A Prolog logikai alapjai, az SLD-rezolúció

Szabó Péter
<pts+prologikai@math.bme.hu>

Válogatott fejezetek a logikai programozásból
kiselőadás
2004. október

⁜ Bevezetés ⁜

- Az *imperatív program* egy recepthez hasonló: meghatározza, hogy mit és milyen sorrendben kell megtenni a programot futtató gépnek. A *logikai program* viszont egy tudást ír le, melyet a gép a neki feltett kérdésekre való válaszoláshoz felhasznál. A végrehajtás sorrendje ilyenkor nem lényeges. Prolog esetén ugyan ismert ez a sorrend, és a program írásakor ezt a gyakorlatban ki is használjuk, de egy általanos logikai programnál ez mellékes: a program állítások gyűjteménye, melyek alkalmazási sorrendjéről a kérdés alapján a futtatókörnyezet dönt. (Prolog esetén az állítás *tény* vagy *szabály.*)
- A *következtetési szabály* fogalmat a továbbiakban szándékosan nem használjuk. Nem keverendő fogalmak: *állítás* (a logikai program része), *levezetési szabály* (ezt alkalmazva a futtatókörnyezet újabb állításokhoz jut a már meglevőekből) és *implikáció* (a $\rightarrow$ művelet az elsőrendű logikában).
- A továbbiakban csak *tiszta* Prolog-programokat (*pure Prolog*) engedünk meg: a meta-logikai eljárások (pl. var/1, findall/3, setof/3, freeze/2) használatát, a vágót, a feltételes szerkezetet (`A -> B ; C`) és a negálást megtiltjuk. Nem szabad továbbá végtelen struktúrákat létrehozni (pl. `X = f(X,Y)`). Az egyszerűség kedvéért a diszjunkciót (';'/2) mellőzzük: ez az eljárások szétdarabolásával könnyen kiváltható.
- Vegyük észre, hogy ha egy ilyen Prolog-program nem tartalmaz se végtelen választási pontot, se végtelen ciklust (minden hívás véges időben lefut), akkor egy Prolog klózon belül szabadon cserélgethetjük a célok sorrendjét, ez nem befolyásolja a kérdésre adott választ. Ekkor egy „`P :- Q, R, S.`" alakú Prolog klóz egy implikációnak felel meg: ha Q, R és S igaz, akkor P is. Formálisan: $P \leftarrow (Q \wedge R \wedge S)$. Megjegyzés: a Mercury nevű, Prolog-szerű nyelv fordítóprogramja valóban szabadon cserélgeti a célokat, és olyan kódot fordít, melynek végrehajtása a leghatékonyabb.
- A továbbiakban arról lesz szó, hogy milyen tudás írható le a klózokkal, vagyis a (megengedett) Prolog klózokkal ekvivalens implikációkkal; továbbá milyen kérdésekre és hogyan lehet válaszolni az így leírt tudás alapján.

