem madár, az emlős.
Ennek megfelelően:

```
emlos(Allat) if allatok(Allat) and not(madar(Allat)).
```

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(Allat) if madar(Allat) and !.
```

Ekkor a madarak csoportjából csak az elsőt írja ki.

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:

```
keszit(balazs,hf).
keszit(hugo,hf).
keszit(geza,hf).
```

Ha a kérdés a következő: (Ki készít házi feladatot?)

```
goal
        keszit(X,hf) and write(X) and nl.
```

akkor a Prolog kiírja a listában az első illeszkedőt: Balázs.

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 készítette el a házi feladatot.

```
goal
        keszit(X,hf) and write(X) and nl and fail or
                write("Senki sem készített házi feladatot").
```

A program az összes megoldást kiírja:
X=balazs
X=hugo
X=geza.

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. A tanulásra konkrét példát a 2.6. feladatban találhatunk.

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. Csoki

A feladat

Gombóc Artúr vásárolni indul. Milyen csokit vegyen? Megmondja a program. Természetesen csak olyan csokit vehet, ami kapható és finom. Ha nem tud megfelelő csokit vásárolni, akkor otthon marad és kiveszi a szekrényből azt a csokit, ami van.

A feladat megoldása

Ebben a programban Gombóc Artúr kétféleképp szerezheti be napi csokiadagját: vagy lemegy a boltba és kiválaszt egy csokit, vagy otthon marad és azt a csokit eszi, ami van.

A két lehetőséget a következőképp írhatjuk le Prolog nyelven:

```
csokolade if vasarol(Csoki) and nl and write("Menj a boltba es vegyel ") and
        write(Csoki) and write(" csokit!") and nl.
csokolade if otthon_marad and nl and write("Maradj otthon es egyel ") and
        write("olyan csokit, ami van!") and nl.
```

Ahhoz, hogy ki tudjuk választani a megfelelő csokit, tudnunk kell, melyik csoki finom és kapható-e.

```
vasarol(Csoki) if finom(Csoki) and kaphato(Csoki).
```

A program ismeri Artúr ízlését, és számon tartja a finom csokikat,

```
finom(mandulas).
finom(mogyoros).
finom(epres).
finom(oszibarackosos).
```

valamint azt is tudja, hogy pontosan mely csoki kaphatók.

```
kaphato(tej).
kaphato(kokuszos).
kaphato(epres).
kaphato(mogyoros).
```

A program tudásához az is hozzátartozik, hogy otthon maradni minden feltétel nélkül lehet. Ebben a programban egy változó van, ez a Csoki. 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,"Csoki",1,0,23,80) and
write("Gomboc Artur vasarolni indul. Milyen csokit vegyen?") and nl and
write("Megmondja a program.") and nl and
write("Termeszetesen csak olyan csokit vehet, ami kaphato es finom.") and nl and
write("Ha nem tud megfelelo csokit vasarolni, akkor otthon marad") and nl and
write("es kiveszi a szekrenybol azt a csokit, ami van.") and nl and
readchar(W) and csokolade and fail.
```

A teljes program

```
domains
        mi=string.
predicates
        vasarol(mi).
        finom(mi).
```

```
        kaphato(mi).
        otthon_marad.
        csokolade.
clauses
/*vagy lemegy a boltba csokit venni, vagy otthon marad*/
        csokolade if vasarol(Csoki) and nl and write("Menj a boltba es vegyel ") and
                write(Csoki) and write(" csokit!") and nl.
        csokolade if otthon_marad and nl and write("Maradj otthon es egyel ") and
                write("olyan csokit, ami van!") and nl.
/*akkor vasarol csokit, ha finom es kaphato*/
        vasarol(Csoki) if finom(Csoki) and kaphato(Csoki).
/*a finom csokik:*/
        finom(mandulas).
        finom(mogyoros).
        finom(epres).
        finom(oszibarackos).
/*a kaphato csokik:*/
        kaphato(tej).
        kaphato(kokuszos).
        kaphato(epres).
        kaphato(mogyoros).
        otthon_marad.
goal
        clearwindow and makewindow(1,11,15,"Csoki",1,0,23,80) and
        write("Gomboc Artur vasarolni indul. Milyen csokit vegyen?") and nl and
        write("Megmondja a program.") and nl and
        write("Termeszetesen csak olyan csokit vehet, ami kaphato es finom.") and nl and
        write("Ha nem tud megfelelo csokit vasarolni, akkor otthon marad") and nl and
        write("es kiveszi a szekrenybol azt a csokit, ami van.") and nl and
        readchar(W) and csokolade and fail.
```

A program működése: mintaillesztés, visszalépés

Nézzük meg, hogyan történik a csokis 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, a csokolade. Ha a program nem rendelkezi goal résszel, nekünk kell begépelnünk egy célállítást, jelen esetben ez a

```
goal:   csokolade
```

A Prolog következtetési mechanizmusa megkeresi a csokolade definícióját, meghatározását, és megpróbálja a feladatot egyeztetni a definícióval. Az egyeztetést az ún. **mintaillesztéssel** hajtja végre. A feladatunkban a csokolade 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:   csokolade
```

célállítás és a

```
csokolade if vasarol(Csoki) and nl and write("Menj a boltba es vegyel ") and
                write(Csoki) and write(" csokit!") 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 vasarol(Csoki) 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 vasarol(Csoki) részfeladattal kell foglalkoznunk. Eljárásunk ugyanaz, mint az eredeti feladatnál, keresünk hozzá egy definíciót:

```
vasarol(Csoki) if finom(Csoki) and kaphato(Csoki).
```

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 (véletlenül mindkettőben Csoki). Így a vasarol(Csoki) feladatot két részfeladatra bontottuk:
az első: finom(Csoki) és
a második: kaphato(Csoki).

A finom(Csoki) részfeladatot a megfelelő definíció első Prolog állításával, a

```
finom(mandulas).
```

tényállítással illeszthetjük. A mandulas konstans, a Csoki 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 Csoki változó felveszi a mandulas értéket. Azonban a Csoki változónak további előfordulásai is vannak a vasarol definíciójában:

```
vasarol(Csoki) if finom(Csoki) and kaphato(Csoki).
```

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 Csoki változó helyett már a mandulas értéket kell figyelembe vennünk, azaz ezt a részfeladatot kell megoldanunk:

```
kaphato(mandulas).
```

Ez azonban nem fog sikerülni, mivel a kaphato definíciójában sehol sem szerepel a mandulas, 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

```
finom(Csoki)
```

részfeladat megoldására most próbálkozzunk a definíció második tényállításával:

```
finom(mogyoros).
```

Ekkor a mintaillesztés után a Csoki felveszi a mogyoros értéket, és a

```
kaphato(mogyoros).
```

részfeladat is megoldható.
Ezzel teljesítettük a vasarol definíció mindkét feltételét, így a vasarol(Csoki) is megoldódott, mégpedig úgy, hogy a Csoki felvette a mogyoros értéket.

Az **nl** beépített eljárás vagy predikátum hatására soremelés történik, a **write**(„Menj a boltba es vegyel”) hatására a rendszer kiírja az idézőjelek közötti szöveget. A **write**(Csoki) kiírja a rendszer a Csoki változó aktuális értékét (most a mogyoros). A **write**(„csokit!”) kiírja az idézőjelek közötti szöveget, az **nl** hatására pedig sort emel.

Ezzel még mindig nem vagyunk kész, mivel a **goal** részben a csoki után még szerepe a **fail** szó, amely visszalépést eredményez. Miután a Prolog rendszer sikeresen megoldotta a feladatot, kiírja, hogy mogyoros csokit kell vásárolnunk, 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 epres csokit is kell vásárolni, 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 csokit, amit vásárolhatunk, kiírja, hogy „Maradj otthon es egyel olyan csokit, ami van!”. Ezután kimerült az összes lehetőség és a rendszer leáll.

2.2. Mézga

A feladat

A program a Mézga családon 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

Rögzítjük, hogy ki férfi és ki nő a szereplők közül,

```
ferfi("Oreg Geza").
ferfi("Geza").
ferfi("Aladar").
ferfi("Maris szomszed").
ferfi("Hufnager Pisti").
no("Paula").
no("Kriszta").
```

továbbá felsoroljuk a Mézga családon belüli apa-gyermeke és anya-gyermeke kapcsolatokat.

```
apja("Oreg Geza","Geza").
apja("Geza","Aladar").
apja("Geza","Kriszta").
anyja("Paula","Aladar").
anyja("Paula","Kriszta").
```

Minden más kapcsolatot ezek segítségével határozunk meg:

Valaki akkor ember, ha férfi vagy nő.

```
ember(Valaki) if ferfi(Valaki) or no(Valaki).
```

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 gyermektelen, ha ember, azonban senkinek sem szülője.

```
gyermektelen(Valaki) if ember(Valaki) and not(szuloje(Valaki,_)).
```

Valaki családtag, ha vagy szülője valakinek, vagy valaki az ő szülője.

```
csaladtag(Valaki) if szuloje(Valaki,_) or szuloje(_,Valaki).
```

Valaki nővére egy személynek, ha asszony és van közös szülőjük.

```
nover(Nover,Valaki) if no(Nover) and szuloje(Szulo,Nover) and
                               szuloje(Szulo,Valaki) and not(Nover=Valaki).
```

Valaki bátyja egy személynek, ha férfi és van közös szülőjük.

```
fiutestver(Baty,Valaki) if ferfi(Baty) and szuloje(Szulo,Baty) and
                                  szuloje(Szulo,Valaki) and not(Baty=Valaki).
```

A teljes program

```
domains
        valaki=string.
predicates
        ferfi(valaki).  /*valaki ferfi*/
        no(valaki).  /*valaki no*/
```

```
        apja(valaki,valaki).  /*ki kinek az apja*/
        anyja(valaki,valaki).  /*ki kinek az anyja*/
        ember(valaki)  /*valaki ember*/
        szuloje(valaki,valaki).  /*ki kinek a szuloje*/
        nagyszulo(valaki,valaki).  /*ki kinek a nagyszuloje*/
        ose(valaki,valaki).  /*ki kinek az ose*/
        gyermektelen(valaki).  /*valaki gyermektelen*/
        csaladtag(valaki).  /*valaki csaladtag*/
        nover(valaki,valaki).  /*ki kinek a novere*/
        fiutestver(valaki,valaki).  /*ki kinek a fiutestvere*/
clauses
        ferfi("Oreg Geza").
        ferfi("Geza").
        ferfi("Aladar").
        ferfi("Maris szomszed").
        ferfi("Hufnager Pisti").
        no("Paula").
        no("Kriszta").
        apja("Oreg Geza","Geza").
        apja("Geza","Aladar").
        apja("Geza","Kriszta").
        anyja("Paula","Aladar").
        anyja("Paula","Kriszta").
/*akkor ember, ha ferfi vagy no*/
        ember(Valaki) if ferfi(Valaki) or no(Valaki).
/*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*/
        nagyszulo(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).
/*gyermektelen, ha senkinek sem szuloje*/
```

```
        gyermektelen(Valaki) if ember(Valaki) and not(szuloje(Valaki,_)).
/*csaladtag, ha vagy szuloje valakinek, vagy valaki az o szuloje*/
        csaladtag(Valaki) if szuloje(Valaki,_) or szuloje(_,Valaki).
/*novere, ha no es van kozos szulojuk*/
        nover(Nover,Valaki) if no(Nover) and szuloje(Szulo,Nover) and
                szuloje(Szulo,Valaki) and not(Nover=Valaki).
/*fiutestvere, ha ferfi es van kozos szulojuk*/
        fiutestver(Baty,Valaki) if ferfi(Baty) and szuloje(Szulo,Baty) and
                szuloje(Szulo,Valaki) and not(Baty=Valaki).
```

A program működése

Ez a program nem tartalmaz **goal** részt, így a nekünk kell begépelnünk mindazt, amit tudni szeretnénk Mézgáékkal kapcsolatban.

