DEBRECENI EGYETEM
INFORMATIKAI KAR
SZÁMÍTÓGÉPTUDOMÁNYI TANSZÉK

A mesterséges intelligencia feladatok megoldása Prologban

konzulens: Dr. Várterész Magda
egyetemi docens

szerző: Kőszegi Judit
programtervező informatikus

Debrecen
2008.

Tartalomjegyzék

Bevezetés

Kőszegi Judit vagyok, a Debreceni Egyetem programtervező informatikus (Bsc) hallgatója. Tanulmányaim során elvégzett tantárgyak közül az számítástudomány-orientált logika, és a mesterséges intelligencia fogtak meg legjobban. Úgy gondolom, ezek a tantárgyak közel állnak hozzám. 2007 nyarán egy egyhónapos nyári ösztöndíj keretében volt alkalmam a logika területén ismereteimet bővíteni, és a 2007/2008-as tanévben pedig demonstrátor lettem a számítógéptudományi tanszéken.

A mesterséges intelligencia gyakorlaton a feladatokat (megoldást kereső rendszerek, illetve kétszemélyes játékok) Java-ban valósítottuk meg. Mivel ezen feladatok állapottér-reprezentációját a matematikai logika nyelvén is meg lehet fogalmazni, kíváncsi lettem, hogy hogyan lehetne másképp, a logikai algoritmusok segítségével keresni ezen problémák megoldását. Témavezetőm bíztatására főként külföldi szakirodalom és egyetemi jegyzetek feldolgozásával megismerkedtem a Prolog nyelv elméletének és gyakorlatának alapjaival, amely egy deklaratív magasszintű programozási nyelv, a logikai paradigma, a matematikai logika fogalom- és eszközrendszerén épül fel.

Célom, hogy a Prolog segítségével tudjak alapvető megoldás-keresési stratégiákat megvalósítani, valamint ezek segítségével konkrét problémákat oldjak meg.

„A logika ... egy olyan tudomány, amely az összes tudományok számára általános érvénnyel bíró fogalmakat vizsgálja, és az ezeket kormányzó általános szabályokat állapít meg”

Alfred Tarski, (1901-1983); Introduction to logic and to the methodology of deductive sciences

1. Logikai alapok

A Prolog a matematikai logika fogalom- és eszközrendszerén alapuló logikai programozási paradigma mentén született meg, ezért szükséges, hogy elsőször vizsgáljuk meg a logika azon területeit, amely alapján a későbbiekben formalizálni tudjuk egy logikai programozási nyelv segítségével problémáinkat, valamint megérthetjük egy logikai program működését.

1.1. Bevezetés, alapfogalmak – a formula

A valós világgal kapcsolatos tudásunk kifejezésére kijelentő mondatokat használunk, mint például:

(i) Minden anya szereti a gyermekeit.

(ii) Anna anya, és Zsolt Anna gyermeke.

Az érvelés általánosan ismert szabályait felhasználva ismereteinkből következtetéseket is levonhatunk. Például ha tudjuk (i)-t és (ii)-t akkor következtethetünk arra, hogy:

(iii) Anna szereti Zsoltot.

Ha közelebbről megvizsgáljuk ezt a példát, kiderül, hogy (i) és (ii) egy családdal kapcsolatos kis világról szól, a családtagok közötti kapcsolatokat írja le – például „… egy anya” , „… gyermeke …-nak”, vagy a viszony: „… szereti …-t” – amely tulajdonságok és viszonyok vagy fennállnak vagy nem az egyes személyekre. A logikai programozás során ki szeretnénk fejezni az egyes individuumok közötti összes kapcsolatot, annak érdekében, hogy olyan következtetésekre juthassunk, mint az (iii).

Hogy a számítógéppel fel tudjuk dolgozni az olyan mondatokat, mint az (i)-(iii), szükségünk van egy formális nyelvre. Ami még ezen kívül fontos, hogy a helyes következtetés szabályait – amilyen az is, hogy hogyan kapjuk az (iii)-t (i)-ből és (ii)-ből – összegyűjtsük és körültekintően formalizáljuk. A matematikai logika eszközrendszere alkalmas lesz erre.

Az első fogalom, amivel foglalkozunk, a *logikai formula*, amelynek segítségével leírhatjuk (formalizálhatjuk) az (i)-(iii)-hez hasonló állításokat, azonosítjuk a vizsgált világ

individuumait (egyedeit) és a köztük lévő kapcsolatokat. Ezért ez kiindulópont a nyelv ábécéjére vonatkozóan. A logikai nyelvünk ábécéje a logikán kívüli szimbólumok mellett még néhány logikai jelet, illetve elválasztójeleket is tartalmaz.

A logikán kívüli szimbólumhalmazok:

- **változók** halmaza
- a **predikátumszimbólumok** halmaza (szimbólumok a kapcsolatok leírására, pl. $szereti, anya, gyermeke$);
- a **függvényszimbólumok** halmaza (az individuumok tartományára vonatkozó függvényeket reprezentálják);
- a **konstansszimbólumok** halmaza (szimbólumok az individuumok azonosítására, pl. a fenti példában a $zsolt$ szimbólumot használhatjuk a konkrét személy, Zsolt azonosítására).

Megjegyzés. A függvény – és predikátumszimbólumok rendelkeznek ú.n. aritással, amely az argumentumaik számát határozza meg. Egy n aritású f függvényszimbólum jele: f/n; egy m aritású p predikátumszimbólum jele pedig p/m lesz.

A matematikai logika leíró nyelvének egy $\mathcal{A}$ ábécéjét a fenti szimbólumhalmazok segítségével adhatjuk meg.

Logikai jelek:

- **logikai összekötőjelek**: $\neg(negáció),\ \wedge(konjukció),\ \vee(diszjunkció), \supset (implikáció)$.
- **kvantorok**: $\forall\ (univerzális), \exists\ (egzisztenciális)$.
- különböző fajtájú individuum**változók**

Elválasztójelek:

- zárójelek **(** és **)**
- vessző **,**

Ezzel a formális nyelvvel olyan állításokat is ki tudnunk fejezni, mint az (i), amely a leírt világ összes elemére vonatkozik Ezért vezettük be a nyelvünkbe az univerzális kvantor

szimbólumát és a változók ábécéjét. A változó egy olyan szimbólum, amely nem konkrét egyedre vonatkozik. Az (i)-(iii) mondatokat tehát a következőképpen tudjuk a nyelvben formalizálni:

(1) $\forall X \forall Y \Big(\big(anya(X) \wedge gyermeke(Y,X) \big) \supset szereti(X,Y) \Big)$

(2) $anya(anna) \wedge gyermeke(zsolt, anna)$

(3) $szereti(anna, zsolt)$

Az elsőrendű logika formális nyelvében az individuumokat lehetőségünk van megnevezni, ugyanakkor alkalmanként nem megnevezzük, hanem körülírjuk őket. Ezeket a sztringeket *term*eknek nevezünk, és a szintaktikájukat a következőkképpen definiáljuk:

Definíció. (Termek) A $\mathcal{A}$ ábécé feletti $\mathcal{T}$ termek halmaza az a legszűkebb halmaz, melyre:

- $c \in \mathcal{T}$ bármely $c \in \mathcal{A}$ konstansszimbólum esetén;
- $X \in \mathcal{T}$ bármely $X \in \mathcal{A}$ változó esetén;
- és ha az f/n függvényszimbólum $\mathcal{A}$-ban és $t_1, t_2, \dots, t_n \in \mathcal{T}$, akkor $f(t_1, t_2, \dots, t_n) \in \mathcal{T}$.

A természetes nyelvben a szavaknak csak bizonyos kombinációi alkotnak értelmes mondatokat. A mondatok megfelelői az elsőrendű logikában termekből épülő speciális szerkezetek. Ezeket formuláknak nevezzük, és a szintaktikájukat a következőképpen definiáljuk:

Definíció. (Formulák) Legyen $\mathcal{T}$ a $\mathcal{A}$ ábécé feletti termek halmaza. A $\mathcal{A}$ feletti formulák $\mathcal{F}$ halmaza az a legszűkebb halmaz, melyre:

- ha az p/n predikátumszimbólum $\mathcal{A}$ -ban és $t_1, t_2, \dots, t_n \in \mathcal{T}$, akkor $p(t_1, t_2, \dots, t_n) \in \mathcal{F}$;
- ha $S \in \mathcal{F}$, akkor $\neg S \in \mathcal{F}$;
- ha S és $T \in \mathcal{F}$, és o binér logikai összekötőjel, akkor (S o T) $\in \mathcal{F}$;
- ha $S \in \mathcal{F}$, és X egy változó $\mathcal{A}$ -ban, akkor $(\forall XS), (\exists XS) \in \mathcal{F}$.

A $p(t_1, t_2, \dots, t_n)$ alakú formulákat **atomi formuláknak**, vagy egyszerűen atomoknak nevezzük.

Kötött, szabad változó-előfordulás: A termek és az atomi formulák minden változójának minden előfordulása *szabad*. A $\neg A$ formulában egy változó-előfordulás akkor *kötött*, ha az A-ban van, és ott kötött. Az $(A \mathrm{o} B)$ formulában egy változó-előfordulás kötött, ha ez az

előfordulás *A*-ban van, és ott kötött, vagy *B*-ben van, és ott kötött. A *QXA* formulában *X* minden előfordulása kötött. A *Q* kvantor teszi kötötté *X* valamely előfordulását, ha ez az előfordulás *A*-ban még szabad. Egy *X*-től különböző változó valamely előfordulása kötött, ha *A*-ban kötött.

Változótisztaság: Egy formulát változóiban tisztának nevezünk, ha benne minden prefixumban a formula paramétereitől és bármely más prefixumban megnevezett változótól különböző változó van megnevezve.

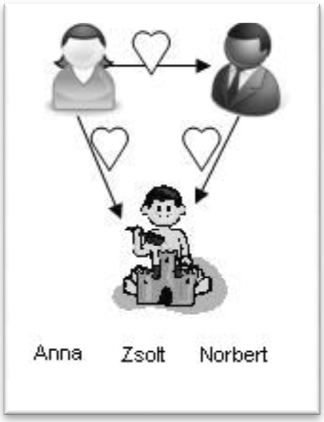
Paraméter, zárt formula, alapterm, alapformula: Ha egy változónak egy *K* kifejezésben van szabad előfordulása, akkor ezt a változót a kifejezés *paraméterének* fogjuk nevezni, jele: *Par(K)*. Ha a formulában egyetlen szabad változó-előfordulás sincsen, a formulát *zárt*nak nevezzük. A formulát/termet, amely nem tartalmaz egyetlen változót sem, *alapkifejezéseknek* nevezzük.

1.2. A nyelv szemantikája

1.2.1. Formulák szemantikája

Az előző részben bevezettük a formulák nyelvét, mely lehetőséget teremt arra, hogy formalizáljuk a természetes nyelv kijelentő mondatainak egy osztályát. Az állítások egy bizonyos világra vonatkoznak, és lehetnek igazak vagy hamisak ebben a világban. A logikai formulák jelentése is definiálható egy absztrakt világra vonatkozóan, és szintén vagy igazak, vagy hamisak. Más szavakkal, hogy definiálni tudjuk a formulák jelentését, formális kapcsolatot kell létrehoznunk a nyelv és az absztrakt világ között.

Amint már említettük, az állítások egyedekre és kapcsolataikra vagy szerepeikre vonatkoznak. Így a világ matematikai absztrakciója egyedek egy nemüres halmaza (univerzuma), néhány, az univerzum felett definiált függvénnyel és relációval. Például az (i)-(iii) mondatokkal kapcsolatos struktúra, lehet az ábrán látható világ egy absztrakciója. Ez a tartomány három egyedből áll – Anna, Norbert és Zsolt. Köztük három kapcsolatot fogunk figyelembe venni ebben a halmazban: egy unáris kapcsolatot, „… anya”, és két bináris kapcsolatot, „… a gyermeke…- nak” és „… szereti …- ot”. Az egyszerűség kedvéért ez a struktúra nem tartalmaz függvényt.



1. ábra.

A formulák nyelvének építőkövei a konstansok, függvényszimbólumok és predikátumszimbólumok voltak, ezért a nyelvet és szerkezetet a következő módon kapcsoljuk össze:

Definíció. (Interpretáció) Egy $\mathcal{A}$ ábécének $\mathfrak{I}$ interpretációja egy nemüres $\mathcal{D}$ halmaz (néha $|\mathfrak{I}|$-nek jelöljük), az interpretáció univerzuma, és egy olyan hozzárendelés, amely:

- minden $c \in \mathcal{A}$ konstanshoz egy $c_{\mathfrak{I}} \in \mathcal{D}$ elmet rendel;
- minden $f/n \in \mathcal{A}$ függvényszimbólumhoz $f_{\mathfrak{I}}: D^n \longrightarrow D$ függvényt rendel;
- minden $p/n \in \mathcal{A}$ predikátumszimbólumhoz pedig egy $\underbrace{p_{\mathfrak{I}} \subseteq \mathcal{D}\ x \ldots x\ \mathcal{D}}_{n}$ relációt rendel.

A konstansok, függvényszimbólumok valamint predikátumszimbólumok interpretációja alapot szolgáltat arra, hogy a nyelv formuláihoz igazságértéket tudjunk rendelni. A formula jelentését a formula összetevőinek jelentése függvényében fogjuk definiálni. Először a formula termjeinek jelentését fogjuk definiálni. Ha a term tartalmaz változókat, szükségünk van a változókiértékelés kiegészítő fogalmának bevezetésére. A φ **változókiértékelés** a változókhoz rendel az interpretáció tartományából értéket. Így, ez egy függvény (jele:φ), amely egy interpretáció objektumait a nyelv változóihoz rendeli. Ha X egy változó, akkor φ változókiértékelés a ω **változókiértékelés X-variánsa**, ha $\varphi(Y) = \omega(Y)$ minden X-től különböző Y változó esetén.