⁜ Elsőrendű logika, ismétlés ⁜

- Azt szeretnénk, hogy a gép gondolkozzon, vagyis a már ismert dolgok alapján helyes következtetésekre jusson és helyes döntéseket hozzon. (Ez azért jó, mert ekkor bizonyos, gondolkodást is igénylő emberi munkakörök kiválthatók számítógéppel.) A gondolkodás szűkebb értelemben azt jelenti, hogy az ismert és igaznak elfogadott állítások alapján egy állításról a gép eldöntse, hogy igaz vagy sem. Egy állítást akkor nevezünk igaznak, ha eljuthatunk hozzá az ismert dolgokból helyes logikai lépések egymásutánjával. A gép akkor döntötte el egy B állításról, hogy igaz-e, ha vagy B-hez, vagy az ellentétéhez ($\neg B$-hez) így eljutott (és a köztes lépéseket kérésre be is tudja mutatni). (A „gondolkodás" során persze sok egyéb állítás is előkerülhetett, melyeket a gép végül nem használt fel a végeredményt adó logikai lépéssorozatban.)
- Ha egy $\nexists X\, P$ alakú állítás hamisnak bizonyul, akkor kíváncsiak vagyunk egy ellenpéldára (vagyis mennyi az X?). Esetleg az összes ellenpéldára is kíváncsiak vagyunk. A Prolog működése felfogható úgy, hogy a B állítás (a *kérdés*, *query*) negáltjáról próbálja meg bebizonyítani, hogy hamis, és a negáltra adott ellenpélda (változóbehelyettesítések) a B igaz voltát bizonyítja.
- A gép nem szabadon, korlátlanul gondolkozik, mert az ismert dolgokat egy P logikai programban pontosan meghatároztuk a számára, továbbá a bizonyítandó vagy cáfolandó B állítást is megadtuk neki. Általában a P programot a programozó, az B állítást pedig a felhasználó fogalmazza meg. Gondoljunk egy orvosi diagnosztikai rendszerre: a P program az orvosi lexikonok és néhány szakorvos tudásának óriási gyűjteménye, az B állítás pedig ilyen alakú: „ha X.Y.-on ezt és ezt figyeltük meg, akkor betegsége Z", vagyis pontosabban: „létezik Z, hogy ha x.y.-on ezt és ezt figyeltük meg, akkor betegsége Z". Bizonyítandó B, és keresendő a Z változó összes behelyettesítése.
- Mi legyen az a nyelv, melyben a logikai programot megírjuk és a kérdést feltesszük? Mindenképpen egy *formális nyelv* legyen, hogy a természetes nyelvi mondatok többértelműségét és szövegkörnyezet-függőségét elkerülhessük. Ajánlatos egy olyan nyelvet választani, melynek tulajdonságai jól ismertek. A matematika logika ága a 20-adik században bőségesen foglalkozott számos ilyen nyelvvel. A legegyszerűbb közismert nyelv, a *nulladrendű logika* vagy *propozicionális logika* nem elég kifejező (pl. hiányoznak belőle a *kvantorok:* $\exists\ \forall$). Az elsőrendű logika viszont már sikerekkel kecsegtet. (Manapság egyéb, bonyolultabb logikákat is használnak.)
- Az elsőrendű logikában megengedett logikai szimbólumok: $\neg\ \wedge\ \vee\ \exists\ \forall\ \rightarrow\ \leftrightarrow$. Ezek jelentése a szokásos (ismert az Analízis, Bsz, Matematikai logika és Formális módszerek tárgyakból). E szimbólumokon kívül egy *formulát* (logikai kifejezést) *változók, konstansjelek, függvényjelek* és *predikátumjelek* alkotnak. Előre rögzítünk egy nyelvet, melyben a használható függvényjelek (aritással), predikátumjelek (aritással) előre adottak. A változónevek lényegtelenek (csak megkülönböztetésre szolgálnak), a konstansok pedig a 0 aritású (tehát argumentum nélküli) függvényjelek. Az építési szabályok a következők: Ún. *term* építhető konstansokból, változókból és függvény-alkalmazásból, például $\mathrm{apja}(\mathrm{felesége}(\mathrm{ernő}, X))$ egy term (jelentése *lehet*: Ernő X-edik házasságában szerzett apósa). Egy *atomi formula* (más néven *literál*) egyetlen predikátumból áll (az argumentumok termek), az egyéb formulák pedig atomiakból építhetők a fent felsorolt logikai szimbólumokkal. A *logikai program* (P) formulák egy véges halmaza.
- Elsőrendű logikában nem lehet leírni vélekedéseket (pl. X azt gondolja, hogy Y nőtlen; a Prolog ugyan megenged metapredikátumokat, pl. `gondolja(X, nőtlen(Y))`, de nincsenek beépített levezetési szabályok, melyek ezeket kezelnék), időbeli kijelentéseket (pl. ha X ma özvegy, akkor holnap is özvegy lesz), magasabbrendű (értsd: függvényekre vonatkozó) állításokat (pl. minden differenciálható függvény folytonos) és – szerencsétlen predikátumválasztás mellett – lezártakra vonatkozó állításokat (pl. ha egy kétváltozós predikátummal adott egy G gráf szomszédossági mátrixa, nem fogalmazható meg ez az állítás egyetlen logikai formulában: U-ból V-be van út; persze definiálható `van_út(U,V)` predikátum). A fenti hiányosságok ellenére is az elsőrendű logikát választjuk, mert már jól ismert, és sok célterületen elég kifejező.
- Mitől függ, hogy egy formula igaz-e? Ez függ a formula *jelentésétől* (*szemantikájától*), tehát az egyes függvényjelek és predikátumjelek jelentésétől is. Ezeknek általában van egy „szokásos" jelentése, tehát „ernő" egy konkrét, Ernő nevű embert jelöl, „apja(ernő)" Ernő apját, „szakmája(apja(ernő), szabó) (ahol szakmája/2 egy predikátum) pedig azt, hogy Ernő apja szabó. Persze elképzelhető más jelentés, a jelek másfajta interpretációja is, például „apja(X)" az X szám köbgyökét, „ernő" a 42 számot, „szakmája" pedig a nagyobbegyenlő relációt is jelentheti. A gép nem ismeri az interpretációt (ez csak a programozó és a felhasználó fejében létezik), ezért minden következtetés, amit a gép – formálisan – levon, érvényes *bármely* interpretációban, tehát a felhasználó fejében levő, ún. *szándékolt interpretációban* is.
- A formula igazsága függ továbbá a szabad, tehát a *kvantorok* ($\forall$ és $\exists$) hatályán kívül eső változók értékétől. Például a $\exists X\, X < Y$ formula igazsága függ Y értékől (például a természetes számok nyelvén, $Y = 0$ esetén hamis, $Y = 42$ esetén pedig igaz), de X-nek a formulán kívüli értékétől nem függ! A változók értéke bármi lehet, nem csak a nyelvbeli konstansok! (Például Y lehet 42 még akkor is, ha a formulákban és a nyelvben csak a 0 és 1 számoknak megfelelő konstansjel szerepel.) Egy formula *zárt,* ha nincs benne szabad változó.
- Egy adott interpretáció és változó-kiértékelés mellett egy formula igazsága a szokásos. Például $\forall Y \exists X$ $\mathrm{szakmája}(\mathrm{apja}(X), Y)$ pontosan akkor igaz, ha a vizsgált populációban és szakmák körében minden szakmát űz egy-egy apa. Ez vagy igaz, vagy nem, de mindenesetre a a vizsgált világ teljeskörű ismeretét feltételezve egyértelmű, mi a helyzet.

