Debreceni Egyetem

Informatikai Kar

Számítógéptudományi Tanszék

Algoritmusok a tételbizonyításban

Témavezető:

dr. Várterész Magda

egyetemi docens

Készítette:

Tanyi Attila

Programtervező

informatikus (B.Sc.)

Debrecen

2009.

Tartalomjegyzék

1. Bevezetés

1.1. Történeti áttekintés

A rezolúció eljárását az amerikai John Alan Robinson matematikus vezette be 1965-ös *A Machine-Oriented Logic Based on the Resolution Principle* című tanulmányában. Ezzel lefektette az automatikus tételbizonyítás és logikai programozás alapjait.

Az ő eredményei, illetve Robert Anthony Kowalski lengyel származású brit matematikus 1971-ben megjelent *Linear Resolution with Selection Function* című tanulmánya és a Horn-klózok procedurális értelmezésében tett előrelépései nyomán jöhetett létre az 1972-ben megjelent Prolog logikai programozási nyelv, amely a Horn-formulák esetén teljes lineáris inputrezolúciós stratégián alapul. A Prolog kifejlesztése Robert Kowalski amerikai és Alain Colmerauer francia számítógéptudósok nevéhez kötődik, és egyike volt az első logikai programozási nyelveknek.

A Prologhoz hasonló logikai programozási nyelvek általában speciális alakú formulákkal, programklózokkal dolgoznak. Az ehhez szükséges átalakításokkal kapott konjunktív normálformák gyakran exponenciális (n változó esetén 2^n) sok konjunkciós tagból állnak. Ezért az 1950-es évektől komoly kutatási területté vált a matematikai logikán belül a normálformák minimalizálása, rengeteg egyszerűsítő eljárást fedeztek fel. A minimális normálformák az elektronikai eszközök áramköreinek tervezésénél is alapvető szerepet játszanak, hiszen ezekben az esetekben a gyártandó áramkörök fizikai méretének csökkenését érhetjük el.

1.2. Motiváció

A szakdolgozatom témáját feltétlenül logikához kapcsolódó területről szerettem volna választani. Az automatikus tételbizonyítás gyakorlati jelentősége miatt keltette fel a figyelmemet. A logikus következtetés, a helyes érvelés az emberek többségének alapvető, és a mindennapi problémamegoldáshoz nélkülözhetetlen képessége. A következtetéseink helyességének ellenőrzésében pedig immár a számítógép is segítségünkre lehet az automatikus tételbizonyítás által. Jelen dolgozatban (a téma összetettsége miatt) csak az ítéletlogikai algoritmusokkal foglalkozok. A tételbizonyításra kidolgozott módszerek közül

Davis és Putnam módszere nulladrendű formulákra, és a Prolog által is használt rezolúció képezi a dolgozat tárgyát, illetve az ennek használatához elengedhetetlen konjunktív normálformákat generáló, illetve minimalizáló algoritmusok, a Quine-McCluskey-módszer.

1.3. Szoftverek

A szakdolgozat LaTeX forrását a TeXnicCenter program segítségével szerkesztettem. Bizonyos ábrák elkészítéséhez az OpenOffice.org Draw szoftvert használtam. A mellékelt programokban az algoritmusok implikálásához a Java nyelv mellett döntöttem, a programozást NetBeans 6.5 fejlesztői környezetben végeztem. A szoftverek Swing felülettel rendelkeznek.

Két program kapott helyet a CD-n. A **minimal** nevű egy számára megadott tetszőleges nulladrendű formulával dolgozik, szintaktikailag ellenőrzi azt, amennyiben helyesnek találja, elkészíti az igazságtábláját, egyszerűsített konjunktív normálformába alakítja, majd a leggyorsabb esés módszerével a fedési tábla alapján létrehozza a minimális alakot. A másik program a **rezolucio** nevet kapta, a Prolog mintájára atomi formulákból felírt tények és szabályok alkotta program alapján igaz/hamis választ ad a feltett kérdésünkre.

2. Definíciók, tételek, bizonyítások

2.1. Alapfogalmak

2.1. definíció. *Literálnak* (l) nevezünk egy atomot (*pozitív literál*) vagy a negáltját (*negatív literál*). l negáltját az $\bar{l}$ jellel jelöljük.