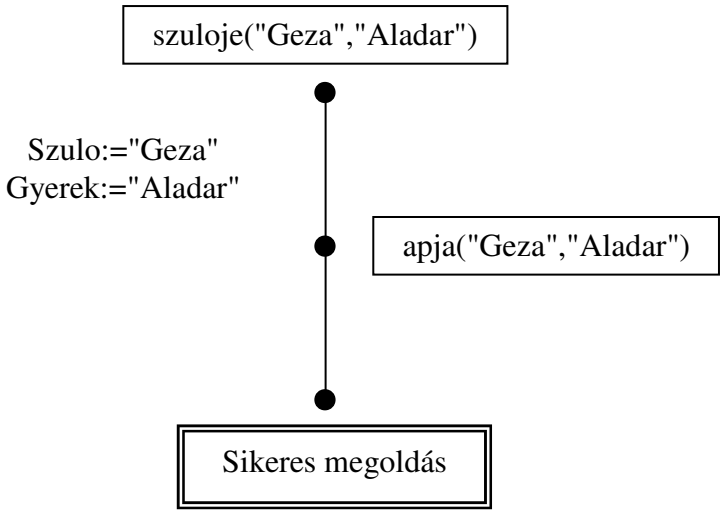
Először megnézzük, hogy Géza szülője-e Aladárnak. Csupán ennyit kell begépelnünk:

```
        szuloje("Geza","Aladar")
```

A szuloje meghatározása

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

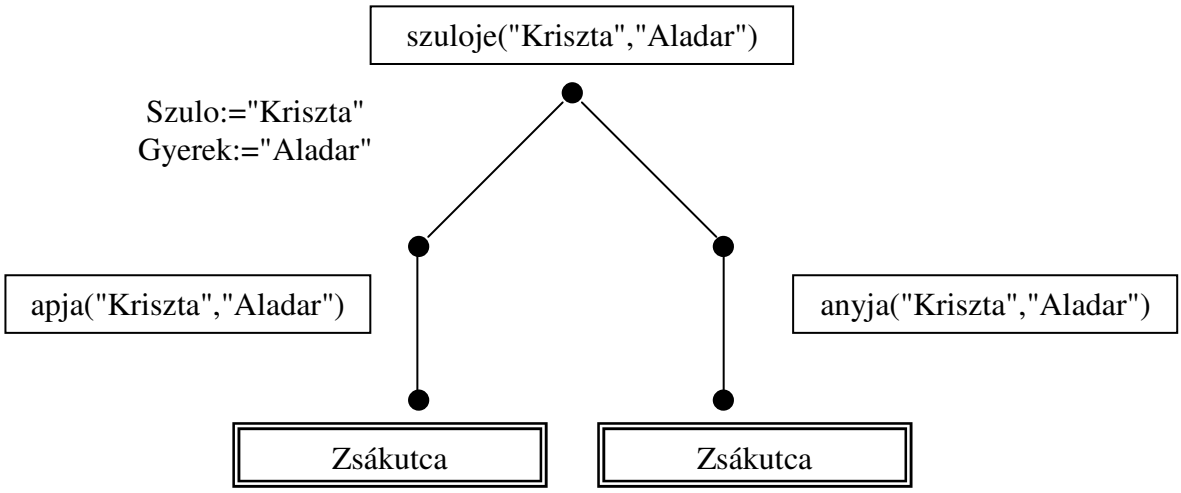
alapján a Prolog rendszer a következőképp fog működni: a Szulo változó illeszkedik a "Geza" konstanssal, a Gyerek az "Aladar" konstanssal. Az első rész meghívásakor – apja("Geza",Gyerek) – mivel van olyan tényállításunk, ahol az első argumentum "Geza" – apja("Geza","Aladar")– az illesztés sikeres, és a Gyerek változó felveszi az "Aladar" 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("Kriszta","Aladar")
```

célállítás ugyanolyan kérdést tesz fel, mint az első, de itt a válasz negatív lesz, azaz Kriszta nem szülője Aladárnak. A Szulo="Kriszta" és a Gyerek="Aladar" 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=Oreg Geza

1 Solution

tehát Gézának egy szülője van: Öreg Géza.

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=Aladar

X=Kriszta

2 Solutions

Itt két megoldása is van a feladatnak.

Nézzük meg, mit fejez ki egy olyan célállítás, amelyben csak változók szerepelnek?

```
szuloje(X,Y)
```

Ez a célállítás azt jelenti, hogy szülő-gyerek kapcsolatot (kapcsolatokat) keresünk a családban. A rendszer ekkor kiírja, hogy:

X=Oreg Geza Y=Geza

X=Geza Y=Aladar

X=Geza Y=Kriszta

X=Paula Y=Aladar

X=Paula Y=Kriszta

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

```
nagyszulo(X,Y)
```

Ekkor két megoldást kapunk:

X=Oreg Geza Y=Aladar

X=Oreg Geza Y=Kriszta

2 Solutions

Tehát egy nagyszülőt találtunk, Öreg Gézát, akinek két unokája van, Aladár és Kriszta.

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 Öreg Géza őse-e Krisztának.

```
ose("Oreg Geza","Kriszta")
```

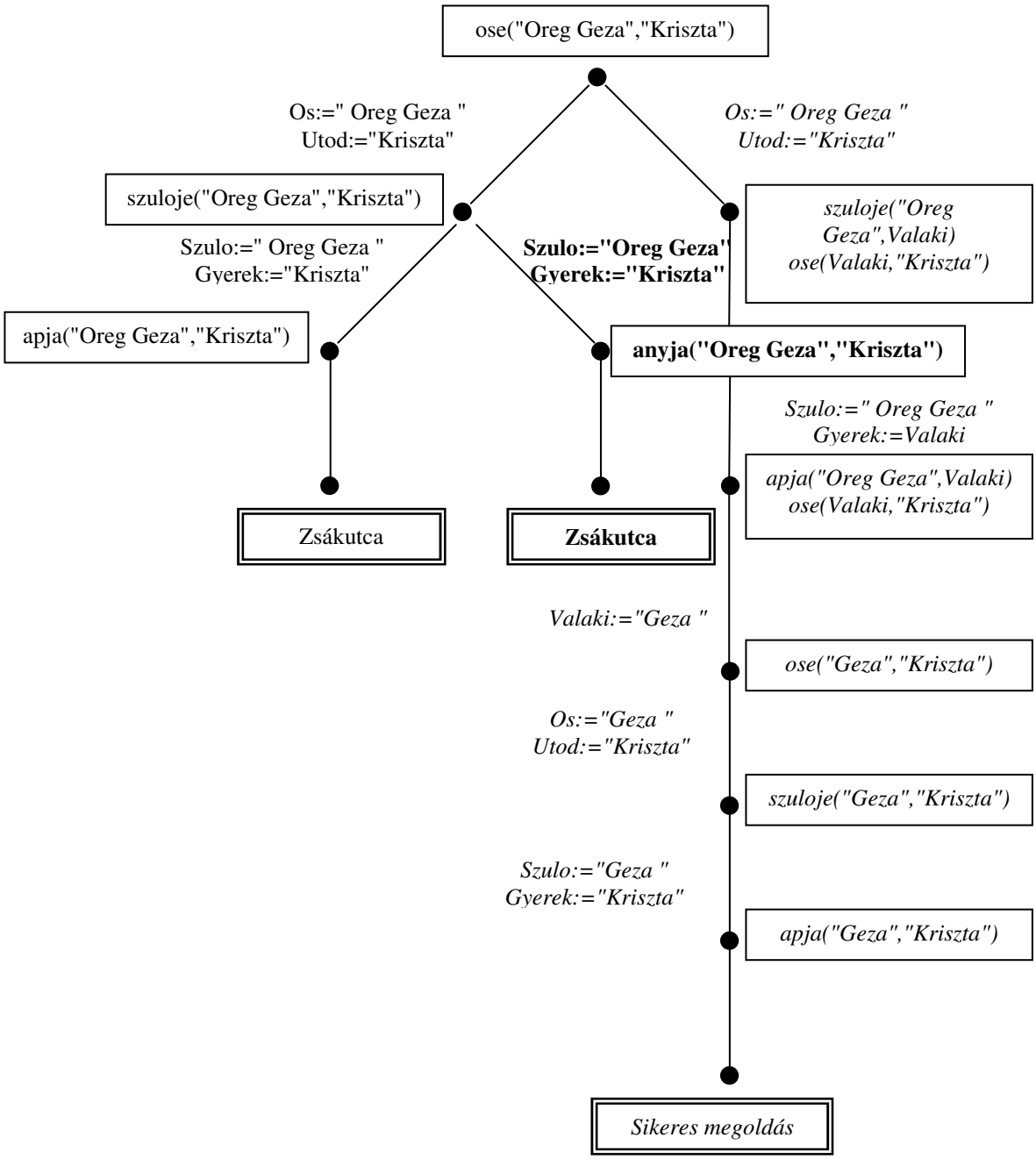
A program futása úgy kezdődik, hogy a célállítást illesztjük az első szabály bal oldalával. Illesztés után Os="Oreg Geza" és Utod="Kriszta", mivel az ose definíciója így szól:

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

Az illesztés után a szuloje("Oreg Geza","Kriszta") részfeladat megoldására kerül sor. Ha igaz lenne, hogy Öreg Géza Kriszta szülője, akkor a program Yes válasszal leállna. Azonban nem igaz sem az apja("Oreg Geza","Kriszta"), sem az anyja("Oreg Geza","Kriszta").

Mivel ezeket a részfeladatokat nem sikerült megoldani, a rendszer visszalép, és most az ose definíciójának második szabályával foglalkozik, tehát keres valakit, akinek Öreg Géza a szülője, ehhez megnézi, hogy van-e valaki, akinek Öreg Géza az apja, apja("Oreg Geza",_). Megtalálja Gézát, mivel az apja("Oreg Geza","Geza") a kapcsolatok között szerepel. Ekkor megnézi, hogy Géza Kriszta őse-e. Ez akkor teljesül például, ha szuloje("Geza","Kriszta"). Ez viszont igaz, mert a megadott tények szerint apja("Geza","Kriszta"). Ezzel a feladatot megoldottuk, azaz igazoltuk, hogy Öreg Géza Kriszta őse. A rendszer kiírja, hogy Yes és leáll.

A megoldás menetét az alábbi ábrán követhetjük nyomon:



A következő példában azt szeretnénk megtudni, hogy kiknek nincs gyermekük a felsorolt személyek közül:

```
gyermektelen(X)
```

Ekkor a következő megoldást írja ki a rendszer:

X=Aladar

X=Maris szomszed

X=Hufnager Pisti

X=Kriszta

4 Solutions

Vajon Hufnáger Pisti családtag-e? A kérdés megválaszolásához csupán az alábbiakat kell beírni:

```
csaladtag("Hufnager Pisti")
```

és rögtön megkapjuk a választ: No, azaz Hufnáger Pisti nem tartozik a Mézga családhoz.

A következőkben azt szeretnénk megtudni, hogy Kriszta kinek a lánytestvére:

```
nover("Kriszta",X)
```

A válasz: X=Aladár, azaz egyedül Aladárnak nővére Kriszta.

Kik azok a személyek, akiknek van fiútestvére?