- Egy interpretációt akkor nevezünk a P formulahalmaz *modelljének,* ha az adott interpretációban minden P-beli formula igaz. A B állítás *logikai következménye* egy P-beli formulahalmaznak ($P \models B$), ha P minden modelljében igaz B is. Tehát ha minden világban, amelyben igazak a P-ben foglaltak, igaz B is, akkor (és csak akkor) B következménye P-nek.
- A programozási környezetünk tehát a következő: a P logikai program állításai zárt formulák, melyek a világra vonatkozó tudást tartalmazzák. Például egy konzervatív világról készülő logikai programba bekerülhet a $\neg\exists X\exists Y\exists Z$ apja$(X) = Z \wedge$ apja(anyja$(X)) = Z$ állítás, míg egy liberálisabb világban ez nem igaz.
- A bizonyítandó B állítás is zárt formula P-ben. A gép feladata megtalálni a bizonyítást, ami a P-ben található formulákból elemi logikai lépésekkel eljut B-ig. Ezt általában indirekt bizonyítással végzi, vagyis felteszi, hogy P összes formulája és $\neg B$ is igaz, és ebből ellentmondásra jut. Az ellentmondás egy olyan (egyszerű) zárt formula, ami minden interpretációban hamis. Ilyen például bármely $F \wedge \neg F$ alakú formula, ahol F formula. Megalállapodás szerint a $\square$ is egy azonosan hamis formula, $\blacksquare$ pedig azonosan igaz.
- Hogyan „gondolkodik" a gép? Rögzítve van néhány *levezetési szabály*, és ezeket próbálja alkalmazni a már ismert, igaz, zárt formulák némelyikére. A szabály egy új formulát eredményez. Direkt bizonyítás esetén ha az új formula megegyezik B-vel, akkor kész a bizonyítás, ellenkező esetben felveszi a már ismert formulák közé, és folytatja egy újabb levezetési szabály-alkalmazással. Az SLD-rezolúcióban is használt indirekt bizonyításban a leállási feltétel az azonosan hamis formulához való érkezés.
- Híres levezetési szabály a *modus ponens*: ha F és $F \rightarrow G$ már ismert, igaz, zárt formulák, akkor G is igaz. Híres még a *példányosítás*: ha $\forall X F(X)$ ismert, igaz, zárt formula és t egy term, ami nem tartalmazza X-et, akkor $F(t)$ is igaz. Híres még a *rezolúció alaplépése*: ha $F \vee G$ és $\neg F \vee H$ ismert, igaz, zárt formulák, akkor $G \vee H$ is igaz. Számtalan levezetési szabály felsorolható még, és mindegyikről könnyű ellenőrizni, hogy *helyes* (*sound*), vagyis alkalmazásával igaz formulákból igaz formulákat kapunk, bármilyen modellben.
- Fontos kérdés még, hogy levezetési szabályok egy készlete teljes-e, vagyis hogy minden P-ből következő B állítás be is bizonyítható velük. Például a *modus ponens* önmagában nem teljes, de ha a *példányosítást* és még néhány egyszerű szabályt hozzáveszünk, már teljes lesz (lásd Gödel teljességi tételét az irodalomban). A *rezolúció alaplépése* önmagában nem teljes, de ha előtte átalakítjuk a formulákat (velük ekvivalens más formulákká, szükség esetén a nyelvet is bővítve), és megengedünk változó-helyettesítéseket is, akkor már teljes.
- Ha tehát a levezetési szabályaink helyesek és teljesek, akkor minden igaz (vagyis P-ből következő) állítás bebizonyítható velük, és egyetlen hamis (vagyis az ellentéte P-ből következő) állítás sem bizonyítható be. Sajnos azonban lehetnek eldönthetetlen állítások is, vagyis olyan állítások, amelyek P-nek valamely modelljében igazak, más modelljében nem. Ha például P két formulából áll: (1) $\forall X\,(X > 1+1 \wedge \exists Y\, X = Y + Y) \rightarrow \neg$prím$(X)$ (egyetlen 2-nél nagyobb páros szám sem prím); (2) prím$(1+1)$ formulákból áll, akkor a prím$(1+1+1)$ formula eldönthetetlen, mert van olyan modell (a természetes számok a szokásos módon), ahol igaz, és van olyan modell, ahol hamis (pl. ahol csak a 2 prím). Rossz hír, hogy Gödel nem-teljességi tétele miatt bármely P (véges) logikai program, amelynek nyelve tartalmazza a természetes számokat (vagy bármely ennél bonyolultabb struktúrát) ellentmondásos vagy található egy benne eldönthetetlen állítás. Röviden: szinte bárhogy is írunk logikai programot, mindig lesz hozzá eldönthetlen állítás.