Definíció. (A termek szemantikája) Legyen $\mathfrak{I}$ egy interpretáció, φ egy kiértékelés és t egy term. Így $\varphi_{\mathfrak{I}}(t)$-ben t értékét a következőképpen definiáljuk:

- ha t egy c constans, akkor $\varphi_{\mathfrak{I}}(t) := c_{\mathfrak{I}}$;
- ha t egy X változó, akkor $\varphi_{\mathfrak{I}}(t) := \varphi(X)$;
- ha t $f(t_1, \ldots, t_n))$ alakú, akkor $\varphi_{\mathfrak{I}}(t) := f_{\mathfrak{I}}(\varphi_{\mathfrak{I}}(t_1), \ldots, \varphi_{\mathfrak{I}}(t_n))$.

Itt jegyezzük meg, hogy egy összetett termnek úgy tudunk értéket adni, hogy a függvényt a közvetlen résztermjeinek értékére vonatkoztatjuk, és a fenti definíciót rekurzívan alkalmazzuk.

Példa. Vegyünk egy olyan nyelvet, amely tartalmazza a $nulla$ konstanst, az egyváltozós s függvényt, és a $plusz$ kétváltozós függvényt. Feltesszük, hogy az $\mathfrak{I}$ tartománya a természetes számok ($\mathbb{N}$) halmaza, valamint:

$$nulla_{\Im} := 0$$
$$s_{\Im}(x) := 1 + x$$
$$plusz_{\Im}(x, y) := x + y$$

Ennél fogva a $nulla$ a 0 természetes számot, az s függvény számlálást, a $plusz$ függvény pedig az összeadást jelöli. A $plusz(s(zero), X)$ term értéke az $\Im$ interpretációban a φ változókiértékelés mellett (ami itt $\varphi(X) := 0$) a következőképpen adódik:

$$\varphi_{\Im}(plusz(s(nulla, X)) = \varphi_{\Im}(s(nulla)) + \varphi_{\Im}(X)$$
$$= (1 + \varphi_{\Im}(nulla)) + \varphi(X)$$
$$= (1 + 0) + 0$$
$$= 1$$

A formula értéke egy igazságérték. Az érték a formula összetevőitől függ, amelyek részformulák vagy termek. Következésképpen, a formula értékei a kiértékelésektől is függnek. A következő definícióban $a\ \Im \vDash_{\varphi} Q$ jelölés a következő állítás rövidítésére szolgál: „Q igaz az $\Im$ interpretációban a φ változókiértékelés mellett ” és $\Im \nvDash_{\varphi} Q$ jelentése: „Q hamis $\Im$ interpretációban a φ változókiértékelés mellett ”.

Definíció. (A formulák szemantikája) Legyen $\Im$ egy interpretáció, φ egy kiértékelés, és Q egy formula. Q értéke $\Im$ interpretációban és φ változókiértékelés mellett a következőképpen definiálható:

- $\Im \vDash_{\varphi} p(t_1, \dots, t_n) akkor\ és\ csak\ akkor, ha\ \langle \varphi_{\Im}(t_1), \dots, \varphi_{\Im}(t_n) \rangle \epsilon p_{\Im}$;
- $\Im \vDash_{\varphi} (\neg F)\ akkor\ és\ csak\ akkor, ha\ \Im \nvDash_{\varphi} F$;
- $\Im \vDash_{\varphi} (F \wedge G)\ akkor\ és\ csak\ akkor, ha\ \Im \vDash_{\varphi} F\ és\ \Im \vDash_{\varphi} G$;
- $\Im \vDash_{\varphi} (F \vee G)\ akkor\ és\ csak\ akkor, ha\ \Im \vDash_{\varphi} F\ vagy\ \Im \vDash_{\varphi} G\ (vagy\ mindkettő)$;
- $\Im \vDash_{\varphi} (F \supset G)\ akkor\ és\ csak\ akkor, ha\ \Im \vDash_{\Im} F\ esetén\ \Im \vDash_{\varphi} G$;
- $\Im \vDash_{\varphi} (\forall X F)\ akkor\ és\ csak\ akkor, ha\ \Im \vDash_{\varphi[X \to t]} F\ minden\ t \in |\Im| - re$;
- $\Im \vDash_{\varphi} (\exists X F)\ akkor\ és\ csak\ akkor, ha\ \Im \vDash_{\varphi[X \to t]} F\ legalább\ egy\ t \in |\Im| - re.$

A formulák szemantikája - ahogy fentebb is definiáltuk - a kiértékelés fogalmához kötődik, amely a formula változóhoz rendeli hozzá az interpretáció tartományának elmeit. Könnyű belátni, hogy egy zárt formula igazságértéke így egyedül az interpretációtól függ. Ezért bevett szokás a logikai programozásban, hogy az összes formulát univerzálisan kvantáltnak fogjuk

fel. Ezért bármikor, amikor egy formulában szabad változó-előfordulások vannak, akkor helyette a formula általános bezárt alakját vizsgáljuk. Mivel a kiértékelésnek nincs köze a zárt formulákhoz, ezt kihagyjuk, amikor a különböző formulák értékét vizsgáljuk.

Példa. Vizsgáljuk meg a következő interpretációt:

- az ábrán látható Zsolt, Norbert és Anna személyeket a *zsolt*, *norbert* és *anna* konstansokkal jelöljük
- a kapcsolatokat „…egy anya", „… a gyermeke ….-nak" és „… szereti…-ot", amelyeket az ábrán láthatunk, a következő predikátumszimbólumok jelölik*: anya*, *gyermeke* és *szereti.*

A fenti definíciót használva könnyű megmutatni, hogy a $\forall X \exists Y szereti(X,Y)$ formula értéke hamis az interpretációban (mivel Zsolt nem szeret senkit), míg a $\exists X \forall Y \neg szereti(Y,X)$ formula értéke igaz $\mathfrak{I}$-ben (mivel Annát nem szereti senki).

1.2.2. Modell és logikai következmény

A formulák nyelvének bemutatásával az volt a cél, hogy eszközt kapjunk a világok – algebrai struktúrák – leírásához. Adott zárt formulák egy $\mathcal{P}$ halmaza és egy $\mathfrak{I}$ interpretáció. Ekkor természetes, hogy felmerül a kérdés, hogy vajon $\mathcal{P}$ formulái megfelelőképpen írják-e le ezt a világot. Ez a helyzet, ha $\mathcal{P}$ összes formulája igaz $\mathfrak{I}$-ben.

Definíció. (Modell) Egy $\mathfrak{I}$ interpretációt a $\mathcal{P}$ formulahalmaz egy modelljének nevezzük, ha $\mathcal{P}$ minden formulája igaz $\mathfrak{I}$-ben.

Világos, hogy a $\mathcal{P}$ formuláit tartalmazó nyelvnek végtelen sok interpretációja létezik. Ugyanakkor előfordulhat, hogy egyikük sem modellje $\mathcal{P}$-nek. Az ilyen formula-halmazokat **kielégíthetetlennek** hívjuk. Ha van legalább egy modellje a formulahalmaznak, akkor az egy **kielégíthető** formulahalmaz.

Célunk, hogy a kérdéses világ leírását felhasználva még több információhoz jussunk ezzel a világgal kapcsolatban. Ezt az új információt új formulákkal reprezentáljuk, amelyek nem szerepelnek az eredeti leírásban. Más szavakkal, formulák egy adott $\mathcal{P}$ halmazához olyan más formulákat keresünk, amelyek szintén igazak a $\mathcal{P}$ által leírt világban

Definíció. (Logikai következmény) Legyen $\mathcal{P}$ zárt formulák egy halmaza. Az F zárt formulát a $\mathcal{P}$ formulahalmaz logikai következményének nevezzük (jele: $\mathcal{P} \vDash F$), ha F $\mathcal{P}$ minden modelljében igaz.

Példa. Hogy illusztráljuk ezt a jelölést példával, megmutatjuk, hogy (3) az (1) és (2) formulák logikai következménye. Legyen $\mathfrak{I}$ egy tetszőleges interpretáció. Ha $\mathfrak{I}$ (1) és (2) modellje, akkor:

(4) $\mathfrak{I} \vDash \forall X \forall Y \left(\left(anya(X) \wedge gyereke(Y, X) \right) \supset szereti(X, Y) \right)$

(5) $\mathfrak{I} \vDash anya(anna) \wedge gyereke(zsolt, anna)$

Mivel (4) fennáll bármely φ kiértékelés esetén:

(6) $\mathfrak{I} \vDash_{\varphi} anya(X) \wedge gyereke(Y, X) \supset szereti(X, Y)$

Vegyük a $\varphi(X) = anna_{\mathfrak{I}}$ és $\varphi(Y) = zsolt_{\mathfrak{I}}$ kiértékelést. Ezen kiértékelés mellett is fennáll (6), így:

(7) $\mathfrak{I} \vDash anya(anna) \wedge gyereke(zsolt, anna) \supset szereti(anna, zsolt)$

Végül, (5) és (6) alapján azt mondhatjuk, hogy $szereti(anna, zsolt)$ is igaz $\mathfrak{I}$-ben. Így (1) és (2) bármely modellje egyben a (3) modellje is.

Egy lehetséges módja, hogy bebizonyítsuk, hogy $\mathcal{P} \vDash F$, az, hogy megmutatjuk, hogy $\neg F$ a $\mathcal{P}$ minden modelljében hamis, vagy másképpen $\mathcal{P} \cup \{\neg F\}$ formulahalmaz kielégíthetetlen (nincs modellje).

Tétel. (Kielégíthetetlenség) Legyen $\mathcal{P}$ zárt formulák egy halmaza, és F egy zárt formula. $\mathcal{P} \vDash F$ akkor és csak akkor, ha $\mathcal{P} \cup \{\neg F\}$ kielégíthetetlen.

Gyakran egyszerűbb megmutatni, hogy az F formula nem logikai következménye a $\mathcal{P}$ formulahalmaznak. Ekkor elegendő megadni $\mathcal{P}$ egy olyan modelljét, amely F-nek nem modellje.

Egy másik fontos fogalom, amely a formulák szemantikáján alapszik, a logikai ekvivalencia.

Definíció. (Logikai ekvivalencia) Két formula (F és G) logikailag ekvivalens (jelölése: $F \sim G$), akkor és csak akkor, ha F-nek és G-nek megegyező igazságértéke van minden $\mathfrak{I}$ interpretációban és minden φ kiértékelés mellett.

1.3. Mit értünk logikai kalkuluson?

A válasz arra a kérdésre, hogy lehetne-e a klasszikus elsőrendű logikát szemantikai fogalmakra való hivatkozás nélkül, pusztán grammatikai szabályokra támaszkodva fölépíteni, határozottam pozitív.

Egy logikai rendszer szemantikamentes fölépítését szintaktikai fölépítésnek mondjuk. A szintaxis a jelekkel és ezekből épített szavakkal a szemantikai értékeikre és használati módjukra való hivatkozás nélkül foglalkozik.

Feladatunk most a szintaktikai következményreláció meghatározása, amely a szemantikai következményreláció helyettesítője szintaktikai fölépítésű rendszerekben.

A szintaktikai fölépítésű logikai rendszereket **logikai kalkulusoknak** szokás nevezni. Általánosan fogalmazva: egy logikai kalkulus nem más, mint egy formalizált nyelv vagy nyelvcsalád grammatikájának leírása, kiegészítve a szintaktikai következményreláció definíciójával. A kalkulus nem mondja meg, hogy mit kell csinálni, hanem csak arról szól, mit lehet csinálni, azaz alkalmazható szabályokat ad, melyek közül válogatni lehet valamely feladat megoldása során.

Az 1.1-es fejezetben (iii) mondatot az (i) és (ii) mondatokból kaptuk meg. A nyelvet formalizáltuk, és a mondatokat, mint az (1), (2) és (3) logikai formulákat. Ezzel a formalizálással a következtetést a formulák manipulációjának folyamataként is fölfoghatjuk, amelyből egy adott formulahalmaz, -mint pl. (1) és (2), melyeket **premisszáknak** nevezünk el-, egy új formulát eredményez, ami a **konklúzió**, pl. a (3). Célunk most, hogy formalizáljuk a következtetési elveket, az olyan szabályokat, amelyeket arra használhatunk, hogy új formulákat generáljunk a meglévőkből. Ezeket a szabályokat nevezzük **következtetési szabályok**nak. Szükséges, hogy a következtetési szabályok összhangban legyenek a helyes érvelési módokkal – valahányszor a premisszák igazak egy bizonyos világban, akkor egy következtetési szabályok alkalmazásával nyert konklúziónak is igaznak kell lennie ebben a világban.

Példa. A premisszák:

(10) $\forall X \left(\forall Y \left(anya(X) \wedge gyereke(Y, X) \supset szereti(X, Y)\right)\right)$

(11) $anya(anna) \wedge gyereke(zsolt, anna)$

(10) és (11) felhasználásával:

(12) $\forall Y(anya(anna) \wedge gyereke(Y, anna) \supset szereti(anna, Y))$

(12) és (11) felhasználásával:

(13) $anya(anna) \wedge gyereke(zsolt, anna) \supset szereti(anna, zsolt)$

Végül a (11) és (13)-ra vonatkozó leválasztási szabály alapján:

(14) $szereti(anna, zsolt)$

Így a (14) következtetést formális úton hoztuk létre a formális következtetési szabályok alkalmazásával. A példa illusztrálja a levezethetőség fogalmát. Ahogyan megfigyelhetjük, (14)-hez (10) és (11) segítségével nem közvetlenül jutottunk el, hanem bizonyos számú következtetési lépcsőkön haladva, ahol mindegyik adott egy új formulát a kiinduló premisszahalmazhoz. Bármely olyan F formulát, amelyet megkaphatunk formális következtetési szabályok segítségével egy adott $\mathcal{P}$ premisszahalmazból, $\mathcal{P}$-ből **levezethetőnek** hívunk. Jelölése: $\mathcal{P} \vdash F$.