2.2. példa. $X, Y, Z, \neg X$

2.3. definíció. Literálok véges halmazát *klóznak* ($\mathcal{C}$) nevezzük, és a literálok diszjunkciójaként értelmezzük. Az üres klóz ($\square$) mindig hamis.

2.4. példa. Klózokra: $\{X, Z\}, \{\neg Y\}, \{Y, \neg Z\}$

2.5. definíció. Klózok (nem feltétlenül véges) halmazát *formulának* nevezzük, és a klózok konjunkciójaként értelmezzük. Az üres formula ($\varnothing$) mindig igaz.

2.6. példa. Formulára: $\{\{X, Z\}, \{Y, \neg Z\}, \{\neg Y\}, \{\neg X, U\}, \{V, \neg U\}\}$

2.7. definíció. Literálok konzisztens - vagyis X-et és $\neg X$-et egyszerre nem tartalmazó - halmazát *hozzárendelésnek* nevezzük. Amennyiben egy $\mathcal{A}$ hozzárendelés az összes X atomi formulára tartalmazza X-et vagy $\neg X$-et, *teljes hozzárendelésnek* nevezzük.

2.8. definíció. $\mathcal{A}$ *kielégíti* $\mathcal{S}$-t ($\mathcal{A} \vDash \mathcal{S}$) akkor és csak akkor, ha minden $\mathcal{S}$-beli $\mathcal{C}$ klózra $\mathcal{C} \cap \mathcal{A} \neq \varnothing$, vagyis az $\mathcal{A}$ általi hozzárendelés az összes $\mathcal{S}$-beli klózt igazzá teszi.

2.2. Normálformák

A különböző normálformákkal kapcsolatos irodalomban gyakran más terminológiát és jelöléseket használnak.

2.9. definíció. Legyenek $l_1, l_2, ..., l_n$ literálok. Ekkor
$l_1 \wedge l_2 \wedge ... \wedge l_n$ *elemi konjunkció*,
$l_1 \vee l_2 \vee ... \vee l_n$ *elemi diszjunkció*.

2.10. példa. $X \wedge \neg Y \wedge Z$ elemi konjunkció,
$\neg X \vee \neg Y \vee Z$ elemi diszjunkció.

2.11. definíció. Legyenek $K_1, K_2, ..., K_n$ elemi konjunkciók,
$D_1, D_2, ..., D_n$ elemi konjunkciók. Ekkor
$D_1 \wedge D_2 \wedge ... \wedge D_n$-t *konjunktív normálformának*,
$K_1 \vee K_2 \vee ... \vee K_n$-t *diszjunktív normálformának* nevezzük.

2.12. példa. $(\neg X \vee \neg Y \vee Z) \wedge (X \vee Y \vee Z)$ konjunktív normálforma,
$(X \wedge \neg Y \wedge Z) \vee (X \wedge Y \wedge \neg Z)$ diszjunktív normálforma.

2.3. Rezolúció

2.13. definíció. $\mathcal{C}_1$ és $\mathcal{C}_2$ legyenek $\{l\} \sqcup \mathcal{C}_1'$ és $\{\bar{l}\} \sqcup \mathcal{C}_2'$ alakú klózok. Ekkor $\mathcal{C}_1$ és $\mathcal{C}_2$ klózok rezolválhatók, és *rezolvensük* a $\mathcal{C} = \mathcal{C}_1' \cup \mathcal{C}_2'$. Ekkor $\mathcal{C}_1$-et és $\mathcal{C}_2$-t *szülőknek* és $\mathcal{C}$-t a *gyermeküknek* nevezzük, és azt mondjuk, hogy l a *kirezolvált* literál.

2.14. megjegyzés. $\{l\} \sqcup \mathcal{C}$ azt a halmazt jelenti, melynek elemei l és a $\mathcal{C}$ minden egyes eleme. (Vagyis $\{l\}$ és $\mathcal{C}$ diszjunkt halmazok.) $\mathcal{C}_1 \cup \mathcal{C}_2$ eleme minden olyan klóz, amely $\mathcal{C}_1$ és $\mathcal{C}_2$ közül legalább az egyiknek eleme.