Definit klózok

- Az eddig ismertetett logikai programozás legnagyobb hibája, hogy a hatékonyság nem biztosíott: ha a gép csak vaktában alkalmazza a levezetési szabályokat, akkor lehet, hogy sosem talál rá a bizonyításra. Vagy esetleg rátalál, de túl lassan vagy túl sok memóriát használva. Két út kínálkozik: (1) okosan választjuk meg a levezetési szabályokat, és azt is előírjuk, mikor milyen sorrendben kell őket alkalmazni, és bebizonyítjuk, hogy előírásaink hatékonyak (2) szűkítjük a nyelvet, és reménykedünk, hogy az egyszerűsített nyelvre hatékonyabb levezetési módszerek léteznek.
- A Prolog alkotói a (2)-es utat követték. (Vannak egyéb irányzatok is, születtek olyan tételbizonyító programok, melyek a teljes elsőrendű logikát megengedik.) A szűkítés a következő: a P logikai programban csak $\forall \ldots A_0 \leftarrow (A_1 \wedge A_2 \wedge \cdots \wedge A_n)$ alakú formulákat engedünk meg, ahol minden A_i atomi formula ($n = 0$ is megengedett, ekkor A_0 feltétel nélkül igaz), a bizonyítandó B állítás pedig $\exists \ldots B_0 \wedge B_1 \wedge \cdots \wedge B_m$ alakú kell legyen. A bizonyítás módszere a később ismertetett *SLD-rezolúció* lesz, ami „Linear resolution for Definite clauses with Selection function"-t jelent. Itt a *definit klóz* (*definite clause*), rövidem *klóz* fogalom a P program formuláinak fent bemutatott szintaxisát jelöli, A_0 a klóz feje, a többi formula pedig a klóz törzse. A *lineáris* kifejezés arra utal, hogy a B állításból származó klóz minden rezolúciós lépésben részt vesz, a *kiválasztási függvény* pedig az a függvény lesz, ami a bizonyítás következő lépését, pontosabban a B' atomi formuláját meghatározza (ettől a függvénytül függ, hogy a bizonyítás milyen gyorsan készül el).
- A *definite clause* kifejezés *definit klózra* fordítása megkönnyíti az angol és magyar nyelvű fogalmak megfeleltetését. Egy másik jelölt a *határozott klóz* volt.
- Vegyük észre, hogy a $\forall \ldots A_0 \leftarrow (A_1 \wedge A_2 \wedge \cdots \wedge A_n)$ klóz ekvivalens a $\forall \ldots A_0 \vee \neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n$ formulával, vagyis egy olyan diszjunkcióval, amiben 1 kivétellel az összes atomi formula negálva van. Indirekt bizonyítással a P-beli klózokon túl feltesszük, hogy a B formula hamis, tehát $\forall \ldots \neg B_1 \vee \neg B_2 \vee \cdots \vee \neg B_m$ igaz. Ez tehát egy olyan diszjunkció, ahol az összes atomi formula negálva van. $m = 0$ esetén ezt a formulát azonosan hamisnak tekintjük.
- A $\forall$ kvantorok némelyike a `nagyszülője(A,C) :- szülője(A,B), szülője(B,C)))`. klóz (elsőrendű logikában $\forall A \forall B \forall C$ nagyszülője $(A, C) \leftarrow$ (szülője$(A, B) \wedge$szülője(B, C))) helyett $\forall A \forall B$ nagyszülője $(A, C) \leftarrow \exists B$(szülője$(A, B) \wedge$ szülője(B, C)) is írható.
- Az SLD-rezolúció nevű bizonyításkereső módszer (lásd később) határozott klózokból gyárt újabb határozott klózokat. Az általános rezolúció *diszjunktív klózokkal* teszi ugyanezt. Egy diszjunktív klózban tetszőlegesen sok negálatlan atomi formula megengedett, tehát $\forall \ldots \neg^? A_1 \vee \neg^? A_2 \vee \cdots \vee \neg^? A_n)$ alakú. Ez az alak bővebb a határozott klózénál.
- Ez a *klóz*-fogalom megegyezik a Prologból ismert klózzal: például a $\forall \ldots P \leftarrow (Q \wedge R \wedge S)$ logikai formulának a „P :- Q, R, S." Prolog-klóz felel meg. $n = 0$ esetén tényállítást, $n \geq 1$ esetén szabályt kapunk.
- A fordított irányú implikáció helytelen. Tehát például $\forall X \forall Y \forall Z$ nagyszülője$(X, Z) \leftarrow$ szülője$(X, Y) \wedge$ szülője(Y, Z) nem nyilatkozik arról, hogy ha a két „szülője" feltétel nem teljesül egyszerre (vagyis nem létezik megfelelő Y), akkor vajon X nagyszülője-e Z-nek vagy sem. Ilyen, negált feltételt vagy következményt tartalmazó formula nem is írható át klózzá – a nyelvet leszűkítettük, hogy a bizonyítás hatékonyobb legyen, íme a szűkebb nyelvben bizonyos szabályok és kérdések nem is fogalmazhatók meg.
- Néhány példa arra, hogy a klózokra való korlátozás miatt miről kell lemondanunk. A „ha jó idő lesz és nem leszünk fáradtak, akkor kirándulunk" könnyen egyetlen klózzá alakítható, „ha jó idő lesz vagy lesz nálunk esőkabát, akkor kirándulunk" két klózzá alakítható, de a „ha rossz idő lesz, akkor moziba vagy színházba megyünk" nem alakítható klózokká. Ha az atomi formulák mellett a negáltjaikait is megengednénk, akkor két klózzá alakítható lenne, így: „ha rossz idő lesz, és nem megyünk moziba, akkor színházba megyünk" és fordítva („ha rossz idő lesz, és nem megyünk színházba, akkor moziba megyünk").