```
fiutestver(X,Y)
```

Ekkor megtudjuk, hogy X=Aladar, Y=Kriszta a feladat megoldása.

2.3. Szoba1

A feladat

A program két egybenyíló szobából álló lakást tervez. A két szobát 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 lakást a bejárati ajtóval, a két szoba között elhelyezkedő átjáróval és a két ablakkal jellemezzük. Ezek irányának megadása jelenti a feladat megoldását. Azt a szobát, mely bejárati ajtóval rendelkezik, bejaratosnak neveztük el. Ezt a bejárati ajtó, a két szoba közötti átjáró és egy ablak jellemez. A másik szobát szobanak nevezzük, ezt egy átjáró és egy ablak jellemez. A szemben(Atjaro,At) azt jelenti, hogy a bejárati ajtóval rendelkező szoba átjáró ajtajának iránya legyen szemben a másik szoba átjárójának irányával, vagyis a két szoba illeszkedjen egymáshoz. A két szoba ablaka pedig ne legyen egymással szemben levő falon, ahogy ezt kikötöttük.

```
lakas(Ajto,Atjaro,Ablak1,Ablak2) if bejaratos(Ajto,Atjaro,Ablak1) and
                szoba(At,Ablak2) and szemben(Atjaro,At) and
                not(szemben(Ablak1,Ablak2)).
```

A bejárati ajtóval rendelkező szoba legyen olyan szoba, 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 szoba(Atjaro,Ablak) and irany(Ajto) and
                Ajto<>"eszak" and Ajto<>Ablak and Ajto<>Atjaro.
```

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

```
szoba(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
        lakas(hol,hol,hol,hol).  /*a lakast a bejarati ajtoval, a ket szoba kozotti atjaroval es a
                                        ket ablakkal jellemezzuk*/
        bejaratos(hol,hol,hol).  /*ebben a szobaban van a bejarati ajto*/
        szoba(hol,hol).  /*ez a masik szoba*/
        szemben(hol,hol).  /*egymassal ellentes iranyok*/
        irany(hol).  /*iranyok megadasa*/
clauses
/*ket szobas lakas, a szobak illeszkednek egymassal, ket ablak nincs egymassal szemben
levo falon*/
        lakas(Ajto,Atjaro,Ablak1,Ablak2) if
                bejaratos(Ajto,Atjaro,Ablak1) and
```

```
                szoba(At,Ablak2) and
                szemben(Atjaro,At) and
                not(szemben(Ablak1,Ablak2)).
/*a bejarati ajtoval rendelkezo szoba olyan szoba, 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
                szoba(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*/
        szoba(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,"Szoba2",1,0,23,80) and
        write("2 szobas lakast akarunk tervezni. \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
lakas(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. Szoba2

A feladat

A program három egybenyíló szobából álló lakást tervez. A három szobát 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 szobák 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 szobán egy, a másik kettőn két ablak legyen.

A feladat megoldása

A feladat hasonlít a két szobás lakás tervezésével foglalkozó feladathoz. A lakást ezúttal a bejárati ajtóval, a szobák között elhelyezkedő átjárókkal és az öt ablakkal jellemezzük. Ezek irányának megadása jelenti a feladat megoldását. Azt a szobát, mely bejárati ajtóval rendelkezik, újból bejaratosnak neveztük el. Ezt a bejárati ajtó, a két szoba közötti átjáró és egy ablak jellemez. A másik két szobát szobanak nevezzük, ezt egy-egy átjáró és két-két ablak jellemez. A szemben(Atjaro1,At1) stb. definíciók azt jelentik, hogy az adott szoba átjáró ajtajának iránya legyen szemben a másik szoba átjárójának irányával, vagyis a két szoba illeszkedjen egymáshoz. Mivel esetünkben három szoba van, így a feltételek bővülnek.

```
lakas(Ajto,Atjaro1,Atjaro2,Ablak1,Ablak2,Ablak3,Ablak4,Ablak5) if
        bejaratos(Ajto,Atjaro1,Atjaro2,Ablak1) and
        szoba(At1,Ablak2,Ablak3) and szoba(At2,Ablak4,Ablak5) and
        szemben(Atjaro1,At1) and szemben(Atjaro2,At2) and
        szemben(At1,At2).
```

A bejárati ajtóval rendelkező szobában 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 szoba 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.

```
szoba(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
        lakas(hol,hol,hol,hol,hol,hol,hol,hol). /*a lakast a bejarati ajto, a ket atjaro es az ot
                                                ablak jellemzi*/
        bejaratos(hol,hol,hol,hol).  /*a bejarati ajtoval rendelkezo szoba*/
        szoba(hol,hol,hol).  /*a masik ket szoba*/
        szemben(hol,hol).  /*egymassal ellentetes iranyok*/
        irany(hol).  /*iranyok megadasa*/
clauses
/*harom szobas lakas, ahol a szobak illeszkednek egymashoz*/
        lakas(Ajto,Atjaro1,Atjaro2,Ablak1,Ablak2,Ablak3,Ablak4,Ablak5) if
                bejaratos(Ajto,Atjaro1,Atjaro2,Ablak1) and
                szoba(At1,Ablak2,Ablak3) and
                szoba(At2,Ablak4,Ablak5) and
                szemben(Atjaro1,At1) and
                szemben(Atjaro2,At2) and
                szemben(At1,At2).
/*a bejarati ajtos szobaban a 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 szobaban az ablakok nem nezhetnek eszakra, egy falon csak egy valami lehet*/
        szoba(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,"Szoba3",1,0,23,80) and
        write("3 szobas lakast akarunk tervezni. \n") and
        write("Az alabbi igenyeket kell figyelembe vennunk: \n") and
        write("Ezek 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 szoban egy, a masik ketton ket ablak van.\n\n") and
        readchar(W) and
        lakas(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

Adott néhány személy 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(anna,budapest,kakukk,10).
cim(beni,budapest,kakukk,12).
cim(cecil,szeged,cinke,12).
cim(denes,szeged,kakukk,14).
```

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(anna,budapest,kakukk,10).
        cim(beni,budapest,kakukk,12).
        cim(cecil,szeged,cinke,12).
        cim(denes,szeged,kakukk,14).
        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("Ismerjuk nehany ember cimet:\n") and
        kiir and nl and
        write("Az alabbi program megadja, hogy ki kinek a szomszedja.\n\n") and
        readchar(W) and
        szomszed(X,Y) and write("szomszedok:  ",X,", ",Y) or
        write("nincsenek szomszedok").
```

2.6. Repülő

A feladat

Menetrendünk 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.

A feladat megoldása

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

```
jarat(budapest,varso).
```

Ez azt jelenti, hogy Budapestről Varsóba közvetlenül el lehet jutni.
A

```
repules(X,Y) if jarat(X,Y) or jarat(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 utazni X-be.

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

```
ut(X,Y) if repules(X,Y).
```

Ha nem, akkor az is lehet, hogy úgy is el tudunk jutni 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 a repülést Z városba, ha az szabad, azaz még nem jártunk ott.

```
ut(X,Y) if repules(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 voltunk(Z) and ! and fail.
        szabad(Z) if assert(voltunk(Z)).
```

Fő kérdésünk, hogy eljuthatunk-e 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.

```
        eljutunk(X,Y) if retractall(voltunk(_)) and assert(voltunk(X)) and ut(X,Y) and
                        X<>Y.
```

A teljes program

```
domains
        varos=string.
predicates
        jarat(varos,varos).     /*a kulonbozo varosok kozotti kozvetlen osszekottetesek*/
        ut(varos,varos).
        repules(varos,varos).
        szabad(varos).
        eljutunk(varos,varos).          /*eljuthatunk-e egyik varosbol a masikba - erre
                                        vagyunk kivancsiak*/
database
        voltunk(varos).
clauses
        jarat(budapest,varso).
        jarat(budapest,london).
        jarat(london,peking).
        jarat(london,boston).
        jarat(varso,moszkva).
        jarat(madrid,roma).
/*oda-vissza ervenyes*/
        repules(X,Y) if jarat(X,Y) or jarat(Y,X).
```

```
/*van ut ket varos kozott, ha van kozvetlen osszekottetes vagy ha van olyan varos, amit erintve el tudunk oda jutni, ahova szeretnenk*/
        ut(X,Y) if repules(X,Y).
        ut(X,Y) if repules(X,Z) and szabad(Z) and ut(Z,Y).
/*Jartunk-e mar az adott varosban? Nem akarunk ketszer ugyanoda visszaterni*/
        szabad(Z) if voltunk(Z) and ! and fail.
        szabad(Z) if assert(voltunk(Z)).
/*A kerdes: eljutunk-e egyik varosbol a masikba?*/
        eljutunk(X,Y) if retractall(voltunk(_)) and assert(voltunk(X)) and ut(X,Y) and
                X<>Y.
```

A program néhány futtatási eredménye

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 arra vagyunk kíváncsiak, hogy el lehet-e jutni Budapestről Bostonba, csupán annyit kell beírnunk, hogy

```
        eljutunk(budapest,boston)
```

A rendszer válasza: Yes.

Ha azt szeretnénk megtudni, hogy Madridból hová lehet repülni, ezt kell beírnunk:

```
        eljutunk(madrid,X)
```

A válasz:

X=roma

1 Solution

2.7. Étel

A feladat

Micimackó főzni tanul. Gondosan megnézi a szakácskönyvben, hogy kedvenc ételei miből készülnek, és hogy az egyes alapanyagokból mennyire van szükség. Közben lázasan gondolkodik:

a) Melyik az a két étel, amelyekhez ugyanazok az alapanyagok kellenek?

b) Melyik az az alapanyag, amely az összes ételhez szükséges?

c) Melyik az az alapanyag, amelyből a legtöbb kell valamelyik étel készítéséhez?

Segítsünk neki!

A feladat megoldása

Azt, hogy melyik étel milyen alapanyagból készül és abból mennyi szükséges az elkészítéshez, a következőképp jelöljük:

```
alapanyag(sultkrumpli,olaj,0.1).
alapanyag(sultkrumpli,so,0.01).
alapanyag(husleves,viz,1).
alapanyag(husleves,so,0.01).
alapanyag(tojasleves,viz,1.1).
alapanyag(tojasleves,so,0.01).
```

Felsoroljuk azokat az ételeket, amiket Micimackó el tud készíteni.

```
etel(sultkrumpli).
etel(husleves).
etel(tojasleves).
```

Ezeket ki is íratjuk a képernyőre. Hogy ne csak egy ételt 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 vagy részre lép, ahol a **!** (**cut**) hatására befejeződik ez a részfeladat. Ekkor már igaz értékkel tér vissza.

```
kiiras if etel(X) and write(X,"  ") and fail or !.
```

Nézzük az a) feladatot! Keresünk két ételt, amikhez ugyanazok az alapanyagok szükségesek. Ezt a részfeladatot azonban nem tudjuk közvetlenül megoldani, előbb azt kell meghatároznunk, hogy melyek azok az ételek, amik nem ugyanazokból az alapanyagokból készülnek. Majd ennek a tagadásával kapunk választ az a) feladatra.