Legyen $\mathcal{P}$ egy formulaosztály (a premisszák osztálya), F egy formula, S egy szemantikus logikai rendszer, K pedig egy kalkulus, amelyek egy közös L nyelvre épülnek.
Azt mondjuk, hogy K **helyes** S-re nézve, ha $\mathcal{P} \vdash F$ esetén mindig fönnáll $\mathcal{P} \vDash F$ is, vagyis K sohasem következtet hibásan, S-sel összevetve.
Azt mondjuk, hogy K **teljes** S-re nézve, ha $\mathcal{P} \vDash F$ esetén mindig fönnáll $\mathcal{P} \vdash F$ is, vagyis K reprodukálja az összes S-beli helyes következtetést.

Most azt vizsgáljuk, hogy egy kalkulusban hogyan alakul ki a szintaktikai következményreláció definíciója. Ez általában két lépésre tagolódik: az első lépésben bevezetik az alapformula fogalmát; ezek osztályát néhány séma segítségével szokás megadni, esetleg kiegészítve még egy rájuk alkalmazható szintaktikai operációval. Ezek a formulák a rendszer „logikai igazságai” közé tartoznak. A második lépésben tetszőleges $\mathcal{P}$ formulaosztályból való levezethetőség fogalmát értelmezik. Ha A alapformula, vagy eleme $\mathcal{P}$-nek, akkor levezethető $\mathcal{P}$ -ből, jelölése: $\mathcal{P} \vdash F$. Az érdekesebb szabályok arról szólnak, hogy ha $\mathcal{P}$-ből levezethető A_1, A_2, ..., akkor bizonyos további B is levezethetőnek számít.

Gyakori levezetési szabályok:

- modus ponens: ha rendelkezésünkre áll a $(F \supset G)$ formula és ennek egy bal oldali közvetlen részformulája, F is, akkor azt mondjuk, hogy modus ponenssel előáll a jobb oldali közvetlen részformula: G. $\boxed{\frac{F \quad F \supset G}{G}}$
- eliminációs szabály univerzális kvantorhoz: $\boxed{\frac{\forall X F(X)}{F(t)}}$
- bevezetési szabály konjukcióhoz: ha a levezetés során előállnak az F és G formulák, akkor az $F \wedge G$ formulát is leírhatjuk a levezetésben: $\boxed{\frac{F \quad G}{F \wedge G}}$

Röviden tehát: a szintaktikai következményreláció az alapformulák és a levezetési szabályok felsorolásán nyugszik. A döntési eljárások során mindig adott még egy ún. **megállási feltétel**, ami általában egy meghatározott formula megjelenése a levezetésben, vagy a levezetés speciális szerkezetűvé válása.

2. A logikai programozás elméleti alapjai

Automatikus tételbizonyításra többféle módszert, kalkulust is ismerünk. Ha programjainkat logikai állítások formájában fogalmazzuk meg, azaz a problémát sikerül tételbizonyítási feladattá átalakítani (a probléma, mint feltételformulák és speciális axiómaformulák lehetséges következménye), akkor program végrehajtásáról egy konstruktív tételbizonyítási algoritmus gondoskodhatna. Ha ennek a kalkulusnak adott a számítógépes implementációja, akkor a megadott formulák alapján ez el tudja dönteni, hogy az adott tételbizonyítási feladatra mi a válasz. Ha az implementált kalkulus úgy tekintjük, mint egy speciális számítógép, logikai gép, akkor a megadott formulák a logikai gép programjai. Ebben az esetben a logikai nyelvet programnyelvnek tekinthetjük.

2.1. A logikai program

A logikai programozást arra használjuk, hogy deklaratív leírásokból következtetésekhez jussunk. Ezek a leírások, amelyeket ezentúl logikai programnak fogunk nevezni, logikai formulák véges halmazát tartalmazza. A logikai program formuláira néhány megkötést kell megadnunk, hogy majd alkalmazni tudjunk rájuk egy viszonylag egyszerű és hatékony automatikus tételbizonyítási eljárást, az ún. SLD-rezolúciót (amelyet majd a következő részben tárgyalunk).

Első lépésben megkülönböztetjük a természetes nyelv kijelentő mondatainak két speciális fajtáját, a tényeket és a szabályokat. A **tények** az egyes egyedek tulajdonságait és a köztük lévő kapcsolatokat írják le. A **szabályok** azt fejezik ki, hogy ha bizonyos kapcsolat(ok) fennáll(nak) az egyedek között, az milyen egyéb kapcsolt(ok) fennállását bizonyítja.

Példaképpen vegyük a következő mondatokat:

(i) „Zsolt Norbert gyermeke.”
(ii) „Timi Zsolt gyermeke.”
(iii) „Norbert Attila gyermeke.”
(iv) „Kata Norbert gyermeke.”
(v) „Egy személy gyerekének a gyermeke ennek a személynek az unokája.”

Ezeket a mondatokat két lépésben formalizálhatjuk. Első lépésben atomi formulákkal írjuk le a tényeket:

1. $gyereke(zsolt, norbert)$
2. $gyereke(timi, zsolt)$
3. $gyereke(norbert, attila)$
4. $gyereke(kata, norbert)$

Az utolsó mondat egy szabály, amelyet a következőképpen formalizálhatunk (a „$:-$„ jel, amelyet a Prolog is használ, a fordított implikáció jele):

5. $\forall X \forall Y \left(unokája(X,Y) :- \exists Z \left(gyereke(X,Z) \wedge gyereke(Z,Y) \right) \right)$

Ekvivalens átalakítások sorozatával az előző formulából a következőt kaphatjuk:

$$\forall X \forall Y \forall Z \left(unokája(X,Y) :- \left(gyereke(X,Z) \wedge gyereke(Z,Y) \right) \right)$$

Tehát állításainkat speciális alakú formulákkal, ú.n. klózokkal írhatjuk le. Alakja a következő (a konjukció jel helyett a Prolog jelölési kovenciójának megfelelően vesszőt írunk, az univerzális kvantorokat pedig nem írjuk ki):

$A_0 :- A_1, \ldots, A_n \ (n \geq 0)$.

Ezzel ekvivalens

$A_0 \vee \neg A_1 \vee \ldots \vee \neg A_n$,

ahol $A_0 \ldots A_n$ atomi formulák, és minden változójuk univerzálisan kvantált. A tények így azok a klózok, ahol $n = 0$. Ebben az esetben a $:-$ jelet elhagyhatjuk, és úgy is felfoghatjuk, hogy a $:-$ jobb oldalán egy olyan formula áll, amely minden interpretációban igaz, ennek a jele: ■. Ellentettje a □, amely pedig minden interpretációban hamis.

Definíció. (Logikai program): logikai programnak nevezzük (program)klózok egy véges halmazát.

Mivel ezekből a klózokból végtelen sok következtetéshez tudna jutni a rendszer, nekünk kell kérdéseket megfogalmazni. Ezen kérdéseket **célklóz**nak nevezzük, formájuk a következő:

$:- A_1, \ldots, A_m$,

ahol az A_i atomi formulát részcélnak nevezzük. Azt a célt, ahol $m = 0$, üres célnak nevezzük, jele: □.

Példa. Az előző példában hozott családdal kapcsolatban a következő kérdéseket tehetjük fel:

	Kérdés:	Célklóz:
(i)	„Timi Zsoltnak a gyermeke?”	$:-gyereke(timi, zsolt)$
(ii)	„Ki Timi unokája?”	$:-unokája(X, timi)$
(iii)	„Kinek az unokája Zsolt?”	$:-unokája(zsolt, X)$
(iv)	„Ki kinek az unokája?”	$:-unokája(X, Y)$

A válaszok:

(i) egy egyszerű „yes”, azaz „igen”, mivel nem szerepel változó a kérdésben

(ii) mivel nem rendelkezünk információval Timi unokáival kapcsolatban, azért a kapott válasz „no”, azaz „nem” lesz.

(iii) mivel Zsolt Attila unokája, ezért a 3. kérdésre a válasz: $X = attila$ lesz.

(iv) az utolsó kérdésre 3 lehetséges válasz van:

$$X = zsolt \qquad Y = attila$$
$$X = kata \qquad Y = attila$$
$$X = timi \qquad Y = norbert$$

Lehetséges összetettebb kérdések megfogalmazása is, pl.: „Ki az a személy, aki Zsoltnak és Katának is az unokája?”. Formalizálva:

$$:-unokája(zsolt, X), unokája(kata, X)$$

Itt a válasz $X = attila$ lesz.

2.2. Az SLD-rezolúció

Ebben a fejezetben bemutatom azt a következtetési mechanizmust, amely a legtöbb logikai programozási rendszernek az alapja. A ’60-as évek közepén J. A. Robinson fejlesztett ki egy olyan következtetési eljárást -a rezolúciós elvet-, amely nemcsak logikai programoknál alkalmazható. Most ennek egy speciális változatát, az SLD rezolúciós elvet vizsgáljuk. Az SLD betűszó arra utal, hogy az algoritmus:

(D) definit programklózokat használ (az előző részben ismertetett programklózok pontos elnevezése),
(L) lineáris inputstratégiával dolgozik,
(S) a célklózban a feldolgozandó literált és a literálnak megfelelő partícióból a rezolválás során felhasználandó programklózt egy rögzített (S) stratégia alapján választja ki.

2.2.1. Kiegészítő fogalmak: termhelyettesítés, illesztő helyettesítés, átnevező helyettesítés

Ez a rész a helyettesítés rövid tárgyalását tartalmazza – alapvető fogalmakat az elkövetkezendő fejezethez. Formálisan a helyettesítés egy leképezés egy adott ábécé változóiból ugyanazon ábécé feletti termekre.

Definíció. (Helyettesítés) A *helyettesítés* termpárok egy véges $\{X_1/t_1, \dots, X_n/t_n\}$ halmaza, ahol minden t_i egy term és minden X_i egy változó, úgy ,hogy $X_i \neq t_i$, és $X_i \neq X_j$, ha $i \neq j$. Az üres helyettesítés jele: Є. A θ helyettesítés X-re való alkalmazása következőképpen definiálható:

$$X\theta := \begin{cases} t, ha\ X/t \in \theta; \\ X\ egyébként \end{cases}$$

Természetes, hogy a helyettesítések tartományát kibővíthetjük, így magában foglalja mind a termeket, mind a formulákat. Más szóval, a helyettesítés alkalmazható termre és atomi formulára a következőképpen:

Definíció. (A helyettesítés alkalmazása) Legyen θ a $\{X_1/t_1, \dots, X_n/t_n\}$ helyettesítés, és E egy term vagy formula. θ *alkalmazása* E-re egy term/atomi formula, amit úgy kapunk meg, hogy egyidejűleg X_i minden szabad előfordulását t_i-vel helyettesítjük E-ben $(1 \leq i \leq n)$.

Példa.

$$p(f(X,Z), f(Y,a))\{X/a, Y/Z, W/b\} = p(f, (a,Z), f(Z,a))$$

$$P(X,Y)\{X/f(Y), Y/b\} = p(f(y), b)$$

Lehetőség van továbbá a helyettesítések kompozíciójának megalkotására:

Definíció. (Helyettesítések kompozíciója) Legyen θ és δ két helyettesítés:

$$\theta := \{X_1/s_1, \dots, X_m/s_m\}$$
$$\delta := \{Y_1/t_1, \dots, Y_n/t_n\}$$

A θ és *σ kompozícióját* a következő halmazból kapjuk meg:

$$\theta\sigma := \{ X_1/s_1\delta, \dots, X_m/s_m\delta, Y_1/t_1, \dots, Y_n/t_n\}$$

úgy, hogy eltávolítunk minden $X_i/s_i\delta$ helyettesítést, amelyre $X_i = s_i\delta$ $(1 \leq i \leq m)$, és minden Y_j/t_j helyettesítést is, amelyre $Y_j \in \{X_1, \dots, X_n\}$ $(1 \leq i \leq n)$.

Példa.

$$\{X/f(Z), Y/W\}\{X/a, Z/a, W/Y\} = \{X/f(a), Z/a, W/Y\}$$

Tétel. (A helyettesítések tulajdonságai) Legyen θ, δ és γ helyettesítések, és E egy term, vagy atomi formula. Ekkor:

- $E(\theta\delta) = (E\theta)\delta$
- $(\theta\delta)\gamma = \theta(\delta\gamma)$
- $\epsilon\theta = \theta\epsilon = \theta$

Itt jegyezzük még meg, hogy a helyettesítések kompozíciója nem kommutatív (felcserélhető), ahogy a következő **példa** is mutatja:

$$\{X/f(Y)\}\{Y/a\} = \{X/f(a), Y/a\} \neq \{Y/a\}\{X/f(Y)\} = \{Y/a, X/f(Y)\}$$

Definíció. (Helyettesítés általánosítása) Legyenek θ és δ helyettesítések. Egy θ helyettesítés *általánosabb* a δ helyettesítésnél, ha van olyan γ helyettesítés, hogy $\sigma = \theta\gamma$. Jele:$\delta \preccurlyeq \theta$.

Definíció. (Illesztő helyettesítés, legáltalánosabb illesztő helyettesítés) Legyen $W = \{A_1, \dots, A_k\}$ azonos predikátumszimbólumot tartalmazó atomi formulák legalább kételemű, véges halmaza. Az olyan θ helyettesítést, amelyre $A_1\theta, \dots, A_k\theta$ rendre azonosak, W *illesztő helyettesítésének* nevezzük. W illesztő helyettesítése W *legáltalánosabb illesztő helyettesítése*, ha W minden illesztő helyettesítésénél általánosabb. A későbbiekben ezt MGU $(Most\ General\ Unification)$-val fogjuk jelölni.

Példa.