A legszűkebb Herbrand-modell

- Ha a *logikai program* csak *határozott klózokból* áll, akkor *biztosan* nem tartalmaz ellentmondást, vagyis van modellje. Megfelel például az a modell, melyben minden predikátum azonosan igaz. De megfelel az a modell is, melyben a klózfejekben szereplő A_0 atomi formulákból képzett *tömör* (*ground*) atomi formulák mind igazak (a feltételtől függetlenül). A fenti két példamodellben túl sok az igaz reláció. A programozót általában a legszűkebb modell érdekli, vagyis az a modell, amiben csak az igaz, ami a programból következik. Tehát például ha a járat/3 predikátum adja meg, hogy honnan hová mikor indul járat, akkor a modellben (jelen esetben a modell a valóságot jelenti) ne legyen más járat, mint ami a járat/3 forráskódjában fel van sorolva. Az alábbiakban ilyen modellt keresünk.
- *Herbrand-interpretációnak* (H-ió) nevezünk egy olyan interpretációt, amiben különböző szintaktikájú tömör termek különbözőek. Pontosabban: H-ió a minden tömör termnek önmagát felelteti meg

(tehát az ő világának elemei objektumai az adott nyelv tömör termei; apja(ernő) nem Ernő apját jelöli, hanem egyszerűen az apja(ernő) termet), a predikátumokra nincs kikötés. Tehát például $1 - 1 \cdot 1$ és 0 egy H-ióban különböző, míg a szokásos interpretációban egyező. Ha egy H-ió modellje a programnak, akkor *Herbrand-modellnek* nevezzük. Mi a *legszűkebb Herbrand-modellt* (*least Herbrand model*) keressük, vagyis a predikátumokhoz szeretnénk hozzárendelni relációkat, hogy modellt kapjunk, és a relációknak a lehető legkevesebb elemük legyen. A „lehető legkevesebb" értelmezése: mivel Herbrand-modellek metszete (vagyis a bennük levő relációk metszete) is Herbrand-modell, ezért egy P (klózokból álló) program legszűkebb Herbrand-modellje legyen az összes Herbrand-modelljének metszete.

- Kevésbé fontos fogalmak: a *Herbrand-univerzum* az összes lehetséges tömör term halmaza az adott nyelvben. Például ha a függvényjelek aritással 0/0, 1/0, $\cdot$/2 és $+$/2, akkor a Herbrand-univerzum elemei 0, 1, $0+0$, $0 \cdot 1$, $0 \cdot (1+1)$ stb. (végtelen sok). A *Herbrand-bázis* pedig az összes tömör atomi formula halmaza, tehát ha $\leq$/2 és prím/1 a predikátumok, akkor például $1 \leq 0$, $1 \leq (1+1)$ és prím$(1+1+1+1)$ is elemei a Herbrand-bázisnak.
- Nem bizonyítjuk: a P program logikai következményei (vagyis az összes belőle következő formulák) közül az atomi formulák közül a tömörek épp megegyeznek P legszűkebb Herbrand-modelljének relációival.
- Nem bizonyítjuk: a legszűkebb Herbrand-modell konstruálható fixpontig menő itarációval. Legyen $\varnothing$ az az interpretáció, amelyben minden predikátum azonosan hamis, legyen ground(P) a P klózainak összes tömör példánya (tehát ahol azonos változó szerepel, azt azonos tömör kifejezéssel kell helyettesíteni). $T_P(I)$ ekkor egy függvény, ami az I interpretáció relációit bővíti: $T_P(I) := \{A_0 | (A_0 \leftarrow (A_1 \wedge A_2 \wedge \ldots \wedge A_n)) \in \text{ground}(P) \text{ és minden } A_i \in I\}$. A T_P függvényt iterálva egyre bővebb interpretációt kapunk. Nem bizonyítjuk: ha $T_P(I') = I'$, akkor I' a legszűkebb Herbrand-modellje P-nek.
- Például ha a program `páratlan(s(nulla)). páratlan(s(s(X))) :- páratlan(X).`, akkor az i-edik iteráció után az interpretációnk épp a $\{\text{páratlan}(\text{s}^{2n+1}(\text{nulla})) | n < 2i\}$ tömör atomi formulákat halmaza. Végtelen sokszor (ω-szor) kell iterálni, hogy az összes páratlan szám (páratlan$(\text{s}^{2n+1}(\text{nulla}))$) meglegyen. Ez lesz a legszűkebb Herbrand-modell.