```
nemugyanaz(E1,E2) if etel(E1) and etel(E2) and
        E1<>E2 and
        alapanyag(E1,A1,_) and not(alapanyag(E2,A1,_)).
ugyanaz(E1,E2) if etel(E1) and etel(E2) and
        E1<>E2 and not(nemugyanaz(E1,E2)).
```

Hasonlóan járunk el a b) feladat megoldásakor. Először meghatározzuk, hogy mely alapanyagok nem kellenek egy étel elkészítéséhez, majd ennek tagadásával megkapjuk, hogy mely alapanyagok kellenek az összes étel megfőzéséhez.

```
egyheznemkell(A) if etel(E) and not(alapanyag(E,A,_)).
osszeshezkell(A) if alapanyag(_,A,_) and not(egyheznemkell(A)).
```

A c) feladat a felhasznált alapanyagok mennyiségét hasonlítja össze. Az a legtöbb, ami minden más mennyiségnél több.

```
legtobb(A) if alapanyag(_,A,M1) and alapanyag(_,B,M2) and A<>B and
                M1>M2.
```

A teljes program

```
domains
        etelnev=string.
        anyagnev=string.
        mennyiseg=integer.
```

```
predicates
        alapanyag(etelnev,anyagnev,mennyiseg).
        etel(etelnev).
        kiiras.
        ugyanaz(etelnev,etelnev).  /*az a) kerdeshez*/
        nemugyanaz(etelnev,etelnev).
        osszeshezkell(anyagnev).  /*a b) kerdeshez*/
        egyheznemkell(anyagnev).
        legtobb(anyagnev).  /*a c) kerdeshez*/
clauses
/*az egyes etelekhez milyen alapanyag szukseges*/
        alapanyag(sultkrumpli,olaj,0.1).
        alapanyag(sultkrumpli,so,0.01).
        alapanyag(husleves,viz,1).
        alapanyag(husleves,so,0.01).
        alapanyag(tojasleves,viz,1.1).
        alapanyag(tojasleves,so,0.01).
/*ilyen eteleket tud fozni*/
        etel(sultkrumpli).
        etel(husleves).
        etel(tojasleves).
/*kiiratjuk az etelek nevet*/
        kiiras if etel(X) and write(X,"  ") and fail or !.
/*a) kerdes: megad ket etelt, amikhez ugyanazok az alapanyagok kellenek*/
        nemugyanaz(E1,E2) if etel(E1) and etel(E2) and
                E1<>E2 and
                alapanyag(E1,A1,_) and not(alapanyag(E2,A1,_)).
        ugyanaz(E1,E2) if etel(E1) and etel(E2) and
                E1<>E2 and not(nemugyanaz(E1,E2)).
/*b) kerdes: megad egy alapanyagot, ami az osszes etelhez kell*/
        egyheznemkell(A) if etel(E) and not(alapanyag(E,A,_)).
        osszeshezkell(A) if alapanyag(_,A,_) and not(egyheznemkell(A)).
/*c) kerdes: megad egy alapanyagot, amibol a legtobb kell*/
```

```
        legtobb(A) if alapanyag(_,A,M1) and alapanyag(_,B,M2) and A<>B and M1>M2.
goal
        clearwindow and makewindow(1,14,15," Etel ",0,0,25,80) and
        write("Micimacko fozni tanul. Az alabbi eteleket szeretne elkesziteni:\n") and
        kiiras and nl and
        write("Gondosan megnezi a szakacskonyvben, hogy\n") and
        write("kedvenc etelei mibol keszulnek,\n") and
        write("es hogy az egyes alapanyagokbol mennyire van szukseg.\n") and
        write("Kozben lazasan gondolkodik:\n") and
        write("a) Melyik az a ket etel, amelyekhez ugyanazok az alapanyagok kellenek?\n")
        and write("b) Melyik az az alapanyag, amely az osszes etelhez szukseges?\n") and
        write("c) Melyik az az alapanyag, amelybol a legtobb kell \n") and
        write("valamelyik etel keszitesehez?\n") and
        write("Segitsunk neki!\n\n") and
        readchar(W) and
        ugyanaz(E1,E2) and write("Ugyanabbol az alapanyagokbol keszulnek: ",E1,"  ",E2)
        and nl and
        osszeshezkell(A) and write("Az osszes etelhez kell: ",A) and nl and
        legtobb(A1) and write("A legtobb kell belole: ",A1).
```

2.8. Vitamin

A feladat

Gyógyszergyár vitaminkészítményei közül
a) add meg azokat a termékeket, melyek csak egyféle vitamint tartalmaznak,
b) azokat a termékeket, melyek az összes vitamint tartalmazzak,
c) azt a vitamint, amely a legtöbb termekben szerepel.

A feladat megoldása

Megadjuk, hogy milyen vitaminok szerepelhetnek a termékekben.

```
vitamin(a).
vitamin(b).
vitamin(c).
```

Azt is rögzítjük, hogy az egyes termékek milyen vitaminokat tartalmaznak.

```
mivanbenne(termek1,a).
mivanbenne(termek2,a).
mivanbenne(termek2,b).
mivanbenne(termek2,c).
```

Hogy szebb legyen a programunk, kiíratjuk, mely termék milyen vitaminokat tartalmaz.

```
kiiras if mivanbenne(X,Y) and write(X,", ami ",Y, " vitamint tartalmaz") and
        nl and fail or !.
```

Az a) részfeladat arra kérdez rá, hogy melyek azok a vitaminok, amik csak egyféle vitamint tartalmaznak. Erre a kérdésre közvetlenül nem tudunk válaszolni, ezért először meghatározzuk, hogy mely termékek tartalmaznak többféle vitamint. Ezután már könnyű dolgunk van, hiszen azok tartalmaznak csak egyféle vitamint, amik nem tartalmaznak többfélét.

```
tobbfele(X) if vitamin(Y) and vitamin(Z) and mivanbenne(X,Y) and
                mivanbenne(X,Z) and Y<>Z.
egyfele(X) if mivanbenne(X,_) and not(tobbfele(X)).
```

A b) részfeladat arra kíváncsi, hogy melyek azok a termékek, amik az összes vitamint tartalmazzák. Itt sem rögtön a feladatra válaszolunk, hanem meghatározzuk, hogy melyek azok a termékek, amik egy vitamint nem tartalmaznak. Az összes vitamint tartalmazza az a termék, amelyre igaz, hogy nincs olyan vitamin, amit ne tartalmazna.

```
egyetnem(X) if vitamin(Y) and not(mivanbenne(X,Y)).
osszes(X) if mivanbenne(X,_) and not(egyetnem(X)).
```

A c) kérdés az, hogy melyik az a vitamin, ami a legtöbb termékben szerepel. Ehhez először minden vitamin esetében ki kell számolni, hogy hány termék tartalmazza azt. Majd meghatározzuk, hogy mely termékre igaz, hogy nem a legtöbb termékben szerepel, vagyis hogy létezik olyan vitamin, ami több termékben megtalálható. E szabály tagadásával kapunk választ a kérdésünkre, vagyis a legtöbb termékben az a vitamin szerepel, amire nem igaz, hogy nem a legtöbben szerepel.

```
szamol(Y,DB) if vitamin(Y) and mivanbenne(X,Y) and
        not(marvolt(X)) and assert(marvolt(X)) 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 mivanbenne(_,Y) and mivanbenne(_,Z)
        and Y<>Z and szamfo(Y,Db1) and szamfo(Z,Db2)
        and Db1<Db2.
legtobb(Y) if mivanbenne(_,Y) and not(nemlegtobb(Y)).
```

A teljes program

```
domains
        vitaminnev=string.
        termeknev=string.
        db=integer.
```

```
predicates
        vitamin(vitaminnev).  /*ilyen vitaminok vannak*/
        mivanbenne(termeknev,vitaminnev).  /*melyik termek melyik vitamint
                                                tartalmazza*/
        kiiras.  /*a termekek es a vitaminok kiiratasa*/
        egyfele(termeknev).  /*az a) kerdeshez*/
        tobbfele(termeknev).
        osszes(termeknev).  /*a b) kerdeshez*/
        egyetnem(termeknev).
        legtobb(vitaminnev).  /*a c) kerdeshez*/
        szamol(vitaminnev,db).
        szamfo(vitaminnev,db).
        nemlegtobb(vitaminnev).
database
        marvolt(termeknev).
clauses
/*milyen vitaminok vannak*/
        vitamin(a).
        vitamin(b).
        vitamin(c).
/*a gyartott termekek milyen vitaminokat tartalmaznak*/
        mivanbenne(termek1,a).
        mivanbenne(termek2,a).
        mivanbenne(termek2,b).
        mivanbenne(termek2,c).
/*kiiratjuk az egyes termekeket es vitaminokat*/
        kiiras if mivanbenne(X,Y) and write(X,", ami ",Y, " vitamint tartalmaz") and nl and
                fail or !.
/*a) csak egyfele vitamint tartamaznak*/
        tobbfele(X) if vitamin(Y) and vitamin(Z) and
                mivanbenne(X,Y) and mivanbenne(X,Z) and Y<>Z.
        egyfele(X) if mivanbenne(X,_) and not(tobbfele(X)).
```

```
/*b) az osszes vitamint tartalmazzak*/
        egyetnem(X) if vitamin(Y) and not(mivanbenne(X,Y)).
        osszes(X) if mivanbenne(X,_) and not(egyetnem(X)).
/*c) az a vitamin, ami a legtobb termekben szerepel*/
/*ehhez minden vitaminnal meg kell szamolni, hogy hany termek tartalmazza*/
        szamol(Y,DB) if vitamin(Y) and mivanbenne(X,Y) and
                not(marvolt(X)) and assert(marvolt(X)) 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 mivanbenne(_,Y) and mivanbenne(_,Z)
                and Y<>Z and szamfo(Y,Db1) and szamfo(Z,Db2)
                and Db1<Db2.
        legtobb(Y) if mivanbenne(_,Y) and not(nemlegtobb(Y)).
goal
        makewindow(1,2,15," Vitamin ",0,0,25,80) and
        clearwindow and nl and nl and
        write("Egy gyogyszergyar az alabbi vitaminkeszitmenyeket gyartja: \n") and
        kiiras and nl and
        write("Melyek azok a termekek, amik csak egyfele vitamint tartalmaznak?\n") and
        readchar(W) and
        egyfele(X) and write("Csak egyfele vitamint tartalmaz a ",X,".") and nl and nl and
        write("Mely termekek tartalmazzak az osszes vitamint?\n") and
        readchar(Q) and
        osszes(Y) and write("Az osszes vitamint a ",Y," tartalmazza.") and nl and nl and
        write("Add meg a legtobb termekben levo vitamint!\n") and
        readchar(T) and
        legtobb(Z) and write("A legtobb termekben levo vitamin: ",Z," vitamin.").
```

2.9. Adás

A feladat

Törpicur szakértője a TV-műsoroknak, mivel egész nap a TV előtt ül. Okoska irigykedik rá, ezért próbára akarja tenni. Az alábbi kérdéseket teszi fel neki:

a) Mondj egy olyan adót, ami csak egyetlen műsortípust sugároz!

b) Adj meg egy műsortípust, amelyet az összes adó sugároz!

c) Melyik az az adó, amely minden másnál többféle adást sugároz?

Vajon mit válaszol Törpicur?

A feladat megoldása

Azt, hogy melyik adó milyen adást sugároz, így jelöljük:

```
adas(nev,musor)
```

Meghatározzuk, hogy milyen adókról van szó:

```
ado(nev)
```

Definiáljuk a voltmar adatbázist.