2.15. definíció. Adott $\mathcal{S}$ *formulából* $\mathcal{C}$ *(rezolúciós) levezetésének* vagy *bizonyításának* egy olyan $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n = \mathcal{C}$ véges klózsorozatot nevezzük, ahol $\mathcal{C}_i$ vagy $\mathcal{S}$ eleme, vagy a $\mathcal{C}_j$ és $\mathcal{C}_k$ klózok rezolvense, ahol $j, k < i$. Ha létezik ilyen levezetés, akkor azt mondjuk, hogy $\mathcal{C}$ *rezolúcióval bizonyítható* $\mathcal{S}$*-ből*, és ezt a tényt $\mathcal{S} \vdash_R \mathcal{C}$-vel jelöljük.
Ha $\mathcal{S}$-ből levezethető az üres klóz ($\square$), akkor $\mathcal{S}$ *(rezolúcióval) cáfolható* ($\mathcal{S} \vdash_R \square$), és ezt a levezetést $\mathcal{S}$ *(rezolúciós) cáfolatának* nevezzük.

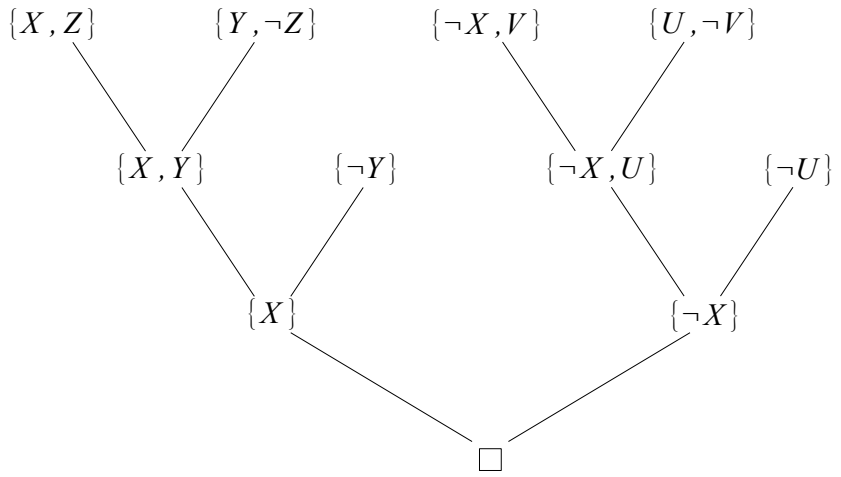
2.16. definíció. $\mathcal{S}$ formulából egy $\mathcal{C}$ klóz *levezetési fája* egy címkézett bináris T fa, a következő tulajdonságokkal:

(i) $\mathcal{T}$ gyökerének címkéje $\mathcal{C}$.

(ii) $\mathcal{T}$ levélelemeiek címkéi $\mathcal{S}$ elemei.

(iii) Ha bármely nem levél csomópont $\mathcal{C}_2$ címkével van ellátva, és a gyermekei $\mathcal{C}_0$-val, illetve $\mathcal{C}_1$-gyel, akkor $\mathcal{C}_2$ $\mathcal{C}_0$ és $\mathcal{C}_1$ rezolvense.

2.17. példa (Ábrázolás). A levezetési fát a fák szokásos ábrázolása helyett rajzoljuk úgy, hogy a gyökérelem kerüljön alulra, a levélelemek pedig felülre (vagyis fejjel lefelé).



Levezetési fa. $\mathcal{S} = \{\{X, Z\}, \{Y, \neg Z\}, \{\neg Y\}, \{\neg X, V\}, \{U, \neg V\}, \{\neg U\}\}, C = \square$.

2.18. tétel (A rezolúciós kalkulus helyes). *Ha egy $\mathcal{S}$ formula rezolúcióval cáfolható, akkor kielégíthetetlen.*

A bizonyításhoz szükségünk van az alábbi lemmára.

2.19. lemma. *Ha az $S = \{\mathcal{C}_1, \mathcal{C}_2\}$ formula (vagyis klózhalmaz) kielégíthető, és a $\mathcal{C}$ klóz $\mathcal{C}_1$ és $\mathcal{C}_2$ rezolvense, akkor $\mathcal{C}$ kielégíthető, és bármely $\mathcal{S}$-t kielégítő hozzárendelés kielégíti $\mathcal{C}$-t is.*

A lemma bizonyítása. Mivel $\mathcal{C}$ $\mathcal{C}_1$ és $\mathcal{C}_2$ rezolvense, így létezik l, $\mathcal{C}_1'$ és $\mathcal{C}_2'$ úgy, hogy $\mathcal{C}_1 = \{l\} \sqcup \mathcal{C}_1'$, $\mathcal{C}_2 = \{\bar{l}\} \sqcup \mathcal{C}_2'$ és $C = \mathcal{C}_1' \cup \mathcal{C}_2'$. Ha egy $\mathcal{A}$ hozzárendelés kielégíti $\{\mathcal{C}_1, \mathcal{C}_2\}$-t, úgy kielégíti $\mathcal{C}_1$-et és $\mathcal{C}_2$-t is. Mivel $\mathcal{A}$ egy hozzárendelés, így az nem fordulhat elő, hogy l és $\bar{l}$ is eleme lenne.