- A fenti program mellett az `utazhat(X).` hívás kétszeresen is sikerül, mindkétszer $X = \text{c}$ behelyettesítést adva.

```
utazhat(X) :- van_jegye(X).
utazhat(X) :- nyugdíjas(X).
utazhat(X) :- vak(Y), kutyája(Y,X).
vak(a).
vak(b).
kutyája(a,c).
kutyája(b,c).
```

A rezolúciós bizonyítás az alábbi klózokból indul:

utazhat$(X) \vee \neg$van_jegye(X)	(1)
utazhat$(X) \vee \neg$van_jegye(X)	(1)
utazhat$(X) \vee \neg$nyugdíjas(X)	(2)
utazhat$(Y) \vee \neg$vak$(X) \vee \neg$kutyája(X, Y): vakvezető kutya	(3)
vak(a)	(4)
vak(b)	(5)
kutyája(a, c)	(6)
kutyája(b, c)	(7)
$\neg$utazhat(X)	(8)

Az új klózok:

$\neg$vak$(X_1) \vee \neg$kutyája(X_1, X_2): (3) és (8), $X_2 = Y_1$	(9)
$\neg$kutyája(a, X_2): (9) és (4), $X_1 = a$	(10)
$\square$: (10) és (6), $X_2 = c$	(11)

Itt már készen van a bizonyítás, de a Prolog további ellenpéldákat keres:

$\neg$kutyája(b, X_2); (9) és (4): $X_1 = b$	(12)
$\square$: (12) és (6): $X_2 = c$	(13)

Az első ellentmondásban az $X = X_2 = c$, és a másodikban is az $X = X_2 = c$ ellenpéldát kaptuk.

- Két term legáltalánosabb egyesítőjét a Prologból tanult algoritmussal lehet megkeresni. Vegyük észre, hogy két kifejezés pontosan akkor egyesíthető, ha külön-külön tömörré tehetők úgy, hogy ugyanazt a tömör termet kapjuk. Másképpen: két term pontosan akkor egyesíthető, ha változólekötések után Herbrand-interpretációban egyenlővé tehetők. Az egyesítés e módja a lehető legóvatosabb: van olyan interpretáció (és modell), ahol $1 - X$ és 0 egyenlővé tehetők, de mivel olyan interpretáció is van, ahol nem, ezért inkább nem egyesítjük az -rezolúcióban. (Emlékeztető: a rezolúció csak olyan állítást tud bebizonyítani, ami minden modellben igaz.)

Egyesítés

- Az egyesítésről mondottak az általános rezolúcióra és az SLD-rezolúcióra is igazak.
- A gép a bizonyítás keresése során a rezolúció alaplépését ismételgeti. Ez a P program klózait bővíti egy olyan új klózzal, melyet a már ismert klózokból az egyik levezetési szabályt egyszer alkalmazva meg lehet kapni. A rezolúció alaplépése vázlatosan a következő (ismétlés): ha van egy $A \vee P$ és egy $\neg B \vee Q$ alakú klóz, ahol A és B atomi formulák, P és Q pedig diszjunktív klózok (0 vagy több atomi formulával), akkor az új klóz $P' \vee Q'$ lesz.
- SLD-rezolúció esetén az $A \vee P$ mindig az eredeti programból jön, ahol A egy klóz feje ($A = A_0$) – arról, hogy melyik klózfej legyen a sok illeszkedő közül, a bizonyítási eljárás specifikációja dönt (ez nem a kiválasztási függvény!). $\not{B} \vee Q$ pedig mindig bizonyítandó állítás negáltjából származó formula – hogy ebből melyik atomi formula lesz B, arról a kiválasztási függvény dönt.
- Ha a két megtalált klózban lennének azonos (nevű) változók, akkor az egyikben a változókat át kell nevezni.
- Az A és B atomi formuláknak *egyesíthetőknek* kell lenniük. Ez azt jelenti, hogy létezik egy olyan *változó-helyettesítés* (*substitution*), amely A és B változóihoz termeket rendel, és a helyettesítés után kapott A' és B' megegyezik. Ha több alkalmas egyesítő (helyettesítés) is van, akkor azok közül a legáltalánosabbat (*mgu, most general unifier*) választjuk. Egy f helyettesítés általánosabb h-nál, ha létezik g helyettesítés, hogy f és g egymás után elvégzése épp h (vagyis $f \circ g = h$). Tehát először az $A \vee P$ formulán elvégezve a helyettesítést kapjuk $A' \vee P'$-t, majd $\not{B} \vee Q$-n elvégezve kapjuk $\not{A'} \vee Q'$-t, és ezeken végrehajtva egy rezolúciós lépést kapjuk $P' \vee Q'$-t.
- A rezolúció alaplépését mostmár elég precízen definiáltuk. Könnyű belátni, hogy az alaplépés helyes (*sound*). A teljesség bizonyítása viszont hosszadalmas.
- Ha a rezolúció sikerrel zárul, vagyis találtunk egy ellenpéldát, akkor kövessük végig, hogy milyen helyettesítéseket alkalmaztunk az ellentmondáshoz való eljutás során. Ezeket az eredeti, bizonyítandó B állításra alkalmazva megkapjuk a legáltalánosabb ellenpéldát. (Ez az ellenpélda megegyezik azzal, amit a Prolog kiír siker esetén.)