```
database
        voltmar(musor)
```

Kiíratjuk a képernyőre a TV-műsort.

```
kiiras if adas(X,Y) and write(X,"  ",Y) and nl and fail or !.
```

Az a) kérdés megválaszolásához előbb megadjuk, hogy mely csatornák sugároznak több műsortípust is. Ezután ennek tagadásával választ kapunk arra a kérdésre, hogy melyek sugároznak csak egyet.

```
tobbetsugaroz(A) if adas(A,B) and adas(A,C) and B<>C.
ado(A) if adas(A,_).
egyetsugaroz(A) if ado(A) and not(tobbetsugaroz(A)).
```

A b) feladat arra kíváncsi, hogy mely műsortípust sugározza az összes adó. Ehhez először megmondjuk, hogy mely adók nem sugároznak bizonyos műsortípusokat. Ennek tagadásával válaszoljuk meg a feladatot.

```
tipus(T) if adas(_,T).
vanolyanaminemsugarozza(T) if ado(A) and tipus(T) and not(adas(A,T)).
mindsugarozza(T) if tipus(T) and not(vanolyanaminemsugarozza(T)).
```

A c) feladathoz először minden adónál meg kell számolni, hogy hányféle műsort sugároz. Ezután megadjuk, hogy melyek azok az adók, amelyekre igaz, hogy van olyan adó, amely többféle műsort sugároz, mint azok. Ez a nemlegtobb szabály. Ennek tagadásával kapjuk meg a kérdésünkre a választ.

```
sugaroz(A,T) if adas(A,T).
sugarszam(A,DB) if sugaroz(A,T) and not(voltmar(T)) and
        asserta(voltmar(T)) and sugarszam(A,DB1) and DB=DB1+1 and ! or DB=0.
fosugarszam(A,DB) if retractall(voltmar(_)) and sugarszam(A,DB).
nemlegtobb(A) if sugaroz(A,_) and sugaroz(A2,_) and A<>A2 and
        fosugarszam(A,DB) and fosugarszam(A2,DB2) and DB2>DB.
legtobb(A) if ado(A) and not(nemlegtobb(A)).
```

A teljes program

```
domains
        nev,musor=string
        szam=integer
predicates
        adas(nev,musor)
        ado(nev)
        kiiras
```

```
        tobbetsugaroz(nev)     /*az a) kerdeshez*/
        egyetsugaroz(nev)
        tipus(musor)   /*a b) kerdeshez*/
        vanolyanaminemsugarozza(musor)
        mindsugarozza(musor)
        sugaroz(nev,musor)    /*a c) kerdeshez*/
        sugarszam(nev,szam)
        fosugarszam(nev,szam)
        nemlegtobb(nev)
        legtobb(nev)
database
        voltmar(musor)
clauses
/*az egyes adok ilyen adasokat sugaroznak*/
        adas(m1,hirado).
        adas(m1,mese).
        adas(m1,sport).
        adas(m2,hirado).
        adas(rtl_klub,hirado).
        adas(rtl_klub,szappanopera).
        adas(duna,film).
        adas(duna,sport).
        adas(duna,hirado).
        adas(duna,mese).
/* adatok kiiratasa*/
        kiiras if adas(X,Y) and write(X,"  ",Y) and nl and fail or !.
/* a) kerdes: megad egy olyan adot, ami csak egyetlen musortipust sugaroz*/
        tobbetsugaroz(A) if adas(A,B) and adas(A,C) and B<>C.
        ado(A) if adas(A,_).
        egyetsugaroz(A) if ado(A) and not(tobbetsugaroz(A)).
/* b) kerdes: megad egy musortipust, amit az osszes ado sugaroz*/
        tipus(T) if adas(_,T).
        vanolyanaminemsugarozza(T) if ado(A) and tipus(T) and not(adas(A,T)).
        mindsugarozza(T) if tipus(T) and not(vanolyanaminemsugarozza(T)).
```

```
/* c) kerdes: megad egy adot, ami minden masnal tobbfele adast sugaroz */
        sugaroz(A,T) if adas(A,T).
        sugarszam(A,DB) if sugaroz(A,T) and not(voltmar(T)) and asserta(voltmar(T)) and
                sugarszam(A,DB1) and DB=DB1+1 and ! or DB=0.
        fosugarszam(A,DB) if retractall(voltmar(_)) and sugarszam(A,DB).
        nemlegtobb(A) if sugaroz(A,_) and sugaroz(A2,_) and A<>A2 and
                fosugarszam(A,DB) and fosugarszam(A2,DB2) and DB2>DB.
        legtobb(A) if ado(A) and not(nemlegtobb(A)).
goal
        clearwindow and makewindow(1,14,15," Adas ",0,0,25,80) and
        write("Torpicur szakertoje a TV-musoroknak, mivel egesz nap\n") and
        write("a TV elott ul. Okoska irigykedik ra, ezert probara akarja\n") and
        write("tenni. Az alabbi kerdeseket teszi fel neki:\n") and
        write("a) Mondj egy olyan adot, ami csak egyetlen musortipust sugaroz!\n") and
        write("b) Adj meg egy musortipust, amelyet az osszes ado sugaroz!\n") and
        write("c) Melyik az az ado, amely minden masnal tobbfele adast sugaroz?\n") and
        write("Torpicur tudta, hogy az egyes csatornak az alabbi musortipusokat
                sugarozzak:\n") and
        kiiras and
        write("Igy az osszes kerdesre tudott valaszolni.  \n\n") and
        egyetsugaroz(A) and write("Csak egyetlen musort sugaroz: ", A) and nl and
        mindsugarozza(T) and write("Az osszes ado sugarozza: ",T) and nl and
        legtobb(X) and write("A legtobbfele musort sugarozza: ",X).
```

2.10. Megye

A feladat

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. Itt is a könnyebb feladatokkal kezdjük és fokozatosan haladunk a bonyolultabbak felé. Minden egyes feladat szövegét alaposan tanulmányozzuk és értelmezzük, majd Prolog-formájúvá alakítjuk.

3.1. Verseny

A feladat

Micimackó és barátai futóversenyt rendeztek.
A verseny után mindenkit megkérdeztek, melyik helyen végzett. A következő válaszokat adták:
Micimackó: Nem lettem sem első, sem utolsó.
Füles: Nem lettem első.
Zsebibaba: Első lettem.
Malacka: Én lettem az utolsó.
Állapítsd meg a helyezési sorrendjüket!

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!
Micimackóról tudjuk, hogy nem lett sem első, sem utolsó, tehát második vagy harmadik lett.

```
micimacko(Mi) if masodik(Mi) or harmadik(Mi).
```

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

```
fules(F) if masodik(F) or harmadik(F) or negyedik(F).
```

Zsebibaba nyerte meg a versenyt.

```
zsebibaba(Zs) if elso(Zs).
```

Malacka pedig utolsó lett.

```
malacka(Ma) if negyedik(Ma).
```

A teljes program

```
domains
        nev=string
        hely=string
predicates
        elso(hely)
        masodik(hely)
        harmadik(hely)
        negyedik(hely)
        micimacko(hely)
        fules(hely)
        zsebibaba(hely)
        malacka(hely)
        megoldas(nev,hely,nev,hely,nev,hely,nev,hely)
clauses
        elso(elso).
        masodik(masodik).
        harmadik(harmadik).
        negyedik(negyedik).
/*Micimacko nem lett sem elso, sem utolso, tehat masodik vagy harmadik lett.*/
        micimacko(Mi) if masodik(Mi) or harmadik(Mi).
/*Fules nem lett elso, azaz vagy masodik vagy harmadik vagy negyedik lett*/
```

```
        fules(F) if masodik(F) or harmadik(F) or negyedik(F).
/*Zsebibaba nyerte meg a versenyt*/
        zsebibaba(Zs) if elso(Zs).
/*Malacka pedig utolso lett*/
        malacka(Ma) if negyedik(Ma).
        megoldas("Micimacko",Mi,"Fules",F,"Zsebibaba",Zs,"Malacka",Ma) if
                micimacko(Mi) and
                fules(F) and zsebibaba(Zs) and malacka(Ma) and Mi<>F and Mi<>Zs and
                Mi<>Ma and F<>Zs and F<>Ma and Zs<>Ma.
goal
        clearwindow and makewindow(1,11,15,"Verseny",1,0,23,80) and nl and
        write(" Micimacko es baratai futoversenyt rendeztek. A verseny utan mindenkit\n")
                and
        write(" megkerdeztek, melyik helyen vegzett. \n") and
        write(" A kovetkezo valaszokat adtak: \n") and
        write(" Micimacko:   Nem lettem sem elso, sem utolso.\n ") and
        write("Fules:  Nem lettem elso.\n ") and
        write("Zsebibaba: Elso lettem.\n")  and
        write(" Malacka: En lettem az utolso.\n ") and
        write("Meg tudod-e allapitani a helyezesi sorrendjuket? \n\n") and readchar(Y) and
        megoldas("Micimacko",Mi,"Fules",F,"Zsebibaba",Zs,"Malacka",Ma) and
        write("  Micimacko: ",Mi,"   Fules: ",F,"   Zsebibaba: ",Zs,"   Malacka: ",Ma) and
        nl and fail.
```

3.2. Vonatos

A feladat

A Kakukk Expressz személyzete a kalauzból, a gépészből és a masinisztából áll. Őket Kissnek, Nagynak és Kovácsnak hívják, azonban nem feltétlenül ebben a sorrendben. A vonat három utasat 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 személyzetről és az utasokról:

a) Dr. Kiss Egerben lakik.

b) Kollegája, Dr. Kovács Budapesten lakik.

c) A szegedi utas névrokona a kalauz.

d) Kovács úr nem gépész.

Kérdésem: mi a masiniszta neve?

A feladat megoldása

Először meg kell adnunk, hogy kik tartoznak az expressz személyzetéhez,

```
szemelyzet(kiss).
szemelyzet(nagy).
szemelyzet(kovacs).
```

és kik az utasokhoz.