Tegyük fel, hogy $l \in A$. Mivel $\mathcal{A}$ kielégíti $\mathcal{C}_2$-t és $\bar{l} \notin A$, $\mathcal{A}$ kielégíti $\mathcal{C}_2'$-t, és így $\mathcal{C}$-t is. Hasonló módon, ha $\bar{l} \in A$, úgy mivel $\mathcal{A}$ kielégíti $\mathcal{C}_1$-et és $l \notin A$, $\mathcal{A}$ kielégíti $\mathcal{C}_1'$-t, és így $\mathcal{C}$-t is. □

A tétel bizonyítása. Ha a $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n$ klózsorozat az $\mathcal{S}$ formula rezolúciós levezetése, akkor a lemma n szerinti indukcióval megmutatja, hogy bármely $\mathcal{S}$-t kielégítő hozzárendelés $\mathcal{C}_i$-t is igazzá teszi. Amennyiben ez a levezetés $\mathcal{S}$ cáfolata, akkor $\mathcal{C}_n = \square$. Mivel nem létezik olyan hozzárendelés, amely az □-t igazzá tenné, ezért $\mathcal{S}$ kielégíthetetlen. □

2.20. tétel (A rezolúciós kalkulus teljes). *Ha egy $\mathcal{S}$ formula kielégíthetetlen, akkor létezik rezolúciós cáfolata.*

Az alábbiakkal előkészítjük a tétel bizonyítását.

2.21. definíció. Legyen $\mathcal{S}$ formula, l pedig literál. Ekkor

$$\mathcal{S}^l = \{\mathcal{C} - \{\bar{l}\} \mid \mathcal{C} \in \mathcal{S} \wedge l \notin \mathcal{C}\}.$$

Vagyis $\mathcal{S}^l$ $\mathcal{S}$ azon l klózaiból áll, amelyek sem l-t, sem $\bar{l}$-t nem tartalmazzák, vagy $\bar{l}$-t tartalmazták eredetileg, de ezt kivettük belőlük. Amennyiben $\mathcal{S}$ tartalmazta az $\{\bar{l}\}$ klózt, $\mathcal{S}^l$ tartalmazni fogja az üres klózt (□).

2.22. megjegyzés. Be kell hogy lássuk, ez a definíció első ránézésre elég mesterkéltnek tűnik. Valójában az alapötlet az, hogy $\mathcal{S}$-t esetekre bontva tudjuk elemezni. Ilyen tekintetben $\mathcal{S}^l$ tartozik ahhoz az esethez, amikor l értéke 1 (igaz), $\mathcal{S}^{\bar{l}}$ pedig ahhoz, amikor l hamis.

Az ötlet első fele, hogy ha l igaz, azok a klózok igazzá vannak téve, amelyek tartalmazzák l-t, hiszen az elemeik diszjunkciójával ekvivalensek. Mivel a formula a klózok konjunkciójával ekvivalens, így ezek a klózok elhagyhatók, hiszen nem változtatnak a formula kielégíthetőségén. Ezt fejezi ki az $\mathcal{S}^l$ definíciójában az $l \notin \mathcal{C}$.

Az ötlet második fele pedig, szintén feltételezve, hogy l igaz, $\bar{l}$ elhagyható minden klózból, hiszen a diszjunkcióként értelmezendő klózok kielégíthetőségét ez nem változtatja, és így az $\mathcal{S}$ formula kielégíthetőségét sem. Ennek nyomán került a definícióba az a rész, melynek nyomán $\mathcal{C}$-t $\mathcal{C} - \{\bar{l}\}$-re cseréljük.