A $(P(X, f(A, Y))$ és a $P(B, Z)$ atomoknak egy illesztő helyettesítse:

$$\{X/B, Y/C, Z/f(A, C)\}$$

Legáltalánosabb illesztő helyettesítése:

$$\{X/B, Z/f(A, Y)\}$$

A következőkben megnéznénk egy illesztési algoritmust két atomi formula legáltalánosabb illesztő helyettesítésének megtalálására, azonban előtte még meg határoznunk egy fogalmat:

Definíció. (Összeférhetetlenségi halmaz) Vizsgáljuk W elemeit párhuzamosan, szimbólumonként balról jobbra haladva. Álljunk meg annál az első szimbólumnál, amelyik W nem minden atomi formulájában egyforma. Emeljük ki W minden atomi formulájából azt a résztermet, amely az ezen a pozíción lévő szimbólummal kezdődik. E résztermek D halmazát W *összeférhetetlenségi halmazának* nevezzük.

Az illesztő algoritmus:

1. $k \coloneqq 0$, $W_k \coloneqq W$, $\delta_k \coloneqq \epsilon$.
2. Ha W_k egyetlen atomot tartalmaz, akkor sikeresen vége: δ_k a W legáltalánosabb illesztő helyettesítése. Egyébként határozzuk meg $W_k$összeférhetetlenségi halmazát: D_k-t.
3. Ha van D_k-ban olyan X_k individuumváltozó és t_k term, hogy X_k nem fordul előt_k-ban, akkor a 4. lépéssel folytatjuk. Egyébként sikertelenül vége, W nem illeszthető.
4. $\delta_{k+1} \coloneqq \delta_k(X_k||t_k)$, $W_{k+1} \coloneqq \{A(X_k||t_k)|A \in W_k\}$.
5. $k \coloneqq k + 1$, és a 2. lépéssel folytatjuk.

Tétel. Ha W egymáshoz illeszthető atomi formulák véges, nemüres halmaza, akkor az illesztő algoritmus mindig a 2. lépéssel fejeződik be, és az utolsó δ_k legáltalánosabb illesztő helyettesítés lesz W-re.

Definíció. (Átnevező helyettesítés) Egy $\{X_1/Y_1, \dots, X_n/Y_n\}$ helyettesítés átnevező helyettesítés, ha $Y_1, \dots, Y_n$ az $X_1, \dots, X_n$ egy permutációja. Az átnevező helyettesítés változók (vagy több általános term) közötti bijektív leképezés. Ez a helyettesítés mindig megőrzi a term struktúráját.

2.2.2. Informális bevezetés egy példán keresztül az SLD-rezolúcióhoz

A logikai program klózai (az eddig megszokott jelöléssel):

$$A_0 : -A_1, \dots, A_n \ (n \geq 0)$$

alakúak, a célklóz pedig

$: -C_1, \dots, C_n \ (n \geq 0)$

alakú, ahol $A_0, \dots, A_n, C_1, \dots, C_n$, atomi formulák. Nézzük a következő **példát**, ahol világunkat az alábbi mondatokkal írhatjuk le: „Egy újszülött szülei büszkék.”, „Tibor Eszter apja”. „Eszter újszülött”. Formalizálva:

$buszke(X) : -szuloje(X, Y), ujszulott(Y).$

$szuloje(X, Y) : -apja(X, Y).$

$szuloje(X, Y) : -anyja(X, Y).$

$apja(tibor, eszter).$

$ujszulott(eszter).$

Megfigyelhető, hogy ez a program csak pozitív tudást ír le, nem tartalmaz olyat például, hogy ki nem büszke. Szintén nem található meg benne, hogy ki nem szülő. A negációt speciális módon tudjuk majd kezelni, de erre itt most nem térünk ki.

Nézzük azt az esetet, amikor a következő kérdést szeretnénk feltenni: „Ki büszke?”. Természetesen majd azt szeretnénk látni, hogy a kérdésre $tibor$ a válasz. Ahogy az előző részekben is tárgyaltuk, a kérdő mondatot a következőképpen tudjuk formalizálni:

$: -buszke(Z)$ (G_0)

Ez a jelölés egyenértékű a $\forall Z \neg buszke(Z)$ formulával, ami pedig ekvivalens a $\neg \exists Z buszke(Z)$ formulával. Így olvashatjuk: „Senki sem büszke”. A célunk most, hogy megmutassuk, hogy ez a kijelentés hamis a $\mathcal{P}$ (formulahalmaz, amely a fenti formulákat tartalmazza) minden modelljében, tehát $\mathcal{P} \vDash \exists Z buszke(Z)$. De nem elég, ha erre a kérdésre igen választ kapjunk, kell, hogy találjunk egy olyan θ helyettesítést, hogy a $\mathcal{P} \cup \{\neg buszke(Z)\theta\}$ halmaz kielégíthetetlen legyen, vagyis $\mathcal{P} \vDash buszke(Z)\theta$.

Elsőnek vegyük (G_0)-át, tehát „Minden Z-t, ahol Z nem büszke”. Megvizsgálva a programot találhatunk olyan mondatot, amely azt fejezi ki, hogy valaki mikor büszke:

$buszke(X) : -szuloje(X, Y), ujszulott(Y).$ (C_0)

Logikailag ekvivalens formája:

$$\forall \left(\neg buszke(X) \supset \neg \big(szuloje(X, Y) \wedge ujszulott(Y) \big) \right)$$

Ha X-et Z-re átnevezzük, eltávolítjuk az univerzális kvantort, és alkalmazzuk a modus ponens szabályt (G_0)-ra, akkor a következő formulát kapjuk:

$\neg\big(szuloje(Z,Y) \wedge ujszulott(Y)\big)$, ezzel ekvivalens:

$:-szuloje(Z,Y), ujszulott(Y)$ (G_1)

Összességében tehát az első lépésben a (G_0) célt egy másik, (G_1) céllal helyettesítettük, amely igaz $P \cup \{(G_0)\}$ minden modelljében. Az maradt hátra, hogy bebizonyítsuk, hogy $\mathcal{P} \cup \{(G_1)\}$ kielégíthetetlen. Itt jegyezzük meg, hogy (G_1) ekvivalens a következő formulával:

$\forall Z \forall Y\big(\neg szuloje(Z,Y) \vee \neg ujszulott(Y)\big)$

Így akkor tudjuk megmutatni, hogy (G_1) kielégíthetetlen $\mathcal{P}$ mellett, ha $\mathcal{P}$ minden modelljében létezik legalább egy személy, aki a szülője egy újszülöttnek. Először ellenőrizzük, hogy vannak-e szülők.

A program tartalmazza a következő klózt:

$szuloje(X,Y) :- apja(X,Y)$. ezzel ekvivalens: (C_1)

$\forall X \forall Y(\neg szuloje(X,Y) \supset \neg apja(X,Y)$

Ezek alapján (G_1) átírható:

$:-apja(Z,Y), ujszulott(Y)$. (G_2)

Az új (G_2) célból megmutathatjuk, hogy kielégíthetetlen $\mathcal{P}$ mellett, ha $\mathcal{P}$ minden modelljében létezik legalább egy olyan szülő, aki az apja egy újszülöttnek. A program szerint „Tibor Eszter apja.”, azaz:

$apja(tibor, eszter)$. (C_2)

Már csak az maradt hátra, hogy az „Eszter nem újszülött” kijelentésről bebizonyítsuk, hogy $\mathcal{P}$ mellett kielégíthetetlen:

$:-ujszulott(eszter)$. (G_3)

De a program tartalmazza a következő tényt:

$ujszulott(eszter)$. (C_3)

A fenti ekvivalens a $\neg ujszulott(eszter) \supset \square$ formulával, ami elvezet minket a rezolúciós cáfolathoz:

$\square$ (G_4)

2.2.3. Az SLD-rezolúció lépései

A fent alkalmazott levezetési szabály általánosítása:

$$\frac{\forall \neg(A_1 \wedge \ldots \wedge A_{i-1} \wedge A_i \wedge A_{i+1} \wedge \ldots \wedge A_m)\ \forall(B_1 \wedge \ldots \wedge B_n \supset B_0)}{\forall \neg(A_1 \wedge \ldots \wedge A_{i-1} \wedge B_1 \wedge \ldots \wedge B_n \wedge A_{i+1} \wedge \ldots \wedge A_m)\theta}$$

A logikai programoknál használt jelöléssel ugyanez:

$$\frac{:-A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_m \quad B_0 :- B_1, \ldots, B_n}{:-(A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_m)\theta}$$

ahol:

(i) $A_1, \ldots, A_m$ atomi formulák,

(ii) $B_0 :- B_1, \ldots, B_n$ egy $\mathcal{P}$-beli (átnevezett) klóz $(n \geq 0)$,

(iii) $MGU(A_i, B_0) = \theta$.

A fenti szabálynak két premisszája van: egy célklóz és egy általános programklóz. Mindkét klóz a bennük szereplő összes változó szerint univerzálisan kvantált. Így lehetőség van arra, hogy átnevezzük a programklózok változóit úgy, hogy a klózok egymáshoz képest változótiszták legyenek.

Tekintsünk egy $\mathcal{P}$ programot, és nézzük meg, hogyan alkalmazzuk rá az SLD-rezolúciót. Legyen G_0a célklóz, amely a következő alakú: $:-A_1, \ldots, A_m$ $(m \geq 0)$.

A stratégia alapján kiválasztjuk a célklózból valamely A_i literált (amennyiben lehetséges). Eztán kiválasztunk néhány átnevezett $B_0 :- B_1, \ldots, B_n$ $(n \geq 0)$ programklózt, aminek a fejét illesztjük a θ_1 helyettesítéssel A_i-vel. Ekkor a rezolúciós levezetési szabályunk eredménye: $:-(A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_m)\theta_1$, az új G_1 célklóz.

A rezolúciós lépés sikeres végrehajtása után a rezolúciós szabály segítségével tovább folytathatjuk az eljárást, így G_2-t is megkapjuk, és így tovább. Lehet olyan eset, amikor az eljárást nem tudjuk folytatni. Két oka lehet, hogy G_i-ből nem kapom meg G_{i+1}-et:

- ha a kiválasztott literált nem lehet kirezolválni, azaz nem illeszthető egyetlen programklóz fejével sem
- ha $G_i = \square$ (tehát megkaptuk az üres célt)

A fenti eljárás a kezdeti célklózból kiinduló véges vagy végtelen sok célklózból álló sorozatot ad eredményül. Minden lépésben egy programklózt (átnevezett változókkal) akarunk rezolválni valamilyen $\Re$ stratégia alapján kiválasztott célklóz literállal, miután elkészítjük a legáltalánosabb illesztő helyettesítésüket. Ezek alapján egy $\langle G_i, C_i \rangle$ alakú klózpár játszik szerepet az eljárás egy-egy lépésében, ahol G_i egy célklóz és C_i pedig egy programklóz, amelynek változóin átnevezést hajtottunk végre. Az $\Re$ stratégia, valamint G_i és C_i együttesen egyértelműen meghatározzák a legáltalánosabb illesztő helyettesítést, amit θ_{i+1}-gyel jelölünk. A G_{i+1} célklózra azt mondjuk, hogy **közvetlen levezetés**sel kaptuk a G_i-ből és C_i-ből az $\Re$ stratégia alapján. Azt is mondhatjuk, hogy G_i-t és C_i-t rezolválva megkaptuk a G_i-t.

Definíció. (SLD-levezetés) Legyen G_0 egy célklóz, $\mathcal{P}$ egy logikai program és $\Re$ egy stratégia. G_0 *SLD-levezetése* véges vagy végtelen célok $G_0 \overset{C_0}{\rightarrow} G_1 \dots G_{n-1} \overset{C_{n-1}}{\longrightarrow} G_n \dots$ sorozata, ahol minden G_{i+1} közvetlen levezetés G_i-ből és C_i-ből az $\Re$ stratégia alapján.

Minden SLD-levezetés a következő alakú: $G_0 \overset{C_0}{\rightarrow} G_1 \dots G_{n-1} \overset{C_{n-1}}{\longrightarrow} G_n \dots$. Közben a legáltalánosabb illesztő helyettesítések egy $\theta_1, \dots, \theta_n$ sorozatát kapjuk. A levezetés **kiszámított helyettesítésének** nevezzük θ-t, amit a következőképpen kapunk meg:

$$\theta := \begin{cases} \theta_1\theta_2 \dots \theta_n, ha\ n > 0, \\ \epsilon, ha\ n = 0. \end{cases}$$

Definíció. (SLD-cáfolat) Legyen adott egy véges SLD-levezetés:

$$G_0 \overset{C_0}{\rightarrow} G_1 \dots G_{n-1} \overset{C_{n-1}}{\longrightarrow} G_n \dots$$

Ha $G_{n+1} = \square$, akkor azt mondjuk rá, hogy a levezetés *cáfolat.*

Definíció (Válaszhelyettesítés) Legyen G_0 egy célklóz, és a G_0-ban előforduló változók egy SLD-cáfolatának egy tetszőleges kiszámított helyettesítése. Ekkor ezt a helyettesítést a G_0-ra vonatkozó *kiszámított válaszhelyettesítésnek* nevezzük.

Példa. Vegyük az előzőekben tárgyalt példaprogramot és a következő célt: $: -buszke(Z)$.