```
utas(dr_kiss).
utas(dr_nagy).
utas(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 Egerben lakik.

```
lakik(dr_kiss,eger).
```

A második azt mondja ki, hogy kollégája, Dr. Kovács Budapesten lakik.

```
lakik(dr_kovacs,budapest).
```

Tudjuk még, hogy a harmadik utas sem Egerben, sem Budapesten nem lakhat, tehát Szegeden lakik.

```
lakik(X,szeged) if utas(X) and not(lakik(X,eger)) and not(lakik(X,budapest)).
```

A harmadik állítás szerint a szegedi utas névrokona a kalauz.

```
foglalkozas(X,kalauz) if nevrokon(X,Y) and lakik(Y,szeged).
```

A negyedik kijelentés szerint Kovács úr nem gépész. Tehát vagy masiniszta vagy kalauz.

```
foglalkozas(kovacs,masiniszta) if not(foglalkozas(kovacs,kalauz)).
```

Tudjuk még, hogy a személyzet közül aki nem kalauz és nem masiniszta, annak feltétlenül gépésznek kell lennie.

```
foglalkozas(X,gepesz) if szemelyzet(X) and not(foglalkozas(X,kalauz))
        and not(foglalkozas(X,masiniszta)).
```

A teljes program

```
domains
        nev,tevekenyseg,varos=symbol.
predicates
        szemelyzet(nev).
        utas(nev).
        nevrokon(nev,nev).
        lakik(nev,varos).
        foglalkozas(nev,tevekenyseg).
clauses
/*a szemelyzet:*/
```

```
        szemelyzet(kiss).
        szemelyzet(nagy).
        szemelyzet(kovacs).
/*az utasok:*/
        utas(dr_kiss).
        utas(dr_nagy).
        utas(dr_kovacs).
/*ki kinek a nevrokona:*/
        nevrokon(kiss,dr_kiss).
        nevrokon(nagy,dr_nagy).
        nevrokon(kovacs,dr_kovacs).
/*a) Dr. Kiss Egerben lakik.*/
        lakik(dr_kiss,eger).
/*b) Kollegaja, Dr. Kovacs Budapesten lakik.*/
        lakik(dr_kovacs,budapest).
/*Tudjuk, hogy a harmadik utas sem Egerben sem Budapesten nem lakhat, tehat Szegeden
lakik.*/
        lakik(X,szeged) if utas(X) and not(lakik(X,eger)) and
                not(lakik(X,budapest)).
/*c) A szegedi utas nevrokona a kalauz.*/
        foglalkozas(X,kalauz) if nevrokon(X,Y) and lakik(Y,szeged).
/*d) Kovacs ur nem gepesz.*/
        foglalkozas(kovacs,masiniszta) if not(foglalkozas(kovacs,kalauz)).
/*Tudjuk meg, hogy a szemelyzet kozul aki nem kalauz es nem masiniszta,
annak feltetlenul gepesznek kell lennie.*/
        foglalkozas(X,gepesz) if szemelyzet(X) and not(foglalkozas(X,kalauz)) and
                not(foglalkozas(X,masiniszta)).
goal
        clearwindow and makewindow(1,11,15,"Vonatos",1,0,23,80) and
        write("A Kakukk Expressz szemelyzete a kalauzbol, a gepeszbol\n") and
        write("es a masinisztabol all. \n") and
        write("Oket  Kissnek, Nagynak es Kovacsnak hivjak,\n") and
        write("azonban nem feltetlenul ebben a sorrendben.\n") and
        write("A vonat harom utasat 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 szemelyzetrol es az utasokrol:\n") and
        write("a) Dr. Kiss Egerben lakik.\n") and
        write("b) Kollegaja, Dr. Kovacs Budapesten lakik.\n") and
        write("c) A szegedi utas nevrokona a kalauz.\n") and
        write("d) Kovacs ur nem gepesz.\n") and
        write("Kerdesem: mi a masiniszta neve?\n") and
        readchar(W) and foglalkozas(X,masiniszta) and nl and write(X).
```

3.3. Idősebb

A feladat

Két testvért, egy fiút és egy lányt megkérdeztek, hogy ki 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("Ket testvert, egy fiut es egy lanyt, megkerdeztek, hogy ki 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. Foglalk

A feladat

Négy ember 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(" Negy ember 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. Alice

A feladat

Alice a Feledékenység erdejében találkozik az Oroszlánnal, aki hétfőn, kedden és szerdán hazudik, a többi napon igazat mond. Az Oroszlán mond két állítást, ezek alapján Alicenak meg kell mondania, hogy milyen nap van ma. Segíts neki!

– Tegnap hazudtam.
– Holnap megint hazudni fogok.

Milyen nap van ma?

A feladat megoldása

Mivel az Oroszlán 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 Oroszlán igazmondó, vagy hazudós napján nyilatkozott, ezért szét kell választanunk az egyes eseteket.

Ha az oroszlán 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
        allat=string
predicates
        igazmondo(allat,nap)
        tegnap(nap,nap)
        holnap(nap,nap)
        allit1(nap)
        allit2(nap)
        megoldas(nap)
clauses
/*az oroszlan 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 oroszlan, 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("    Alice a Feledekenyseg erdejeben talalkozott az Oroszlannal,")and nl and
        write("    aki minden hetfon, kedden es szerdan hazudik,") and nl and
        write("    a tobbi napon igazat mond.") and nl and
        write("    Az Oroszlan mond ket allitast, ezek alapjan Alicenak meg kell
        mondania,") and nl and
        write("    hogy milyen nap van ma. Segits neki!") and nl 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. Vampir1

A feladat

Erdélyben emberek és vámpírok laknak. A vámpírok mindig hazudnak, az emberek mindig igazat mondanak. Az emberek és a vámpírok fele őrült –minden igaz állítást hamisnak és minden hamis állítást igaznak hisznek. A lakosság másik fele teljesen egészséges, pontosan tudja, melyik állítás igaz, melyik hamis. Egészséges emberek és őrült vámpírok egyaránt csak igazat mondanak, őrült emberek és egészséges vámpírok pedig mindig hazudnak. Az alábbi eset szereplői a Karloff testvérek, Michael és Peter. Ezt mondják:
Michael Karloff: „Én vámpír vagyok."
Peter Karloff: „ Én ember vagyok."
Michael Karloff: „A testvéremnek és nekem egyforma az elmeállapotunk."
Melyikük a vámpír?

A feladat megoldása

Craig felügyelőnek nem jelentett gondot a rejtély megoldása, mi is könnyűszerrel megbirkózunk vele.

Azt, hogy mi az illető foglalkozása, azaz vámpír-e vagy ember, így jelöljük:

```
foglalkozas(ki)
```

Hogy milyen az illető elmeállapota, vagyis normális-e vagy őrült, így írjuk le:

```
elme(ki)
```

A feladatból tudjuk, hogy Erdélyben emberek és vámpírok élnek, akiknek valamilyen az elmeállapotuk (ezt jelen esetben K-val és M-mel jelöljük, mert pontosan még nem tudjuk). Az állításoknál a zárójelen belüli rész első tagja Michael elmeállapotát jelenti, ezután a foglalkozása következik. A harmadik tag Peter elmeállapotát jelöli, ezt az ő foglalkozása követi.

```
allit0(K,vampir,M,ember) if elme(K) and elme(M) .
allit0(K,ember,M,vampir) if elme(K) and elme(M) .
```

Ha Michael igazat mond, akkor ő vámpír, mégpedig őrült, Peternek pedig vele megegyező az elmeállapota, vagyis ő is őrült.

```
allitM(orult,vampir,orult,N) if  foglalkozas(N).
```

Ha Michael hazudik, akkor csakis ember lehet, mégpedig őrült, akkor pedig nem egyezik az elmeállapotuk, tehát Peter normális.

```
allitM(orult,ember,normalis,N) if foglalkozas(N).
```

Ha Peter igazat mond, akkor ő ember, mégpedig normális.

```
allitP(K,L,normalis,ember) if  elme(K) and foglalkozas(L).
```

Ha Peter hazudik, akkor ő vámpír, aki ráadásul normális.

```
allitP(K,L,normalis,vampir) if elme(K) and foglalkozas(L).
```

Akkor kapunk megoldást, ha valamennyi állítást figyelembe vesszük.

```
megoldas(K,L,M,N) if allit0(K,L,M,N) and allitM(K,L,M,N) and allitP(K,L,M,N).
```

Ezek után már könnyen meg tudjuk mondani, hogy a testvérpár mely tagja a vámpír.

```
vampir("Michael") if  megoldas(K,vampir,M,N) and elme(K)
        and elme(M) and foglalkozas(N).
vampir("Peter") if  megoldas(K,L,M,vampir) and elme(K)
        and foglalkozas(L) and elme(M).
```

A teljes program

```
domains
        ki=string
        kik=string
predicates
        foglalkozas(ki)          /* foglalkozasa: vampir vagy ember */
```

```
        elme(ki)        /* elmeje: normalis vagy orult */
        allit0(ki,ki,ki,ki)
        allit0(ki,ki,ki,ki)
        allitP(ki,ki,ki,ki)
        allitM(ki,ki,ki,ki)
        megoldas(ki,ki,ki,ki)
        vampir(kik)
clauses
        foglalkozas(vampir).
        foglalkozas(ember).
        elme(normalis).
        elme(orult).
/*Erdelyben vampirok es emberek laknak.*/
        allit0(K,vampir,M,ember) if elme(K) and elme(M) .
        allit0(K,ember,M,vampir) if elme(K) and elme(M) .
/*ha Michael igazat mond, akkor o vampir, megpedig orult,
Peternek pedig vele megegyezo az elmeallapota, vagyis o is orult*/
        allitM(orult,vampir,orult,N) if  foglalkozas(N).
/*ha Michael hazudik, akkor csakis ember lehet, megpedig orult,
akkor pedig nem egyezik az elmeallapotuk, tehat Peter normalis*/
        allitM(orult,ember,normalis,N) if foglalkozas(N).
/*ha Peter igazat mond, akkor o ember, megpedig normalis*/
        allitP(K,L,normalis,ember) if  elme(K) and foglalkozas(L).
/*ha Peter hazudik, akkor o vampir, aki raadasul normalis*/
        allitP(K,L,normalis,vampir) if elme(K) and foglalkozas(L).
/*akkor kapunk megoldast, ha valamennyi allitast figyelembe vesszuk*/
        megoldas(K,L,M,N) if allit0(K,L,M,N) and allitM(K,L,M,N) and allitP(K,L,M,N).
/*Ki a vampir?*/
        vampir("Michael") if  megoldas(K,vampir,M,N) and elme(K)
                and elme(M) and foglalkozas(N).
        vampir("Peter") if  megoldas(K,L,M,vampir) and elme(K)
                and foglalkozas(L) and elme(M).
goal
        clearwindow and makewindow(1,12,75,"Vampir",1,0,23,80) and nl and
```

```
        write("Erdelyben emberek es vampirok laknak. \n") and
        write("A vampirok mindig hazudnak, az emberek mindig igazat mondanak. \n")
        and write("Az emberek es a vampirok fele is orult -\n") and
        write("minden igaz allitast hamisnak es minden hamis allitast igaznak hisznek. \n")
        and write("A lakossag masik fele teljesen egeszseges, pontosan tudja, melyik allitas
                igaz, melyik hamis. \n") and
        write("Egeszseges emberek es orult vampirok egyarant csak igazat mondanak, \n")
        and write("orult emberek es egeszseges vampirok pedig mindig hazudnak. \n") and
        write("Az alabbi eset szereploi a Karloff testverek, Michael es Peter. Ezt
                mondjak:\n") and
        write("Michael Karloff: En vampir vagyok.\n") and
        write("Peter Karloff: En ember vagyok.\n") and
        write("Michael Karloff: A testveremnek es nekem egyforma az elmeallapotunk.\n")
        and write("Melyikuk a vampir?\n\n") and readchar(Y) and
        megoldas(K,L,M,N) and vampir(X) and write(X," a vampir!") and nl and fail.
```

3.7. Vampir2

A feladat

Ez a történet is Erdélyben játszódik, ahol emberek és vámpírok laknak. A vámpírok mindig hazudnak, az emberek mindig igazat mondanak. Az emberek és a vámpírok fele őrült – minden igaz állítást hamisnak és minden hamis állítást igaznak hisznek. A lakosság másik fele teljesen egészséges, pontosan tudja, melyik állítás igaz, melyik hamis. Egészséges emberek és őrült vámpírok egyaránt csak igazat mondanak, őrült emberek és egészséges vámpírok pedig mindig hazudnak. Az alábbi eset szereplői egy házaspár, Sylvan és Sylvia Nitrate: vagy mindketten emberek, vagy mindketten vámpírok. A szereplők elmeállapotáról semmit nem tudunk.
Craig felügyelő: (Mrs. Nitrate-nek) "Mondjon valamit magukról!"
Sylvia: "A férjem ember."
Sylvan: "A feleségem vámpír."
Sylvia: "Egyikünk egészséges, a másikunk nem."
Emberek vagy vámpírok a szereplők?