2.23. tétel. *Ha* UNSAT$= \{\mathcal{S} \mid \mathcal{S}$ *kielégíthetetlen formula*$\}$*, akkor az* UNSAT *az alábbi kikötések által indukcióval definiált* $\mathcal{U}$ *formulahalmaz:*

(i) $\square \in \mathcal{S} \Rightarrow \mathcal{S} \in \mathcal{U}$*,*
illetve

(ii) $\mathcal{S}^l \in \mathcal{U} \wedge \mathcal{S}^{\bar{l}} \in \mathcal{U} \Rightarrow \mathcal{S} \in \mathcal{U}$*.*

2.24. definíció. $\mathcal{R}(\mathcal{S})$-t $\mathcal{S}$ *rezolúciós lezártjának* nevezzük, amennyiben az az alábbi indukciós definíció által meghatározott halmaz:

1. Ha $\mathcal{C} \in \mathcal{S}$, $\mathcal{C} \in \mathcal{R}(\mathcal{S})$.

2. Ha $\mathcal{C}_1, \mathcal{C}_2 \in \mathcal{R}(\mathcal{S})$, és $\mathcal{C}$ $\mathcal{C}_1$ és $\mathcal{C}_2$ rezolvense, akkor $\mathcal{C} \in \mathcal{R}(\mathcal{S})$.

A rezolúciós kalkulus teljességének bizonyítása. A tétel állítása, miszerint ha egy $\mathcal{S}$ formula kielégíthetetlen, akkor létezik rezolúciós cáfolata, $\square \in \mathcal{R}(\mathcal{S})$-sel ekvivalens kijelentés, ezt fogjuk igazolni.

Az UNSAT definíciójához igazodó indukció segítségével fogunk haladni a bizonyítás során. Ha $\square \in \mathcal{S}$, akkor természetesen $\square \in \mathcal{R}(\mathcal{S})$. Az indukciós lépéshez tegyük fel, hogy bizonyos l-re és $\mathcal{S}$-re $\square \in \mathcal{R}(\mathcal{S}^l)$ és $\square \in \mathcal{R}(\mathcal{S}^{\bar{l}})$. Azt kell megmutatnunk, hogy $\square \in \mathcal{R}(\mathcal{S})$. A feltevésünk szerint, léteznek az $\mathcal{S}^l$-ből, illetve $\mathcal{S}^{\bar{l}}$-ból az üres klóz levezetéséhez tartozó $\mathcal{T}_0$ és $\mathcal{T}_1$ levezetési fák. Ha $\mathcal{T}_0$ minden levéleleme $\mathcal{S}$-ben lévő klózzal van címkézve, akkor $\mathcal{T}_0$ máris $\square$ levezetése $\mathcal{S}$-ből. Ha nem, definiáljunk egy $\mathcal{T}_0'$ fát úgy, hogy a $\mathcal{T}_0$ fa minden $\mathcal{C}$ címkéjét, ami levélelem szülője, és nem $\mathcal{S}$-ben lévő klózzal címkézett, lecseréljük $\mathcal{C} \cup \bar{l}$-ra. Azt állítjuk, hogy $\mathcal{T}_0'$ $\{\bar{l}\}$ levezetési fája $\mathcal{S}$-ből. Az S^l definíciója alapján egyértelmű, hogy $\mathcal{T}_0'$ minden levéleleme szerepel $\mathcal{S}$-ben. Most le kell ellenőriznünk, hogy $\mathcal{T}_0'$ minden nem levél $\mathcal{C}$ csomópontja a $\mathcal{C}_0'$ és $\mathcal{C}_1'$ gyermekeinek $\mathcal{C}'$ rezolvensével van-e felcímkézve. Tegyük fel, hogy ezek sorra $\mathcal{T}_0$ $\mathcal{C}$, $\mathcal{C}_0$, $\mathcal{C}_1$ klózaihoz tartoznak. Mivel $\mathcal{T}_0$ egy rezolúciós levezetési fa, $\mathcal{C}$ $\mathcal{C}_0$ és $\mathcal{C}_1$ rezolvense. Érdemes megjegyezni, hogy $\mathcal{T}_0$-ban egy rezolválásban sem lehet l vagy $\bar{l}$ a kirezolvált literál, mivel egyik sem jelenik meg a $\mathcal{T}_0$ fa címkéi között (az S^l definíciója alapján).