G_0 : $:-buszke(Z).$

C_0 : $buszke(X_0):-szuloje(X_0,Y_0),ujszulott(Y_0).$

$buszke(Z)$ és $buszke(X_0)$ legáltalánosabb illesztő helyettesítése: $\theta_1 = \{X_0\backslash Z\}$. Az első lépés után:

G_1 : $:-szuloje(Z,Y_0),ujszulott(Y_0).$

C_1 : $szuloje(X_1,Y_1):-apja(X_1,Y_1).$

A stratégia szerint most mindig a következő legbaloldalibb részcélt használjuk. A második rezolúciós lépés során a legáltalánosabb illesztő helyettesítés: $\theta_1 = \{X_1\backslash Z, Y_1\backslash Y_0\}$. Az ezt követő lépések:

G_2 : $:-apja(Z,Y_0),ujszulott(Y_0).$

C_2 : $apja(tibor,eszter).$

G_3 : $:-ujszulott(eszter).$

C_3 : $ujszulott(eszter).$

G_4 : $\square.$

A válaszadó helyettesítés:

$$\theta_1\theta_2\theta_3\theta_4 = \{X_0\backslash Z\}\{X_1\backslash Z, Y_1\backslash Y_0\}\{Z\backslash tibor, Y_0\backslash eszter\}\epsilon$$
$$= \{X_0\backslash tibor, X_1\backslash tibor, Y_1\backslash eszter, Z\backslash tibor, Y_0\backslash eszter\}$$

Z-t $tibor$-ral helyettesítettük, így a feltett kérdésünkre – Ki büszke? – megkaptuk a választ: Tibor büszke.

Egy másik **példa**program:

$nagyapja(X,Z):-apja(X,Y),szuloje(Y,Z).$
$szuloje(X,Y):-apja(X,Y).$
$szuloje(X,Y):-anyja(X,Y).$
$apja(a,b).$
$anyja(b,c).$

A : $-nagyapja(a, X)$ célklóz levezetése:

$:-nagyapja(a,X).$
$\quad \big| \quad nagyapja(X_0, Z_0) :- apja(X_0, Y_0), szuloje(Y_0, Z_0).$
$:-apja(a, Y_0), szuloje(Y_0, X).$
$\quad \big| \quad apja(a, b).$
$:-szuloje(b, X).$
$\quad \big| \quad szuloje(X_2, Y_2) :- anyja(X_2, Y_2).$
$:-anyja(b, X).$
$\quad \big| \quad anyja(b, c).$
$\quad \square$

Mivel X helyébe c-t helyettesítettünk, megkaptuk a választ: "a" "c"-nek a nagyapja.

Definíció (Kudarcos levezetés) Egy G_0 egy célklóz egy levezetését kudarcosnak nevezzük, ha az utolsó elem nemüres, és már nem tudjuk rezolválni a program egyetlen klózával sem.

Példa. Az előző program és célklóz egy másik levezetése:

$:-nagyapja(a, X).$
$\quad \big| \quad nagyapja(X_0, Z_0) :- apja(X_0, Y_0), szuloje(Y_0, Z_0).$
$:-apja(a, Y_0), szuloje(Y_0, X).$
$\quad \big| \quad apja(a, b).$
$:-szuloje(b, X).$
$\quad \big| \quad szuloje(X_2, Y_2) :- apja(X_2, Y_2).$
$:-apja(b, X).$

Egy teljes levezetés vagy cáfolatot vagy kudarcos levezetést jelent Egy adott kezdeti G_0 célklóznak lehet több teljes levezetése is egy adott $\mathfrak{R}$ statégiára nézve, ha egy kiválasztott belső literált egynél több programklózzal is tudunk rezolválni. A G_0 SDL-fája az összes ilyen levezetést ábrázolja.

Definíció (SLD-fa) Legyen G_0 egy célklóz, $\mathcal{P}$ egy logikai program és $\mathfrak{R}$ egy stratégia. G_0 SLD-fája egy olyan felcímkézett fa, amely kielégíti a következő feltételeket:

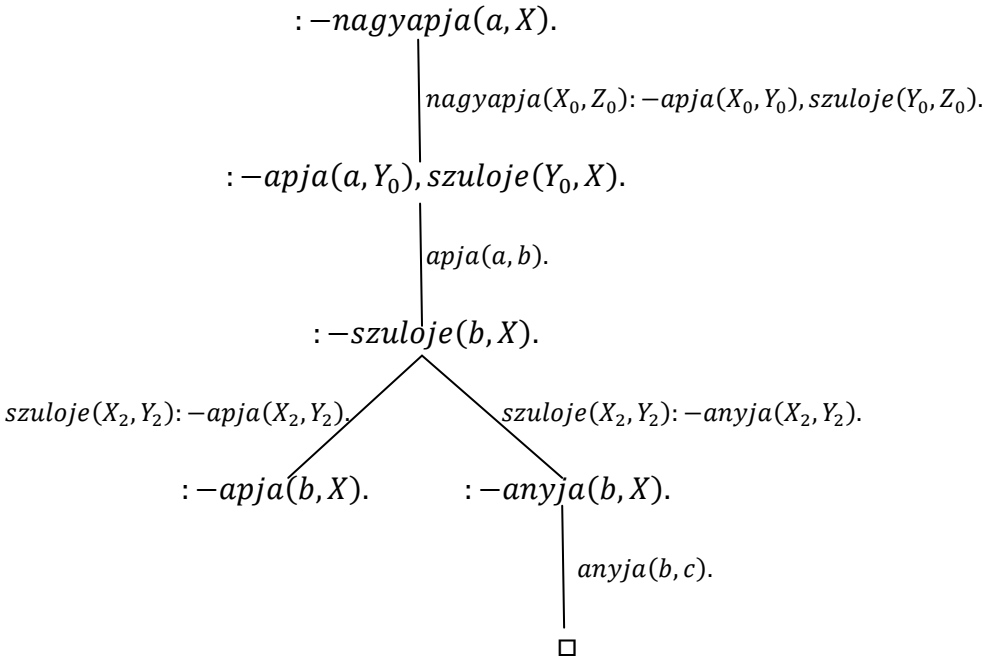
- G_0 a fa gyökere és

- ha a fa tartalmaz egy G_i-vel jelölt csomópontot, és van olyan $C_i \in \mathcal{P}$ átnevezett klóz, hogy G_i az $\Re$ stratégia alapján rezolválható C_i-vel és rezolválja G_{i+1}, akkor a G_i gyermeke G_{i+1} lesz, a közöttük futó él címkéje pedig C_i.

Az SLD-fa csomópontjai a rezolúciós levezetés célklózaival vannak elnevezve, az élek pedig a program klózaival. Egyértelmű megfeleltetés létezik az SLD-fa útjai és a G_0 teljes levezetés között egy meghatározott statégia mellett.

A következő $G_0 \overset{C_0}{\rightarrow} G_1 \ldots G_k \overset{C_k}{\rightarrow} \ldots$ sorozat pontosan akkor lesz a G_0 teljes levezetése az $\Re$ stratégia alapján akkor és csak akkor, ha létezik az SLD-fának egy olyan útja, ahol minden szomszédos G_i, G_{i+1} csúcsot egy C_i-vel címkézett él köt össze.

Példa. (az előző programra):



Tétel. (Az SLD-rezolúció helyessége) Legyen $\mathcal{P}$ egy logikai program, $\Re$ egy stratégia és θ pedig a $:-A_1,\ldots,A_m$ $(m \geq 0)$ célklóz válaszhelyettesítése az $\Re$ statégia alapján. Ekkor $\forall((A_1 \wedge \ldots \wedge A_m)\theta)$ logikai következménye a $\mathcal{P}$ programnak.

Tétel. (Az SLD-rezolúció teljessége) Legyen $\mathcal{P}$ egy logikai program, $\Re$ egy stratégia és $:-A_1,\ldots,A_m$ $(m \geq 0)$ egy célklóz. Ha $\mathcal{P} \vDash \forall((A_1 \wedge \ldots \wedge A_m)\sigma)$, akkor létezik az $:-A_1,\ldots,A_m$ célklóznak cáfolata az $\Re$ statégia alapján a θ válaszhelyettesítéssel, melyre $(A_1 \wedge \ldots \wedge A_m)\sigma$ a $(A_1 \wedge \ldots \wedge A_m)\theta$-nak egy esete.

3. A Prolog

3.1. Története

Robert Kowalski elméleti munkája alapján Alain Colmerauer és csoportja hozta létre az logikai programozási paradigma első megvalósítását, a Prolog nyelvet 1972-1973-ban a marseille-i egyetemen. A Prolog név a francia eredetű programmation en logique (logikai programozás) kifejezés rövidítése.

Az első kísérleti Prolog interpreter Algol-W nyelven készült, míg a ténylegesen terjesztett változat Fortranban íródott.

A matematikai logika számítástudományi alkalmazásával már az 1970-es évek elejétől foglalkoztak Magyarországon, eljutott hozzánk a Marseille Prolog Fortran implementációja, dokumentációja és néhány további kutatási jelentés. 1975 májusára Szeredi Péter a világon másodikként elkészített egy teljesen saját fejlesztésű Prolog értelmezőt, mely a Marseille Prolog alapelveire épített, de annak kódjától teljesen függetlenül íródott. Fejlesztési eszköze a CDL volt.

Ezek után Magyarországon, és a világon máshol is számos Prolog alkalmazást fejleszettek ki. Én a szakdolgozatom készítése során az 1980-ban kifejlesztett LPA Win Prologot használtam.

3.2. Szintaxis

A Prolog nyelv építőkövei:

- **változók**: nagybetűvel vagy aláhúzás jellel kezdődő jelsorozat;
- **konstansok**: a hagyományos értelemben használt, kisbetűvel kezdődő azonosító, számkonstans, üres lista vagy idézőjelek között karaktersorozat lehet;
- **függvényszimbólum**;
- **termek**: individuumok körülírására használt, meghatározott formájú karaktersorozat, amely lehet konstans, változó, vagy termekből függvényszimbólummal képzett alakú;
- **atomi formulák**: egy n-argumentumú p predikátum és $t_1, \dots t_n$ termek esetén a $p(t_1, \dots t_n)$ egy atomi formula;
- a 2.4 fejezetben a logikai programról leírtaknak megfelelően épülnek fel a programklózok, beleértve a **tények**et, **szabályok**at és a **célklóz**t.

Speciális, és gyakran használta adatszerkezet a **lista**. Az üres lista jele: []. Nem üres lista esetén van a listának egy X feje, és az F hátramaradó része, a farka, ami maga is lista, Ekkor a következőképpen jelöljük a listát : $[X|F]$.

Használhatunk még az LPA Prologban néhány kényelmes beépített lehetőséget is, mint például a metapredikátumok, interpreter-vezérlők és interface parancsok.

- A *metapredikátumok* segítségével a felhasználó képes a Prolog termek osztályzására.
- Az *interpreter-vezérlők* közé tartozik pl. a **fail** (kudarc), melyet az SLD-algoritmus visszalépését vezérli a levezetésgráfban. Hasonló a mindig igaznak számító **true** predikátum. A **cut** (vágás) hatására a levezetésgráf bizonyos ágain az SLD-algoritmus nem megy végig. Ez is a visszalépések szabályzása útján hat.
- *Interface parancso*k pl. a **write** – szöveg kiírása a képernyőre.

3.3. Működése

Egy logikai program futása során egy háttérben működő algoritmus ellenőrzi, hogy a célklóz a porgramklózok logikai következménye-e. Az ellenőrzés során az SLD-rezolúció algoritmizált számítógépes megvalósítása működik. A programozó tud paraméterezni bizonyos beállításokat, ennek hatására némileg módosulhat a futtatókörnyezet által végrehajtott algoritmus. Ily módon tehát a Prolog az automatikus tételbizonyítás egy, a korlátai ellenére nagy számítóerejű megvalósítása.

Az algoritmus:

1. Kezdetben az aktuális célklóz a program eredeti célklóza.
2. Ha az aktuális célklóz az üres klóz, akkor pozitív eredménnyel vége a programnak: rezolúciós cáfolatot kaptunk, a célklóz következménye a program többi klózának. Ekkor a 6. lépés következik, egyébként (ha nem üres klóz az aktuális) a következő, azaz 3. lépés.
3. Az aktuális célklózból kiválasztja balról az első literált, ez az aktuális literál.
4. Ha van olyan partíció, amely az aktuális literállal kezdődik, akkor e partíció legelső (az előző lépések során még ki nem választott) klózát kiválasztjuk. Ha nincs ilyen partíció, vagy ennek már nincs kiválasztható klóza, akkor a 6. lépés következik, egyébként az 5. lépés.

5. Illesztjük a kiválasztott partíció kiválasztott klózát az aktuális klózzal és képezzük az így kapott két klóz elsőrendű rezolvensét. Ez lesz az új aktuális célklóz, és visszalépünk a 2. lépésre. Egyébként (ha az illesztés vagy a rezolúció nem hajtható végre), a választás kudarcosnak minősül, és visszalépünk a 4. lépéshez.
6. Ha az aktuális célklóz az eredeti célklóz, az algoritmusnak vége. Egyébként pedig a jelenlegi aktuális célklóz törlődik, és az azt megelőző aktuális célklóz lesz újra az aktuális célklóz. Ekkor pedig visszalépünk a 3. lépésbe.

Ez az algoritmus a levezetésgráf „mélységben először” bejárási stratégiájának felel meg.

Bármely ciklusa végrehajtásakor, az ún. *visszalépés*ek végrehajthatósága miatt, szükség van az előző lépések során végrehajtott lépésekről (gyakorlatilag az összes előforduló központi klózról és mellékklózról) való információ tárolására.

Az illesztés végrehajtása: adott az s és t term. A két term a következőképpen illeszthető:

- ha s és t is konstansok, csak akkor illeszthetőek, ha egyformák (ugyanazt az individuumot jelölik)
- ha s egy változó, t akármi lehet, illeszthetőek, ekkor s helyébe t-t helyettesítünk (a hatékonyság érdekében ki lehet kapcsolni a funkciót, amely ellenőrzi a változóütközést).
- ha s és t egyéb struktúrájúak, csak akkor illeszthetőek, ha:
    - s és t függvényszimbóluma ugyanaz és
    - minden argumentumuk illeszthető

3.4. Szemantika

A Prolog rekurzív logikai formulákat kezelni képes rendszer. Az ilyen rendszerek vizsgálatának leggyakoribb matematikai eszköze az ún. fixpontlogika. A programnyelv-szemantika a programok mint egy programnyelv kifejezéseinek értelmezésével, a program jelentésének vizsgálatával foglalkozik. Ezen belül:

- a **deklaratív szemantika** vizsgálja, hogy a programot mint formulahalmazt értelmezve, milyen logikai következmények vonhatóak le belőle;
- az **eljárásszemantika** vizsgálja, hogy a program mint algoritmikusan megvalósított kalkulus milyen (azaz pl. milyen korlátai vannak a lineáris inputrezolúciónak);

- a **kiszámítási szemantika** pedig, hogy a klóz- illetve literálsorrendnek milyen szerepe van a program futásában.