A feladat megoldása

Az elmeállapot és a foglalkozás meghatározása ugyanúgy történik, mint az előző feladatnál. Ezúttal is háromféle állítást különböztetünk meg: amit eleve tudunk Erdélyről, amit a férfi és amit a nő mond. A jelölés itt is megegyezik az előző feladatéval:

```
allit0(K,ember,M,ember) if elme(K) and elme(M).
allit0(K,vampir,M,vampir) if elme(K) and elme(M).
```

Ha Sylvia igazat mond, akkor vagy normális ember vagy őrült vámpír és a férje ember (vagy őrült vagy normális).

```
allitNo(normalis,ember,orult,ember).
allitNo(orult,vampir,normalis,ember).
```

Ha a nő hazudik, akkor ez azt jelenti, hogy vagy normális vámpír vagy őrült ember és a férfi vámpír.

```
allitNo(normalis,vampir,normalis,vampir).
allitNo(orult,ember,orult,vampir).
```

Ha a férfi hazudik, akkor ő vagy őrült ember vagy normális vámpír és a nő ember.

```
allitFerfi(K,ember,orult,ember) if elme(K).
allitFerfi(K,ember,normalis,vampir) if elme(K).
```

Ha igazat mond, akkor vagy normális ember vagy őrült vámpír és a nő vámpír.

```
allitFerfi(K,vampir,normalis,ember) if  elme(K) .
allitFerfi(K,vampir,orult,vampir) if  elme(K) .
```

A megoldást megkapjuk, ha a háromféle állítást figyelembe vesszük.

```
megoldas(K,L,M,N) if allit0(K,L,M,N) and allitNo(K,L,M,N)
        and allitFerfi(K,L,M,N).
```

A teljes program

```
domains
        ki=string
        kik=string
predicates
        foglalkozas(ki)         /* foglalkozasa: vampir vagy ember */
        elme(ki)                /* elmeje: normalis vagy orult */
        allit0(ki,ki,ki,ki)
        allitNo(ki,ki,ki,ki)
        allitFerfi(ki,ki,ki,ki)
        megoldas(ki,ki,ki,ki)
        rejtely(kik)
clauses
        foglalkozas(vampir).
        foglalkozas(ember).
```

```
        elme(normalis).
        elme(orult).
/*Erdelyben emberek es vampirok elnek*/
        allit0(K,ember,M,ember) if elme(K) and elme(M).
        allit0(K,vampir,M,vampir) if elme(K) and elme(M).
/*ha a no igazat mond, akkor vagy normalis ember
vagy orult vampir es a ferj ember (vagy orult vagy normalis)*/
        allitNo(normalis,ember,orult,ember).
        allitNo(orult,vampir,normalis,ember).
/*ha a no hazudik, akkor vagy normalis vampir vagy
orult ember es a ferj vampir (vagy orult vagy normalis)*/
        allitNo(normalis,vampir,normalis,vampir).
        allitNo(orult,ember,orult,vampir).
/*ha a ferj igazat mond, akkor vagy normalis ember
vagy orult vampir es a no vampir */
        allitFerfi(K,vampir,normalis,ember) if elme(K) .
        allitFerfi(K,vampir,orult,vampir) if elme(K) .
/*ha a ferj hazudik, akkor vagy orult ember vagy
normalis vampir es a no ember*/
        allitFerfi(K,ember,orult,ember) if elme(K).
        allitFerfi(K,ember,normalis,vampir) if elme(K).
/*a megoldast megkapjuk, ha a kiindulo allitasokat,
a ferfi es a no kijelenteseit is figyelembe vesszuk*/
        megoldas(K,L,M,N) if allit0(K,L,M,N) and allitNo(K,L,M,N)
                and allitFerfi(K,L,M,N).
        rejtely("Emberek") if megoldas(K,ember,M,ember) and elme(K) and elme(M).
        rejtely("Vampirok") if megoldas(K,vampir,M,vampir)and elme(K) and elme(M).
goal
        clearwindow and makewindow(1,12,75,"Vampir",1,0,23,80) and nl and
        write("Erdelyben emberek es vampirok laknak. \n") and
        write("A vampirok mindig hazudnak, az emberek mindig igazat mondanak. \n")
        and write("Az emberek es a vampirok fele is orult -\n") and
        write("minden igaz allitast hamisnak es minden hamis allitast igaznak hisznek. \n")
        and write("A lakossag masik fele teljesen egeszseges, pontosan tudja, melyik allitas
```

```
        igaz, melyik hamis. \n") and
    write("Egeszseges emberek es orult vampirok egyarant csak igazat mondanak, \n")
    and
    write("orult emberek es egeszseges vampirok pedig mindig hazudnak. \n") and
    write("Az alabbi eset szereploi egy hazaspar, Sylvan és Sylvia Nitrate: \n") and
    write("vagy mindketten emberek, vagy mindketten vampirok. \n") and
    write("A szereplok elmeallapotarol semmit nem tudunk. \n") and
    write("Craig felugyelo (Mrs. Nitrate-nek): Mondjon valamit magukrol!\n") and
    write("Sylvia: A ferjem ember.\n") and
    write("Sylvan: A felesegem vampir.\n") and
    write("Sylvia: Egyikunk egeszseges, a masikunk nem.\n") and
    write("Emberek vagy vampirok?\n\n") and readchar(Y) and
    megoldas(K,L,M,N) and rejtely(X) and write(X) and nl and fail.
```

3.8. Hölgy

A feladat

Egy bizonyos ország királya próbára tette a rabjait. A dolgot azonban nem bízta a véletlenre, feliratokat tett a szobák ajtajára és minden esetben elárult bizonyos tényeket a raboknak a feliratokkal kapcsolatban. Ha egy rab elég okos volt és logikusan gondolkodott, megmenekülhetett a haláltól és ráadásul elnyerhette egy szépséges hölgy kezét.
Az első napon az egyik rabnak elmagyarázta a király, hogy a két szoba mindegyikében vagy egy hölgy van, vagy egy tigris található. Az is lehet, hogy mindkettőben tigris van, az is, hogy mindkettőben hölgy, és persze az is, hogy egy egyikben hölgy, a másikban tigris.
A király rámutatott az ajtón levő feliratokra:
1. Ebben a szobában hölgy van, a másikban pedig tigris.
2. Egyik szobában hölgy van, a másikban pedig tigris.
Az egyik felirat igaz, a másik hamis.
Melyik ajtót nyitnád ki, ha te volnál a rab?

A feladat megoldása

Azt, hogy a szobában tigris található a következőképp jelöljük:

```
t(tigris).
```

Azt pedig, hogy a szobában hölgy van, így:

```
h(holgy).
```

A király megmondta, hogy a két szoba mindegyikében vagy hölgy van vagy tigris. Ez jelentheti azt, hogy mindkettőben hölgy van,

```
allit0(A,B) if h(A) and h(B).
```

vagy mindkettőben tigris,

```
allit0(A,B) if t(A) and t(B).
```

vagy az elsőben hölgy és a másodikban tigris,

```
allit0(A,B) if h(A) and t(B).
```

vagy pedig az elsőben tigris és a másodikban hölgy üldögél.

```
allit0(A,B) if t(A) and h(B).
```

Most sorra kell vennünk az egyes eseteket, mivel mindkét állítás lehet igaz és hamis is.
Ha az első állítás igaz, akkor az első szobában hölgy van, a másikban tigris.

```
allit1i(A,B) if h(A) and t(B).
```

Ha az első állítás hamis, tehát nem igaz, hogy az első szobában hölgy, a másokban tigris van, akkor ezt azt jelenti, hogy vagy mindkét szobában tigris van,

```
allit1h(A,B) if t(A) and t(B).
```

vagy az elsőben tigris, a másikban hölgy van,

```
allit1h(A,B) if t(A) and h(B).
```

vagy mindkettőben hölgy található:

```
allit1h(A,B) if h(A) and h(B).
```

Ha a második állítás igaz, akkor vagy az első szobában van a hölgy és a másikban tigris található,

```
allit2i(A,B) if h(A) and t(B).
```

vagy az elsőben tigris és a másikban hölgy van:

```
allit2i(A,B) if t(A) and h(B).
```

Ha a második állítás hamis, akkor nem teljesül az, hogy az egyik szobában hölgy, a másikban tigris van, vagyis vagy mindkettőben hölgy,

```
allit2h(A,B) if h(A) and h(B).
```

vagy mindkettőben tigris található:

```
allit2h(A,B) if t(A) and t(B).
```

Azt is tudjuk, hogy az egyik felirat igaz, a másik hamis. Ezt is figyelembe kell vennünk a megoldás keresésénél.

```
megoldas(A,B) if allit0(A,B) and allit1i(A,B) and allit2h(A,B).
megoldas(A,B) if allit0(A,B) and allit1h(A,B) and allit2i(A,B).
```

A teljes program

```
domains
        ki=string
predicates
        t(ki)
        h(ki)
        allit0(ki,ki)
        allit1i(ki,ki)
        allit2i(ki,ki)
        megoldas(ki,ki)
        allit1h(ki,ki)
        allit2h(ki,ki)
clauses
        t(tigris).      /*a szobaban tigris van*/
        h(holgy).       /*a szobaban holgy van*/
/*a ket szoba mindegyikeben vagy holgy van vagy tigris*/
        allit0(A,B) if h(A) and h(B).  /*vagy mindkettoben holgy*/
        allit0(A,B) if t(A) and t(B).  /*vagy mindkettoben tigris*/
        allit0(A,B) if h(A) and t(B).  /*vagy az elsoben holgy es a masodikban tigris*/
        allit0(A,B) if t(A) and h(B).  /*vagy az elsoben tigris es a masodikban holgy*/
        allit1i(A,B) if h(A) and t(B). /*ha az elso allitas igaz,az elso szobaban holgy van,
                                        a masodikban tigris*/
        allit1h(A,B) if t(A) and t(B).      /*ha az elso allitas hamis, akkor vagy mindket
                                            szobaban tigris van*/
```

```
        allit1h(A,B) if t(A) and h(B). /        *vagy az elsoben tigris, a masikban holgy*/
        allit1h(A,B) if h(A) and h(B).          /*vagy mindkettoben holgy*/
        allit2i(A,B) if h(A) and t(B).          /*ha a masodik allitas igaz, akkor vagy az elso
                                                szobaban van holgy es a masodikban tigris*/
        allit2i(A,B) if t(A) and h(B).          /*vagy az elsoben tigris es a masikban holgy*/
        allit2h(A,B) if h(A) and h(B).          /*ha hamis a masodik allitas, az azt jelenti,
                                                hogy vagy mindkettoben holgy*/
        allit2h(A,B) if t(A) and t(B).          /*vagy mindkettoben tigris van*/
        megoldas(A,B) if allit0(A,B) and allit1i(A,B) and allit2h(A,B).     /*az egyik felirat
                                                igaz, a masik hamis*/
        megoldas(A,B) if allit0(A,B) and allit1h(A,B) and allit2i(A,B).
goal
        clearwindow and makewindow(1,7,2,"A holgy es a tigris",0,0,24,80) and nl and
        write("Egy bizonyos orszag kiralya probara tette a rabjait.\n") and
        write("A dolgot azonban nem bizta a veletlenre, feliratokat tett a szobak ajtajara es
                minden esetben elarult bizonyos tenyeket a raboknak\n") and
        write("a feliratokkal kapcsolatban.\n") and
        write("Ha egy rab eleg okos volt es logikusan gondolkodott,\n") and
        write("megmenekulhetett a halaltol es raadasul elnyerhette egy szepseges holgy
                kezet.\n") and
        write("Az elso napon az egyik rabnak elmagyarazta a kiraly, \n") and
        write("hogy a ket szoba mindegyikeben vagy egy holgy van, vagy egy tigris
                talalhato.\n") and
        write("Az is lehet, hogy mindkettoben tigris van, az is, hogy mindkettoben holgy,
                es persze az is, hogy egy egyikben holgy, a masikban tigris.\n") and
        write("A kiraly ramutatott az ajton levo feliratokra:\n") and
        write("1. Ebben a szobaban holgy van, a masikban pedig tigris. \n") and
        write("2. Egyik szobaban holgy van, a masikban pedig tigris.\n") and readchar(W)
        and megoldas(A,B) and
        write("\n\n Az elso szobaban ",A," van,\n a masodikban pedig ",B,".").
```