- A Prolog-os egyesítő algoritmus megengedi az $X = g(X)$ és $T = f(T, Y)$ kapcsán előkerülő, végtelen termet eredményező egyesítéseket. A végtelen term a matematikában nem szabályos, és a Prolog csak azért engedi meg, mert a unify_with_occurs_check/2 túl lassú (könnyű n db egyenlőséget felírni, amelyben az *előfordulás-ellenőrző egyesítés* 2^n idejű).
- Előfordulás-ellenőrzés nélkül az SLD-rezolúció nem helyes! Például a T-vel és $f(T, Y)$-nal egyszerre egyesíthető termeket bizarrnak nevezzük, és egy Y term őrületes, ha van olyan bizarr term, aminek ő a második argumentuma, akkor nyilvánvaló, hogy nem létezik őrületes term (mivel ekkor létezne bizarr term is, tehát létezne végtelen term, ami a matematikában nem igaz). Az alábbi Prolog-program

```
effes(f(T,_Y),T).
bizarr(T) :- effes(T,T).
őrületes(Y) :- bizarr(f(_X,Y)).
```

mellett a `őrületes(Y).` cél előfordulás-ellenőrzés mellett helyesen meghiúsul, míg nélküle sikeres lesz, egyesítés nélkül, vagyis azt kaptuk, hogy minden term őrületes.

SLD-rezolúció

- Tekintsük a P program és a $\neg B$ állítás $\vee$-okkal felírt alakját. Ha a gép el tudja érni, hogy a B-beli diszjunkcióból fogyjanak el az atomi formulák, tehát váljon üressé, akkor eljutott egy hamis formuláig ($\square$), tehát az indirekt bizonyítás sikeresen befejeződött. Ehhez a gép B-ben cserélgetni kezdi a formulákat: kiválaszt egy $\neg B_j$ formulát, és kicseréli egy vagy több negált atomi formula diszjunkciójára. (A régi B-ről elfeledkezik, már soha többet nem fogja használni.) Ezáltal B szerkezete megmarad, csak a benne található atomi formulák változnak. A cserélgetés során B hossza nőhet is, de a végcél az, hogy az összes formula elfogyjon belőle.
- Az SLD-rezolúció alaplépése a következő (ismétlés): kiválaszt egy $\neg B_j$ formulát a legutóbbi csupa negatív atomi formulát tartalmazó formulából, és kiválaszt egy A_0 klózfejet a P programból. Ha B_j és A_0 megegyezik (vagy egyesíthetőek, lásd később), akkor $\neg B_j$-t kicseréli a klóz törzsére (csupa $\neg A_i$ diszjunkciója). Vegyük észre, hogy ez a lépés a rezolúció alaplépése. Az egyesítés előtt a programklózban átnevezi a változókat, hogy ne ütközzenek a B-beli

változókkal. Továbbá a klózok egyesítése (lásd később) változást okoz a formulákban szereplő változókban: ezt a változást véghez kell vinni a a keletkező B összes atomi formuláján, tehát valójában nem csak B_j változik, hanem a többi atomi formula is.

- Hátra van még annak belátása, hogy az SLD rezolúció helyes és teljes, továbbá annak vizsgálata, hogy milyen feltételek mellett fejeződik be véges időben. (Ezeket később.)
- Mi történik akkor, ha a bizonyítandó B állítás nem következik a programból? Ezt úgy érzékeli a gép, hogy az SLD-rezolúció alaplépését már sehogy sem tudja véghezvinni, vagyis talál egy olyan B_j-t, ami semelyik klózfejre nem illeszthető. Ekkor viszatér egy korábbi, az előző lépés(eke)t megelőző B-re, és az ott próbálja másképpen alkalmazni az alaplépést. Tehát egy *visszalépéses keresés* (*backtrack*) jön létre. Ha végül minden ágon elakad, akkor megáll, és kiírja, hogy az állítás nem következik programból. Elképzelhetők azonban végtelen ágak is, ekkor az SLD-rezolúció nem áll le, a gép végtelen ciklusba kerül.
- Az alaplépés alkalmazásakor B_j és A_0 szabadon választható (feltéve, hogy egyesíthetőek). A gép azt fogja választani, amit a kiválasztási függvény előír. Ily módon a bizonyítás megtalálásának hatékonysága (és végessége) a kiválasztási függvénytől is függ. A Prolog is SLD-rezolúciót futtat, kiválasztási függvénye mindig B_1-et választja, és az azonos eljáráshoz tartozó klózokat a visszalépés során a programban előfordulásuk sorrendjében próbálja végig. Ez egy elég buta megoldás (és néha fölöslegesen vezet végtelen ciklushoz), de a memórihasználat és egyéb implementációs szempontok miatt ezt választották.
- Tekintsük az alábbi Prolog-programot:

```
nagyapja(X,Z) :- apja(X,Y), szülője(X,Y).
szülője(X,Y) :- apja(X,Y).
szülője(X,Z) :- apja(X,Z).
apja(a,b).
anyja(b,c).
```