3.5. Alkalmazása

A Prologot gyakran használják *mesterségesintelligencia-alkalmazások* megvalósítására, illetve a számítógépes nyelvészet eszközeként (különös tekintettel a természetes nyelvfeldolgozásra, melyre eredetileg tervezték). A kutatások egyik fontos támogatója és gerjesztője volt az a tény, hogy a Prolog-variánsokat (pl. a Kernel Nyelvet) operációs rendszerként használó számítógépek előállítására, az ún. Ötödik Generációs Számítógéprendszer-Projektre az 1980-s években elég nagy hangsúlyt fektettek (elsősorban japán kezdeményezésre), bár mára az eredeti kutatás leállt. Manapság a fő kutatási terület a nyelv felhasználása tudásalapú rendszerekre.

"Az emberi intelligencia és a döntéshozó képesség modellezésére tervezett és fejlesztett számítógépes programok, melyekkel megkísérlik az egyszerű gondolkodást és más emberi tulajdonságokat reprodukálni."

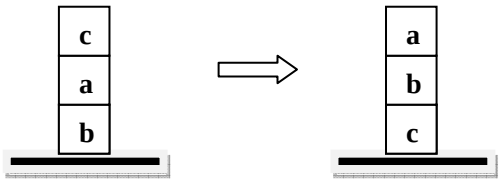
Magyar Számítástechnikai Szótár

4. Mesterséges intelligencia feladatok és megoldásaik

4.1. Bevezető fogalmak az alapvető megoldás-kereső stratégiákhoz

Ebben a fejezetben a problémák reprezentálásához egy általános sémát, az úgynevezett állapottér-reprezentációt fogjuk használni. Az állapottér-reprezentáció egy olyan gráfként ábrázolható, amelynek csúcsai a probléma állapotai, az állapotváltozásokat leíró operátorokat szemléltetik az irányított élek. Így a feladat megoldás keresése útkeresés ebben a gráfban.

Nézzük az ábrán látható **példá**t! Feladatunk, hogy átrendezzük a kockákat az ábra szerint úgy, hogy egyszerre csak egy, felül lévő kockát helyezhetünk át, és vagy az asztalra vagy pedig egy másik, felül lévő kocka tetejére tehetjük.

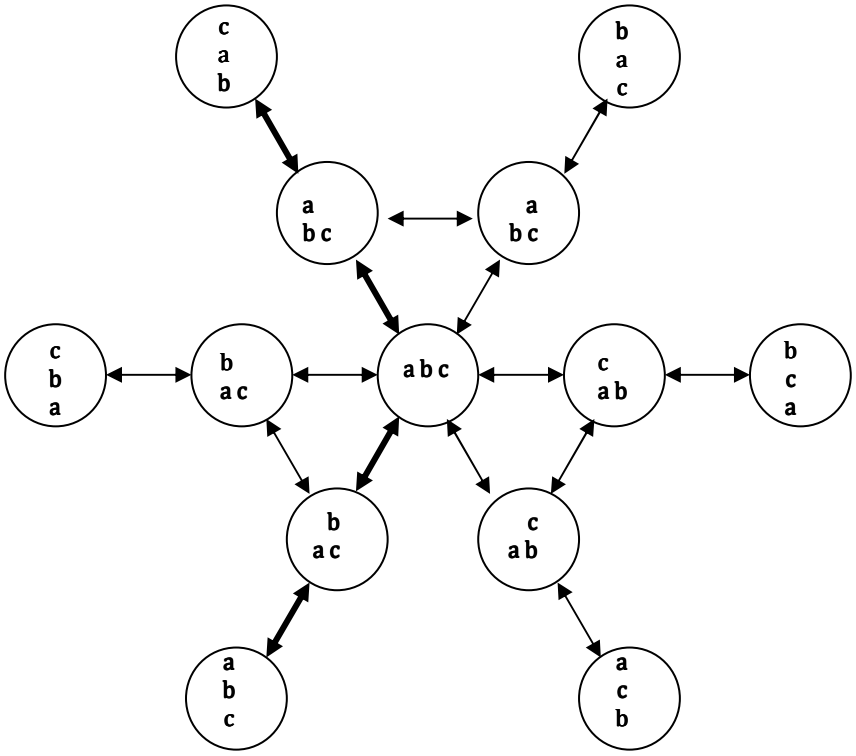


2. ábra.

A feladat megoldását tekinthetjük úgy is, mint a különböző alternatívák közötti választásokat. A kezdőállapotban egyetlen lehetőségünk van: a „**c**” kockát az asztalra tesszük. Ezután három alternatíva közül választhatunk:

- az „**a**” kockát az asztalra tesszük
- az „**a**” kockát a „**c**” kockára tesszük
- a „**c**” kockát az „**a**” kockára tesszük

Ha választottunk egyet, akkor ismét újabb alternatívák közül választhatunk. A következő ábrán láthatjuk a probléma lehetséges állapotait (ezek a gráf csúcsai), valamint az irányított élek jelzik, hogy melyik állapotból melyiket kaphatjuk meg.



3. ábra.

Ahogy a példán keresztül is megfigyelhettük, egy adott probléma esetén definiálnunk kell a probléma állapotterét és azokat a lehetséges mozgatásokat, tevékenységeket (a későbbiekben ezeket operátoroknak fogjuk nevezni), amelyek a probléma egy állapotát egy másikba viszik át. Az állapotok közül ki kell még jelölni a kezdőállapotot (a gráfban a startcsúcs), valamint az(oka)t az állapot(oka)t, amely(eke)t meg szeretnénk kapni a kezdőállapotból. Ez(eke)t célállapot(ok)nak nevezzük (a gráfban a terminális csúcs(ok)). Előfordulhat, hogy egy operátor alkalmazásának költsége is lesz: a példában, pl. ha az egyik kockát nehezebb elmozdítani, mint a másikat.

Ha az operátorok alkalmazásának vannak költségei, akkor feladatunk lehet például a legkisebb összköltségű, a startcsúcsból terminális csúcsba vezető út megkeresése a gráfban. Kereshetjük a legrövidebb utat is.

4.2. Állapottér-reprezentáció Prologban

Egy X és Y állapot esetén, ha X állapotban van olyan alkalmazható operátor, hogy segítségével Y-t kapunk, akkor az állapottér-gráfban található él a két állapotot reprezentáló csúcs között. Ekkor azt mondjuk, hogy Y gyermeke X-nek.

Ennek megfelelően $el(X,Y)$ igaz, ha Y és X között húzódik él a gráfban. Ha a költséget is figyelembe kell venni, akkor a predikátumszimbólum háromargumentumú lesz: $el(X,Y,koltseg)$.

Ezt a relációt a programunkban reprezentálhatjuk tények halmazaként is, de mivel az állapottér-reprezentációnk tetszőleges bonyolultságú lehet, ez sokszor nem praktikus, vagy nem is lehetséges. Ezért az el relációt implicit módón is definiálhatjuk úgy, hogy egy adott csúcsra alkalmazzuk az alkalmazható operátorainkat.

A **példá**nkban: korlátozott számú kockánk és helyünk van, ami ésszerű megszorítás is, mivel a robotunknak korlátozott helye van csak az asztalon, amin pakolgat. Jelen pillanatban 3 kockánk van, és asztalon 3 kockányi hely van. Egy lehetséges reprezentáció: vermek listája. A verem is lényegében lista, a lista feje a verem teteje. Egy-egy verem az egymáson lévő kockákat reprezentálja.

- A startcsúcs:

  $[[c,a,b],[],[]]$

- A terminális csúcsok: az általuk reprezentált állapotban egy sorba rendezett kockaoszlopnak kell léteznie, de ez az asztalon bárhol elhelyezkedhet:

  $[[a,b,c],[],[]]$
  $[[],[a,b,c],[]]$
  $[[],[],[a,b,c]]$

- Az el reláció a következő szabály szerint programozható: $Csucs1$ és $Cucs2$ között van él, ha van két olyan verem a $Csucs1$-ben: $Verem1$ és $Verem2$, és a $Verem1$ tetején lévő kocka áthelyezhető $Verem2$-be. Az áthelyezéssel nyert állapotot fogja reprezentálni a $Csucs2$. Az összes állapotot generálhatjuk Prolog programmal, amely során vermek listáit fogjuk megkapni:

```
el(Vermek, [Verem1, [Teteje1 | Verem2]|TobbiVerem]) :-
                                        %Teteje1 – et Verem2 – be
    torol([Teteje1|Verem1], Vermek, Vermek1]),   % az 1. verem keresése
    torol(Verem2, Vermek1, TobbiVerem).          % a 2. verem keresése

torol(X, [X|L], L).
torol(X, [Y|L], [Y, L1]) :-
    torol(X, L, L1).
```

- Terminális csúcsok meghatározása:

$cel(Csucs) :-$
$\quad eleme([a,b,c], Csucs).$

- A keresőalgoritmust leírhatjuk a következőképpen:

  $megold(Start, Megoldas)$

Itt $Start$ a kezdőállapotot reprezentáló startcsúcs az állapottér-gráfban, a $Megoldas$ pedig egy út $a\ Start$ és egy terminális csúcs között. A példában:

$megold([[c,a,b],[],[]], Megoldas).$

Sikeres keresés esetén a $Megoldas$ tartalma: kocka-rendezések listája. Ez a lista reprezentálja azt, hogy a kezdőállapotból milyen állapotokon keresztül, milyen lépésekkel juthatunk el a rendezett célállapotunkba ($[a,b,c]$).

4.3. A mélységi kereső

Ha adott egy probléma állapottér-gráffal reprezentálva, sokféle lehetőség közül választhatunk, hogy megtaláljuk azt az utat, amely elvezet minket a megoldáshoz. Az egyik ilyen stratégia a mélységi kereséssel való megoldás.

Abból indulunk ki, hogy hogyan tudunk egy adott N csúcsból egy terminális csúcsba vezető utat (Ut) találni úgy, hogy

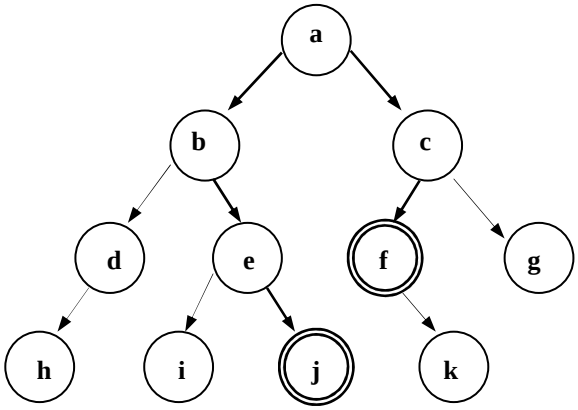
- ha N terminális csúcs, akkor $Ut = [N]$, vagy
- ha N-ből vezet él egy $N1$ csúcsba, és $N1$-nek ismerjük egy célcsúcsba vezető útját, $Ut1$-et , akkor $Ut = [N|Ut1]$.

Prologban:

$megold(\ N, [\ N\ ]) :-$
$\quad cel\ (\ N\ ).$

$megold(\ N, [\ N1\ |\ Ut1\ ]\ ) :-$
$\quad el(\ N, N1),$
$\quad megold(\ N1, Ut1).$

Ez a program valójában a mélységi kereső implementációja. A keresőt „mélységinek” az állapottér-gráf csúcsainak előállítási sorrendje miatt hívják. A lehetséges csúcsok közül mindig a legnagyobb mélységi számút választja kiterjesztésre. A legnagyobb mélységi számú az, amelyik a legtávolabb helyezkedik el a startcsúcstól.



4. ábra.

A fenti ábra egy egyszerű állapottér-gráf: „**a**” a startcsúcs, „**f**” és „**j**” pedig terminális csúcsok. A mélységi kereső a következő sorrendben járja be a csúcsokat: **a,b,d,h,e,i,j**. A megtalált megoldás: $[a, b, e, j]$.

A megoldás megtalálásához a következő kérdést kell feltennünk a rendszernek:

$?- megold(a, Megoldas).$

A mélységi kereső a legalkalmasabb arra, hogy rekurzív módon programozzunk Prologban. Ennek az oka a Prolog maga, hiszen a megoldást keresve az alternatívák vizsgálatát mélységi kereső módjára végzi.

Példa. A nyolc királynő problémája:

- a csúcsok egy-egy táblaállapotot reprezentálnak 0 vagy több királynővel, amelyek a tábla egymást követő oszlopaiban helyezkednek el
- a rákövetkező csúcsot úgy kaphatjuk meg, ha a következő üres oszlopban egy királynőt helyezünk el úgy, hogy az ne üsse az eddig elhelyezett királynőket
- a startcsúcs: egy üres tábla, egy üres listával reprezentálva
- terminális csúcs: egy olyan tábla, amelyen 8 királynő van (az előbbiekben már kikötöttük, hogy ezek nem üthetik egymást)

A csúcsokat egy listával fogjuk leírni, amely az eddig felhelyezett királynők Y koordinátáit tartalmazza:

$el(\ Kiralynok, [\ Kiralyno \mid Kiralynok\ ]\ ) :-$
$\quad eleme(\ Kiralyno, [\ 1,2,3,4,5,6,7,8]\ ),$
$\quad nemtamad(\ Kiralyno, Kiralynok).$

$cel(\ [\_,\_,\_,\_,\_,\_,\_,\_]).$

$nemtamad(\_, [\ ]\ ).$

$nemtamad(X/Y, [[X1/Y1|Tobbi]]) :-$
$\quad Y =/= Y1,$
$\quad Y1 - Y =/= X1 - X,$
$\quad Y1 - Y =/= X - X1,$
$\quad nemtamad(X/Y, Tobbi).$

A kérdés:

$?- megold(\ [], Megoldas\ ).$

A $Megoldas$: egy lista, amely a helyesen elhelyezett 8 királynő Y koordinátáit tartalmazza (X koordináták növekvő sorrendjében).