3.9. Ali Baba

A feladat

Ali Baba híres negyven rablójának egyike betört Abdul üzletébe és ellopott néhány gyémántot. Szerencsére az összes gyémánt megkerült és kiderült, hogy vagy Abu, vagy Ibn, vagy pedig Haszib volt. A tárgyaláson a következőket állították:
Abu: Nem én követtem el a rablást.
Ibn: Nem Haszib volt.
Haszib: De, én voltam.
Később közülük ketten bevallották, hogy hazudtak. Ki volt a tettes?

A feladat megoldása

A rablók 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 Abu 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 Abu 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 Ibn igazat mond, akkor vallomása szerint Haszib ártatlan.

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

Ha Ibn hazudik, akkor ez azt jelenti, hogy Haszib bűnös.

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

Amennyiben Haszib 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)  /* Abu allitasa */
        allitI(ki,ki,ki,ki,ki,ki)  /* Ibn allitasa */
        allitH(ki,ki,ki,ki,ki,ki)  /* Haszib 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).
/* Abu 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).
/*Ibn 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).
/*Haszib 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,"Ali Baba",0,0,24,80) and clearwindow and nl and
        write("Ali Baba hires negyven rablojanak egyike betort Abdul uzletebe es \n") and
        write("ellopott nehany gyemantot. Szerencsere az osszes gyemant megkerult es \n")
        and write("kiderult, hogy vagy Abu, vagy Ibn, vagy pedig Haszib volt. \n") and
        write("A targyalason a kovetkezoket allitottak: \n") and
        write("Abu: Nem en kovettem el a rablast. \n") and
        write("Ibn: Nem Haszib volt. \n") and
        write("Haszib: 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 megold s: \n\n") and readchar(Y) and
megoldas(A,P,I,Q,H,R) and
write("Abu ",A," es ",P,".\n") and
write("Ibn ",I," es ",Q,".\n") and
write("Haszib ",H," es ",R,".\n") and nl and fail or nl.
```

3.10. Farkas

A feladat

Egy erdőben lovagok és lókötők élnek. A lovagok mindig igazat mondanak, a lókötők mindig hazudnak. Közülük néhányan farkasemberek, akik éjszakánként emberevő farkassá változnak. Három ember (A,B,C) mond egy-egy állítást, ezek alapján határozd meg, hogy ki kicsoda!
A: Legalább az egyikünk lovag.
B: 'C' a farkasember.

A feladat megoldása

Tudjuk, hogy az erdőben farkasemberek és emberek élnek:

```
f(farkasember).
f(ember).
```

akik egyben lovagok vagy lókötők:

```
l(lovag).
l(lokoto).
```

A, B és C közül pontosan egy farkasember, aki egyben lovag is. Tehát vagy A a farkasember, aki egyben lovag is, ekkor B és C emberek, akik vagy lovagok vagy lókötők (M és N),

```
allit0(farkasember,lovag,ember,M,ember,N) if l(M) and l(N).
```

vagy B a farkasember, aki egyben lovag is, a többiek emberek, vagy lovagok vagy lókötők (L és N),

```
allit0(ember,L,farkasember,lovag,ember,N) if l(L) and l(N).
```

a harmadik esetben C a farkasember-lovag, A és B emberek, vagy lovagok vagy lókötők (L és M):

```
allit0(ember,L,ember,M,farkasember,lovag) if l(L) and l(M).
```

Vegyük sorra az állításokat:

Ha A igazat mond, akkor ő lovag, mivel a lovagok mondanak mindig igazat. Az állítás szerint legalább egyikük lókötő. A semmiképp sem lehet lókötő, mivel ő biztos, hogy lovag. F, G, H-val jelöljük, hogy ember-e vagy farkasember az illető.

Ekkor lehet, hogy B és C egyaránt lókötő.

```
allitA(F,lovag,G,lokoto,H,lokoto) if f(F) and f(G) and f(H).
```

Lehet, hogy csak C a lókötő.

```
allitA(F,lovag,G,lovag,H,lokoto) if f(F) and f(G) and f(H).
```

Lehet, hogy csak B lókötő.

```
allitA(F,lovag,G,lokoto,H,lovag) if f(F) and f(G) and f(H).
```

Ha A hazudik, akkor ő lókötő. Az állítása viszont nem teljesül, vagyis nem igaz, hogy legalább egyikük lókötő. Azaz egyikük sem lókötő. Ez ellentmond annak, hogy A lókötő, ezért A nem hazudhat.

Ha B igazat mond, akkor ő nyilván lovag és akkor C farkasember, aki a feltételek szerint lovag is.

```
allitB(F,L,G,lovag,farkasember,lovag) if l(L) and f(F) and f(G).
```

Ha B nem mond igazat, akkor ő lókötő, mert a lókötők hazudnak, C pedig nem farkasember, azaz C ember.

```
allitB(F,L,G,lokoto,ember,N) if l(L) and l(L) and l(N) and f(F) and f(G).
```

A megoldást akkor kapjuk meg, ha a háromféle állítás teljesül.

```
megoldas(F,L,G,M,H,N) if allit0(F,L,G,M,H,N) and allitA(F,L,G,M,H,N)
        and allitB(F,L,G,M,H,N).
```

A teljes program

```
domains
        ki=string
predicates
        f(ki)  /*farkasember vagy ember*/
        l(ki)  /*lovag vagy lokoto*/
        allitO(ki,ki,ki,ki,ki,ki) /* pontosan egy farkasember, aki egyben lovag is*/
        allitA(ki,ki,ki,ki,ki,ki) /* 'A' allitasa */
        allitB(ki,ki,ki,ki,ki,ki) /* 'B' allitasa */
        megoldas(ki,ki,ki,ki,ki,ki) /* a megoldas */
clauses
        f(farkasember).
        f(ember).
        l(lovag).
        l(lokoto).
/* vagy az A vagy a B vagy a C farkasember */
        allitO(farkasember,lovag,ember,M,ember,N) if l(M) and l(N). /* A a farkasember,
                aki egyben lovag is, B es C emberek, M,N - lovag vagy lokoto */
        allitO(ember,L,farkasember,lovag,ember,N) if l(L) and l(N). /* B a farkasemben,
                aki egyben lovag is, a tobbiek emberek, L,N - lovag vagy lokoto */
        allitO(ember,L,ember,M,farkasember,lovag) if l(L) and l(M). /*C a farkasember,
                o lovag is, a tobbiek emberek, L,M - lovag vagy lokoto.*/
/* ha A igazat mond - akkor o lovag */
/* legalabb egyikuk lokoto - mivel A lovag, o nem lehet lokoto*/
        allitA(F,lovag,G,lokoto,H,lokoto) if f(F) and f(G) and f(H). /*vagy B es C lokoto*/
                /* F,H,G - ember vagy farkasember */
        allitA(F,lovag,G,lovag,H,lokoto) if f(F) and f(G) and f(H). /*vagy csak C lokoto*/
        allitA(F,lovag,G,lokoto,H,lovag) if f(F) and f(G) and f(H). /*vagy csak B lokoto*/
/* ha A hazudik - akkor o lokoto */
/*Az allitasa viszont nem teljesul, vagyis nem igaz, hogy legalabb egyikuk lokoto.
Azaz egyikuk sem lokoto. Ez ellentmond annak, hogy A lokoto, ezert A nem hazudhat.*/
/* ha B igazat mond - akkor o lovag*/
/* C pedig farkasember, aki lovag is*/
```

```
        allitB(F,L,G,lovag,farkasember,lovag) if l(L) and f(F) and f(G).
/* ha B hazudik - akkor o lokoto es C ember*/
        allitB(F,L,G,lokoto,ember,N) if l(L) and l(L) and l(N) and f(F) and f(G).
/* a megoldas: ha allitO,allitA,allitB is teljesul */
        megoldas(F,L,G,M,H,N) if allitO(F,L,G,M,H,N)  and allitA(F,L,G,M,H,N)
                and allitB(F,L,G,M,H,N).
goal
        makewindow(1,7,75,"Farkasember",0,0,24,80) and clearwindow and nl and
        write(" Egy erdoben lovagok es lokotok elnek. A lovagok mindig igazat mondanak,
                \n") and
        write(" a lokotok mindig hazudnak. Kozuluk nehanyan farkasemberek, \n") and
        write(" akik ejszakankent emberevo farkassa valtoznak. \n\n") and
        write(" Harom ember (A, B es C) kozul ketto mond egy-egy allitast, ezek alapjan
                \n") and
        write(" hatarozd meg, hogy ki kicsoda! \n\n") and
        write(" Kozuluk pontosan egy a farkasember, aki egyben lovag is! \n") and
        write("   A mondja: Legalabb egyikunk lokoto. \n") and
        write("   B mondja: 'C' farkasember. \n\n") and
        write("\n\t\t A megold s: \n\n") and readchar(Y) and
        megoldas(F,L,G,M,H,N) and
        write("  A: ",F," ,s ",L,", B: ",G," ,s ",M) and
        write(", C: ",H," ,s ",N,".") and nl and fail or nl.
```

IRODALOM

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

Rozgonyi-Borus Ferenc (1997): RAM-ba zárt világ. Mozaik Kiadó, Szeged

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

Smullyan, Raymond (1997): A hölgy vagy a tigris? és egyéb logikai feladatok. Typotex, Budapest

Smullyan, Raymond (1998): Mi a címe ennek a könyvnek? Drakula rejtélye, és más logikai feladványok. Typotex, Budapest

Smullyan, Raymond (1999): Seherezádé rejtélye és más bámulatos fejtörők, régiek és újak. Typotex, Budapest

Smullyan, Raymond (2001): Alice Rejtvényországban. Carolli mesék nyolcvan év alatti gyermekeknek. Typotex, Budapest

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