- Az SLD-rezolúció menetét, a gép időbeli működését egy *levezetési fában* (*SLD-fa*) ábrázolhatjuk. A fa gyökere a B állítás (diszjunktív alakja), a további csúcsok pedig a B állítás módosításai. Egy B' csúcsból él megy a B'' csúcsba (és az él címkéje a kiválasztott atomi formula sorszáma és a kiválasztott klóz sorszáma), ha B'-ből B'' egyetlen alaplépéssel elérhető. A fa levelei $\Box$ címkéjűek (ekkor az indirekt bizonyítás eljutott az ellentmondásig), vagy egyéb diszjunkciók (még nem teljesen megvizsgálva vagy kiderült, hogy alaplépés innen nem lehetséges). A bizonyítás maga egyetlen gyökértől $\Box$ levélig menő út a fában. A fa többi ága ekkor nem érdekes. Ha az összes megoldásra szükség van, akkor az összes ilyen utat és a rajtuk végrehajtott változó-behelyettesítéseket kell tekinteni. A fenti példaprogram esetén a `nagyapja(a,X).` kérdés levezetési fája (póriasan):

```
nagyapja(a,X).
  apja(a,Y0), szülője(Y0,X).
     szülője(b,X).
       apja(b,X).
         nyelő, nem lehet továbblépni
       anyja(b,X).
         (üres csúcs: hamis)
```

- Egyetlen bizonyítást (gyökér $\to \Box$ út) ábrázolhatunk *bizonyítási fában* is. Üres gyökeréből él megy a B állítás atomi formuláiba. Belső csúcsaiban két, egymással egyesíthető atomi formula szerepel. A levelekben egyetlen atomi formula van, ez egyesíthető egy tényállítással. Egy csúcsból kifelé induló elek az eredeti $B_j + A_0$ csúcsból az $A_1+? \dots A_n+?$ csúcsokba menek. A bizonyítás akkor teljes, ha elérkeztünk egy levélben egy tényállításig. A bizonyítási fa *konszisztens,* ha a csúcsokban levő egyesítések egyszerre elvégezhetők. A fenti példaprogram esetén a `nagyapja(X,Z).` kérdés levezetési fája (póriasan):

```
nagyapja(X0,Z0)
  apja(X0,Y0) = apja(a,b)
    igaz
  szülője(Y0, Z0) = szülője(X1, Y1)
    anyja(X1, Y1) = anyja(b,c)
      igaz
```

- Képzeljünk el egy végtelen levezetési fát, melyben minden lehetséges csúcs és él szerepel. Az SLD-rezolúció ekkor a fában egy útkeresés a gyökértől valamely $\Box$ csúcsig. Ha létezik $\Box$ csúcs, akkor ahhoz szélességi kereséssel véges időben el lehet jutni. (Ha nem létezik záró csúcs, akkor lehet, hogy végtelen ideig tart a keresése, például a `p :- p.` program esetén egy végtelen hosszú út a fa.) A Prolog mégsem szélességi, hanem mélységi keresést használ, aminek hátránya, hogy végtelen ciklusba eshet. Ennek az az oka, hogy a szélességi keresés memóriaigénye nagy (az előző szint összes B' állapotát a memóriában kell tartani), továbbá egy előrelépés lassú.
- Miért szűkebb az SLD-rezolúció az általános rezolúciónál? Azért, mert az általános rezolúció diszjunktív klózaiban (és a B állításban is) tetszőleges számú negált és nem negált atomi formula szerepelhet. Az általános rezolúció kiválaszt két tetszőleges klózt (tehát az egyik nem mindig a B'), és belőlük egyesíti egy A és egy $\not A$ atomi formula argumentumait, majd végrehajtja a rezolúció alaplépését, egy új formulát hozva létre. Tehát $A \vee P$ és $\neg A \vee Q$-ból $P' \vee Q'$ keletkezik (a vessző az egyesítés hatását mutatja). Az általános rezolúciónak tehát több választási lehetősége van, ezért lassabban jut el a megoldáshoz ($\Box$).
- !! technikai:

  Not: felhasznált irodalom Imp: kiv. fv. mit választ (most: atomi formulát) Imp: néhány hosszabb formális definíció a könyvből