A mélységi keresőnél problémát okozhat, ha kör van a gráfban (nem fog visszalépni, hanem elkezd „körözni” az algoritmus). Egyik lehetséges megoldás a körfigyelés: figyeljük, hogy a startcsúcsból az aktuális csúcsba vezető úton lévő csúcsok között nem szerepel-e már az aktuális csúcs:

$melysegi(Ut, Csucs, Megoldas)$, ahol:

$Csucs$: az a csúcs, ahonnan éppen keressük az aktuális utat egy terminális csúcsba. Ut: út (csúcsok listája) a startcsúcsból a $Csucs$-ba, $Megoldas$: út a $Csucs$ és egy terminális csúcs között. Az egyszerűség kedvéért az Ut-ban fordított sorrendben fognak szerepelni a csúcsok.
A kereső körfigyeléssel:

$megold(\ Csucs, Megoldas\ ) :-$
$\quad melysegi(\ [], Csucs, Megoldas).$

$melysegi(\ Ut, Csucs, [\ Csucs, Ut]) :-$
$\quad cel(\ Csucs).$

$melysegi(\ Ut, Csucs, Megoldas\ ) :-$
$\quad el(\ Csucs, Csucs1),$
$\quad not\ eleme(\ Csucs1, Ut\ ),$ $\%körfigyelés$
$\quad melysegi(\ [\ Csucs \mid Ut\ ], Csucs1, Megoldas\ ).$

Kis módosítás, hogy kevesebb argumentumunk legyen: a $Csucs$ és az Ut egy listába (P) kerülhet:

$P = [Csucs|Ut]$
$melysegi1(P, Megoldas).$

Probléma lehet még, ha az állapottér végtelen, és ekkor megtörténhet, hogy a kereső belefut egy végtelen ágba, és nem kerül közelebb a megoldáshoz, hanem egyre mélyebbre halad, így nem fogja megtalálni az esetleges jó megoldást. Ennek a kiküszöbölésére meghatározhatunk egy maximális mélységet, amelynél tovább nem megyünk:

$$melysegi2(Csucs, [Csucs], \_) :-$$
$$\quad cel(Csucs).$$

$$melysegi2(Csucs, [Csucs|Megoldas], MaxMelyseg) :-$$
$$\quad Maxmelyseg > 0,$$
$$\quad el(Csucs, Csucs1),$$
$$\quad Max1 \; is \; MaxMelyseg - 1,$$
$$\quad melysegi2(Csucs1, Megoldas, Max1).$$

4.4. A szélességi kereső

A mélységi keresővel ellentétben a szélességi kereső először azokat a csúcsokat találja meg, amelyek a legközelebb vannak a startcsúcshoz. Ebből következik, hogy az állapottér-gráfot is másféle sorrendben építi fel, és legelőször a legrövidebb utat találja meg.

A szélességi keresőt nem olyan egyszerű programozni, mint a mélységit, hiszen nemcsak egy csúcsra koncentrálunk egyszerre, hanem számon kell tartanunk a kiterjesztésre váró csúcsok halmazát is. Hogyha megoldásként a célhoz vezető utat is meg szeretnénk kapni, akkor nemcsak a csúcsokat, hanem a hozzájuk vezető utakat is tárolni kell. A későbbiekben ezt a halmazt a nyílt utak halmazának fogjuk nevezni (nyílt az a csúcs, amelyet még nem terjesztettünk ki, de már elértünk; nyílt az az út, amelynek a végén nyílt csúcs található).

A fentiek alapján a $szelessegi(Utak, Megoldas)$ akkor lesz igaz, ha az $Utak$ halmaz kiterjeszthető egy terminális csúcsra is.

A nyílt utak halmazát reprezentálhatjuk például listák halmazaként. Ekkor minden út csúcsoknak fordított sorrendű listája lesz, így a lista első eleme mindig a legutóbb elért csúcs, az utlosó elem pedig mindig a startcsúcs. A keresés egy egyelemű listával fog kezdődni: $[[StartCsucs]]$.

A megoldás menete a következő lesz:

- Ha az első út tartalmaz egy terminális csúcsot a lista fejében, akkor megkaptuk a megoldást, máskülönben:

- távolítsuk el az első utat a nyílt utak halmazából és generáljuk le az összes lehetséges egy lépéses kiterjesztését ennek az útnak, adjuk hozzá ezeket a nyílt utak halmazához, majd hívjuk meg a szélességi kereső eljárást az újonnan létrejött halmazára.

Példa.

(1) Kezdetben a halmazunk:

$[[a]]$

(2) Generáljuk $[a]$ kiterjesztéseit:

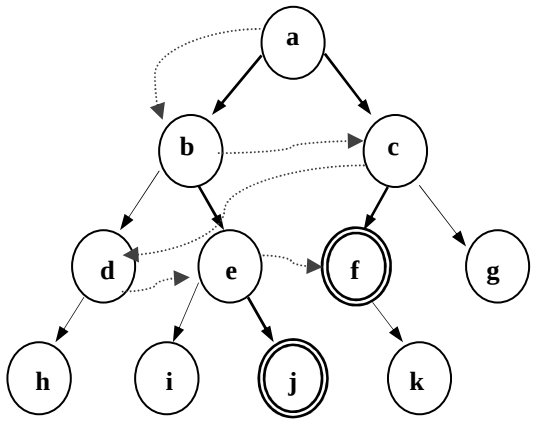
$[[b,a],[c,a]]$

(3) Távolítsuk el az első utat, $[b,a]$ a halmazból és generáljuk a kiterjesztéseit:

$[[d,b,a],[e,b,a]]$

Adjuk hozzá a fenti listákat a nyílt utak halmazához:

$[[c,a],[d,b,a],[e,b,a]]$



5. ábra.

(4) Távolítsuk el $[c,a]$-t és adjuk hozzá a kiterjesztésit a halmazhoz:

$[[d,b,a],[e,b,a],[f,c,a],[g,c,a]]$

A következő lépésekben $[d,b,a]$-t és $[e,b,a]$-t terjesztjük ki, miután a következő halmazt kapjuk:

$[[f,c,a],[g,c,a],[h,d,b,a],[i,e,b,a],[j,e,b,a]]$

Akkor a keresés eléri $[f,c,a]$-t, amely tartalmazza az f terminális csúcsot, így az eljárás visszatér a megoldással.

A program megvalósításakor használt függvények:

- $kiterjeszt$: Előállítja az összes olyan utat, amelyet egy lépéssel megkaphatunk az adott útból. Generáláskor figyel a körmentességre is, és ha már nem lehet tovább kiterjeszteni az adott utat, akkor meghívja a $szelessegi$-t.
- $eleme$: ugyanaz, mint a mélységi kereső példájánál megadott függvény
- $osszefuz$: két listát összefűz egy listává

A program:

```
megold( Kezdo, Megoldas ) :-
    szelessegi( [ [Kezdo] ], Megoldas ).

szelessegi( [ [ Csucs | Ut]|_], [ Csucs| Ut] ) :-
    cel ( Csucs ).
```

```
szelessegi( [ [N | Ut]| Utak], Megoldas ) :-
    kiterjeszt([M, N|Ut],
        (s(N, M), not eleme(M, [N|Ut])),
        UjUtak),                    %UjUtaK = körmentes változata [n|Ut] – nak
    osszefuz(Utak, ujUtak, Utak1), !,
                    szelessegi(Utak1, Megoldas);
szelessegi(Utak, Megoldas).         %abban az esetben, ha N – ből nincs több él

osszefuz([X|L1], L2, [X|L3] ) :-
    osszefuz(L1, L2, L3).
```

A szélességi kereső másik megvalósítása lehetne, ha nem utakat tárolunk, hanem az eddig bejárt fát, így elkerülhetnénk a csúcsok duplikációját, ám a program némileg bonyolultabbá válna.

A szélességi kereső mindig a legrövidebb utat találja meg (ha van megoldás), a mélységi nem mindig. Ha figyelembe kell vennünk az utak költségeit, akkor már a szélességi kereső sem fogja minden esetben az optimális megoldást megtalálni.

4.5. Best first: a heurisztikus kereső

Az előző részekben ismertetett megoldáskereső módszerekkel az a probléma, hogy sokszor nagyon sok utat bejár, mielőtt megtalálná a megoldást. Ha van a problémával kapcsolatban olyan tudásunk, melyet a keresés során fel lehetne használni, merre keressük a megoldást, és nem szisztematikusan szeretnénk az állapottér-gráfot előállítani, akkor vélhetőleg kisebb keresőfa előállítása lenne szükséges.

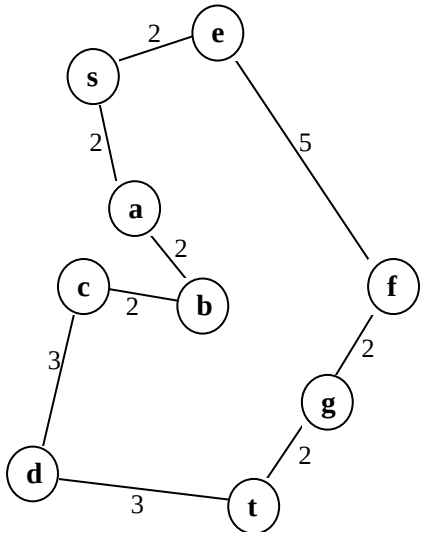
A tudásunknak megfelelő, számmal kifejezhető heurisztikus becslést rendelünk az állapottér-gráf csúcsaihoz. Ez a szám azt fogja kifejezni, hogy az adott csúcs mennyire ígéretes a megoldás megtalálása szempontjából. Ha tudunk egy heurisztikát rendelni az egyes állapotokhoz, akkor az azt jelenti, hogy lehetőségünk van mindig az akkor legjobbnak látszó irányba elindulni, és nem csak próbálgatunk.

A best-first kereső lényegében a szélességi kereső módosítása, hiszen ugyanúgy a startcsúccsal kezdünk, és számon tartjuk a nyílt utak halmazát. A különbség az, hogy míg a szélességi kereső mindig a legrövidebb nyílt utat (a legkisebb mélységi számút) választja kiterjesztésre, a best-first kiszámítja a heurisztikus becslést a nyílt csúcsokra, és az eszerinti legjobbat választja ki.

Legyen $k(n, n')$ költség az a költség, amely során n-ből n'-t kapjuk. A költségek a gráf éleihez vannak rendelve. Rendeljünk egy f függvény által kiszámított becslést az állapottér-gráf minden csúcsához. Egy n csúcshoz rendelt érték: $f(n)$. A legígéretesebb az a csúcs lesz, amely a legkisebb f értékkel rendelkezik. Az $f(n)$ becslést úgy határozzuk meg, az az s startcsúcsból az n csúcson keresztülvezető és célcsúcsig tartó legjobb út becsült költsége legyen. $f(n)$-t a következő képlettel határozhatjuk meg: $f(n) = g(n) + h(n)$, ahol $g(n)$ a s-ből n-be vezető optimális út már kiszámított költsége, $h(n)$ pedig egy heurisztikus érték az n-ből t terminális csúcsba vezető út optimális költségére.

Amikor egy n csúcsot elértünk a keresés során, a következő lesz a szituáció: az s-ből n-be vezető út már ismert, a $g(n)$ költséget az élek (amelyek az úton vannak) költségeinek összköltségeként kapjuk meg. Az az út nem feltétlenül az s-ből n-be vezető optimális út (lehet, hogy van jobb is, csak még nem találtuk meg), de ez a $g(n)$ költség az eddig ismert legkisebb költség s és n között. A másik termnek, $h(n)$ értékének a kiszámítása már problémásabb, mivel a világunk n-t t-vel összekötő részét még nem fedeztük fel ezidáig. Tehát $h(n)$ egy heurisztikus becslés lesz, amely a világunkról ismert általános ismereteken alapszik, így nincsenek általános kiszámítási szabályok h-ra, hanem mindig az adott problémától függenek.

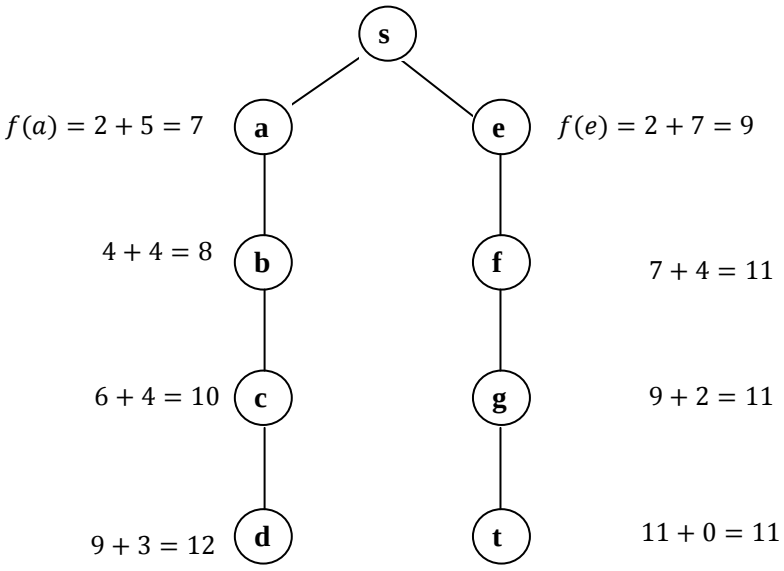
A best-first algoritmust a következő módon tudjuk elképzelni: a folyamat néhány konkurens részfolyamatot tartalmaz, mindegyik keresi a saját alternatíváit, vagyis építi a saját részfáját. A részfák újabb részfákat tartalmaznak, amelyeket a részfolyamatok újabb részfolyamatai építenek, stb. A konkurens részfolyamatok közül egyszerre mindig csak egy aktív, az, amelyik az éppen legígéretesebb, azaz a legkisebb f-értékkel rendelkezik. A többi folyamatnak addig kell várakozni, amíg az aktuális f-érték megváltozik úgy, hogy már nem az lesz a legkisebb. Ekkor megint meg kell keresni a legkisebb f-értékkel rendelkező alternatívát, és aktívvá tenni. A folyamat futás közben bővíti a részfáját, és visszaadja a megoldás, ha megtalált egy terminális csúcsot.



6. ábra.

A fenti ábrán látható egy térkép, a feladat az, hogy megtaláljuk a legrövidebb utat **s** és **t** város között. Hogy megbecsüljük a fennmaradó útköltséget X városból **t**-be, egy egyszerű távolságfüggvényt, a $tav(X, \mathbf{t})$-t fogjuk használni. Így:

$$f(X) = g(X) + h(X) = g(X) + tav(X, \mathbf{t})$$



7. ábra.

Ebben a példában a best-first kereső 2 folyamatot tartalmaz: mindkettő egy-egy részfát épít, *Folyamat1* az a csúcson keresztül, *Folyamat2* pedig e-n keresztül. Kezdetben *Folyamat1* aktív, mert az ő f-értéke kisebb, mint a másiké. Az első lépés után *Folyamat1* c-ben, *Folyamat2* e-ben van, az értékek pedig a következőképpen alakulnak:

$f(\mathbf{c}) = g(\mathbf{c}) + h(\mathbf{c}) = 6 + 4 = 10$

$f(\mathbf{e}) = g(\mathbf{e}) + h(\mathbf{e}) = 2 + 7 = 9$

Ekkor $f(\mathbf{e}) < f(\mathbf{c})$, ezért most *Folyamat2* lesz aktív, *Folyamat1* pedig várakozik. A következő lépések:

$f(\mathbf{f}) = 7 + 4 = 11$
$f(\mathbf{c}) = 10$
$f(\mathbf{c}) < f(\mathbf{f})$

Ezek után *Folyamat2* megáll, *Folyamat1* következik, de rögtön a következő a **d** lesz, és mivel$f(\mathbf{d}) = 12 > 11$, megint a *Folyamat2* lép működésbe, és eljut egészen a **t** célig.

A program megvalósítása: a szélességi kereső módosítása lesz, a nyílt utakat fában fogjuk tárolni. A programban a fák két termmel lesznek reprezentálva:

(1) $l(N, F/G)$ reprezentál egy levelet a fában, N az állapottér-gráf egy csúcsa, G a $g(N)$ érték, F pedig az $f(N) = G + h(N)$ érték.
(2) $t(N, F/G, Reszfak)$ reprezentál egy fát nemüres részfákkal, N a fa gyökere, $Reszfak$ a részfák listája, G a $g(N)$ érték, F pedig frissített f-értéke az N csúcsnak, ami az N gyermekei f-értékének a minimumát jelenti. A keresés könnyítése végett a $Reszfak$ az f-értékük szerinti növekvő sorban vannak rendezve.

Az f-értékek frissítése azért szükséges, hogy felismerhessük a legígéretesebb részfát a fa minden szintjén a keresés közben. Ezzel a módosítással általánosítani tudjuk f definícióját úgy, hogy nemcsak csúcsokra, hanem fákra is vonatkoztat hatjuk:

- a fa egy egyszerű csúcsára (levelére): $f(n) = g(n) + h(n)$
- egy n gyökerű T fára, ahol $m_1, \dots, m_k$ n gyermekei: $f(T) = \min_{1 \le i \le k}(f(m_i))$

A program:

```
bestfirst( Kezdo, Megoldas) :-
    legnagyobb( Nagy),                    %Nagy > akármelyik f érték
    kiterjeszt( [], l(Start, 0/0), Nagy, _, igen, Megoldas).

kiterjeszt( P, l(N, _), _, _, igen, [N|P]):-
    cel(N).
```

```
kiterjeszt( P,l(N,F/G),Hatar,Fa1,Megoldott,Megold) :-
    F <= Hatar,
    (mh( M/K,(el(N,M,K),not eleme(M,P)),Utod),!,
        utodlista( G,Utod,Ts),
        legjobbf( Ts,F1),
        kiterjeszt( P,t(N,F1/G,Ts),Hatar,Fa1,Megoldott,Megold) ;
    Megoldott = soha).                    %nincs több gyermek csúcs

kiterjeszt( P,t(N,F/G,[T|Ts]),Hatar,Fa1,Megoldott,Megold) :-
    F <= Hatar,
    legjobbf( Ts,BF),min( Hatar,BF,Hatar1),
    kiterjeszt( [N|P],T,Hatar1,T1,Megoldott1,Megold),
    folytat( P,t(N,F/G,[T1|Ts ]),Hatar,Fa1,Megoldott1,Megoldott,Megold).

kiterjeszt(_,t(_,_,[]),_,_,soha,_) :- !       %ez a fa nem lesz benne a megoldásban

kiterjeszt( _,Fa,Hatar,Fa,nem,_) :-
    f (Fa,F),F > hatar.                   %nem tud tovább nőni,mert elértünk a határt

folytat(_,_,_,_,igen,igen,Megold).

folytat( P,t( N,F/G,[T1|Ts]),Hatar,Fa1,Megoldott1,Megoldott,Megold :-
    ( Megoldott1 = nem,beszur( T1,Ts,Nts);
     Megoldott1 = soha,NTs = Ts),
    legjobbf( NTs,F1),
    kiterjeszt( P,t( N,F1/G,NTs),Hatar,Fa1,Megoldott,Megoldas).

utodlista( _,[],[]).

utodlista( G0,[N/C|NCs],Ts) :-
    G is G0 + C,
    h( N,H),                              %a h(N)heurisztikus érték H – ba
    F is G + H,
    utodlista( G0,NCs,Ts1),
    beszur( l(N,F/G),Ts1,Ts).

beszur( T,Ts,[T|Ts]) :-
    f(T,F),legjobbf( Ts,F1),
    F <= F1,!.

beszur( T,[T1,Ts],[t1,Ts1]) :-
    beszur( T,Ts,Ts1).

f(l(F/_),F).                              % a levél f – értéke
f(t(_,F/_,_),F).                          %a fa f – értéke

legjobbf([T|_],F) :-                      %az f – értékek közül a legjobb
    f(T,F).

legjobbf([],Nagy) :-
    legnagyobb( Nagy).
```

$min(X, Y, X) :-$
$\quad X <= Y, !.$

$min(X, Y, Y).$

A program kulcseljárása a $kiterjeszt$, aminek hat argumentuma van:

$$kiterjeszt(P, Fa, Hatar, Fa1, Megoldott, Megoldas)$$

Az eljárás addig építi az aktuális részfát, amíg az f-értéke kisebb vagy egyenlő nem lesz a $Hatar$-nál. Az argumentumok jelentése:

- P : Út a startcsúcs és a Fa között.
- Fa : Az aktuális részfa.
- $Hatar$: A Fa kiterjesztésének f-érték maximuma.
- $Fa1$: A Fa-t a $Hatar$ éréken belül terjesztettük ki, tehát $Fa1$ f-értéke már nagyobb, mint a $Hatar$ (kivéve, ha a kiterjesztés közben már megtaláltuk a célt).
- $Megoldott$: Egy olyan jelző, amelynek értéke „$igen$”, „nem”, vagy „$soha$” lehet.
- $Megoldas$: A megoldási út a startcsúcsból a $Fa1$-en keresztül a célcsúcsig, f-értéke a $Hatar$-on belül (kivéve, ha megtaláltuk a célt).

P, Fa és $Hatar$ a kiterjeszt eljárás bemenetei, a $Megoldott$, $Megoldas$ és $Fa1$ pedig kimeneti paraméterek, amelyek a következő értékeket kaphatják:

(1) $Megoldott = igen$
$Megoldas$ = a Fa $Hatar$-on belüli kiterjesztése által kapott megoldási út
$Fa1$ = definiálatlan

(2) $Megoldott = nem$
$Fa1$ = a Fa olyan kiterjesztése, amelynek f-értéke meghaladja a $Hatar$-t
$Megoldas$ = definiálatlan

(3) $Megoldott = soha$
$Fa1$ és $Megoldas$ = definiálatlan

Az utolsó eset akkor következik be, ha már biztos, hogy ez a részfa nem lesz a megoldás része, így soha többet nem válik aktívvá. Ez az eset akkor áll fenn, ha a Fa f-értéke még

kisebb vagy egyenlő, mint $Hatar$, de nem tud tovább növekedni, mert a leveleinek nincsenek gyermekei, vagy ha vannak is, kiterjesztésük által már kört kapnánk.

A következő klóz, amivel foglalkozunk, a Fa, amelynek részfái lehetnek:

$$Fa = t(N, F/G, [T|Ts])$$

Először a legelső részfa, a T kerül kiterjesztésre. Ez a részfa még nem rendelkezik a $Hatar$ határértékkel, de feltehetőleg egy kisebb f-érték lesz, amely a többi részfa, Ts f-értékeitől függ. Ez biztosítja, hogy az éppen növekvő részfa a legígéretesebb részfa. A $kiterjeszt$ eljárás a részfák közül azok f-értékei szerint válogat. Miután megtalálta a jelöltet, egy kisegítő eljárás, a $folytat$ határozza meg, hogy ezután mi következik. Ha a megoldást megtaláltuk, akkor a program visszatér a megoldással, máskülönben folytatódik a kiterjesztés. A $Fa = l(N, F/G)$ generálja az N csúcs gyermekeit és előállításuknak költségeit. Az $utodlista$ eljárás egy listát készít ezen gyermekek részfáiról és kiszámítja a g-értékeiket és f-értékeiket. További programmagyarázat:

- $el(N, M, C)$: Az állapottér-gráfban N egy gyermeke M, és C az N és M között lévő él költsége.
- $h(N, H)$: H heurisztikus becslés az N csúcsból a célig vezető út költségére.
- $legnagyobb(Nagy)$: a $Nagy$ egy, a felhasználó által definiált olyan érték, amely biztosan nagyobb lesz az összes lehetséges f-értéknél.

Ebben a részben a best-first algoritmus egy elég gyakran használt, A^* algoritmusnak nevezett változatával foglalkoztunk. Most pontosítanánk az algoritmusra vonatkozó megszorításokat.

Egy kereső-algoritmust **elfogadható**nak nevezünk, ha mindig az optimális megoldást állítja elő, amennyiben a megoldás létezik. Jelölje $h^*(n)$ az optimális költségű n csúcsból a terminális csúcsig vezető út költségét az állapottér-gráf minden n csúcsára. Az A^* algoritmus elfogadhatóságára vonatkozó tétel azt mondja ki, hogy ha az állapottér-gráf csúcsaira alkalmazott h függvény: $h(n) \leq h^*(n)$ minden n csúcsra, akkor az algoritmus elfogadható.

A fenti tételnek gyakorlati jelentősége van. Ha még nem is ismerjük a pontos h^* értéket, elég, ha alulról becsüljük azt, és azt használjuk h-ként az A^* algoritmusban. Ez elégséges feltétel ahhoz, hogy az eljárás az optimális megoldást állítsa elő. Egy triviális eset lehet, amikor

$h(n) = 0$ minden n-re. Már ez is garantálja az elfogadhatóságot, azonban az lesz a hátránya, hogy nem fog adni a kereséshez semmi támpontot, így hasonlóképpen fog működni az algoritmus, mint a szélességi kereső, ahol minden él költsége 1. Mi olyan h függvényt szeretnénk, amelynek értékei kisebbek lesznek h^*-nál, de olyan közel vannak hozzá, amennyire csak lehetséges. Ideális esetben pontosan ismerjük h^*-ot, de ekkor valójában az optimális megoldást nem kell keresni.

Összegzés

Szakdolgozatomat összefoglalva: mielőtt a Prologgal kezdtem volna foglalkozni, elsajátítottam a matematikai logika alapjait, a logikai kalkulusok általános jellemzőit és egy automatikus tételbizonyítási módszert, az SLD-rezolúciós elvet. A logikai programozási nyelvek ezt a kalkulust implementálják, így mindenképpen ismernünk kell, hogy megértsük működését, valamint helyes és hatékony programot tudjunk írni.

Ezek után megismerkedtem a Prolog történetével, alapvető építőköveivel – a szintaxissal és ezek szemantikájával.

Az állapottérrel reprezentált mesterséges intelligencia feladatok néhány alapvető megoldáskereső stratégiájának (mélységi, szélességi keresés, best-first algoritmus) megvalósítását vizsgáltam Prolgoban. További feladat lehet még a többi módszer implementálása és összehasonlítása az előzőekkel, valamint a mesterséges intelligencia többi területének, - pl. kétszemélyes játékok, szakértő rendszerek, stb. – vizsgálata.

Irodalomjegyzék

[1] Ivan Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley Publishing Company, Great Britain, cop. 1986.

[2] Ulf Nilsson and Jan Maluszyński. *Logic, Programming and Prolog* (2. Edition). Wiley, 2000.

[3] Pásztorné Varga Katalin és Várterész Magda. *A matematikai logika alkalmazásszemléletű tárgyalása*. Panem, Budapest, 2003.

[4] Ruzsa Imre – Máté András. *Bevezetés a modern logikába*. Oziris Kiadó, Budapest, 1997.

Internetes forrás:

[5] http://www.lpa.co.uk/