Ezután tekintsük a $\mathcal{C}$, $\mathcal{C}_0$ és $\mathcal{C}_1$ klózokat $\mathcal{T}_0'$-ben. Ha például $\mathcal{C}_0$ és $\mathcal{C}_1$ (és így biztosan $\mathcal{C}$ is) $\mathcal{S}$-ben nem szereplő klózokkal címkézett levélelemek fölött helyezkednek el, akkor $\mathcal{C}' = \mathcal{C} \cup \{\bar{l}\}$ $\mathcal{C}_0' = \mathcal{C}_0 \cup \{\bar{l}\}$ és $\mathcal{C}_1' = \mathcal{C}_1 \cup \{\bar{l}\}$ rezolvense, mivel ez kell ahhoz, hogy $\mathcal{T}_0'$

rezolúciós levezetési fa legyen. A többi lehetséges esetben vagy mindhárom klóz ugyanaz marad a $\mathcal{T}_0'$-ben, mint $\mathcal{T}_0$-ban volt, vagy $\mathcal{C}$ változik, és $\mathcal{C}_0$ illetve $\mathcal{C}_1$ közül pontosan az egyik, $\{\bar{l}\}$ hozzáadásával. Mindezen esetekben $\mathcal{C}'$ továbbra is egyértelműen $\mathcal{C}_0'$ és $\mathcal{C}_1'$ rezolvense, és ismét meggyőződtünk róla, hogy $\mathcal{T}_0'$ levezetési fa.

Hasonlóan, ha minden $\mathcal{C}$ címkét $C \cup \{l\}$-re cserélünk a $\mathcal{T}_1$ levélelemei fölött elhelyezkedő olyan csomópontokban, amelyek címkéin levő klózok nem szerepelnek $\mathcal{S}$-ben, $\mathcal{T}_1'$-t kapjuk, $\{l\}$ levezetési fáját $\mathcal{S}$-ből (vagy, ha minden levél S-beli volt, akkor az üres klózét). Most már definiálhatjuk $\square$ levezetési fáját $\mathcal{S}$-ből egyszerűen hozzáillesztve $\mathcal{T}_0'$-t és $\mathcal{T}_1'$-et $\mathcal{T}$ gyökérelemének gyermekei mellé, amelyet $\square$-tel címkézünk. Mivel $\square$ $\{l\}$ és $\{\bar{l}\}$ rezolvense, az eredményül kapott $\mathcal{T}$ fa $\square$ rezolúciós levezetési fája $\mathcal{S}$-ből. $\square$

3. Normálformák

Ahhoz, hogy a rezolúciót elvégezhessük, a rezolválandó formulákat konjunktív normálformára kell hoznunk, majd ebből kell előállítanunk a klózhalmazt.

3.1. Kanonikus konjunktív normálforma

Tetszőleges nulladrendű formulához konstruálható vele ekvivalens konjunktív és diszjunktív normálformában lévő formula. Szemantikus átalakítást fogunk végezni, ehhez a formula igazságtáblázatára lesz szükségünk. Ebben a formulában szereplő atomok összes lehetséges kombinációjához megadjuk a formula értékét. Például a

$$\neg X \wedge (X \supset Y) \vee Z$$

formula igazságtáblája a következő:

X	Y	Z	$\neg X \wedge (X \supset Y) \vee Z$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Konjunktív normálformát kapunk, ha az igazságtábla minden 0 értékű sorához egy olyan termet rendelünk, amelyben az atomok negálva szerepelnek, ha az adott sorban az atomhoz tartozó oszlopban 1 áll, egyébként negálatlanul, az atomokat pedig diszjunkcióval kötjük össze. Ezután a termek konjunkciója adja a KNF alakot.

A példához tartozó KNF alakú formula így:

$$(X \vee Y \vee \neg Z) \wedge (X \vee \neg Y \vee \neg Z).$$

Az ilyen KNF alakot, amelyben minden elemi diszjunkcióban a formula minden ítéletváltozója szerepel negálva vagy negálatlanul, *kanonikus vagy kitüntetett konjunktív normálformának* nevezzük (KKNF).

3.2. Szintaktikai átalakítás KNF alakra

Szintaktikai műveletekkel, az igazságtáblázat ismerete nélkül is konstruálhatunk KNF alakot. Ennek lépései:

1. A logikai jelek közötti összefüggések alapján az implikációkat eltávolítjuk.

$$A \supset B \sim \neg A \vee B$$

2. De Morgan törvényeivel és a dupla tagadás szabályával elérjük, hogy a negáció csak atomokra vonatkozzon.

$$\neg(A \vee B) \sim \neg A \wedge \neg B$$
$$\neg(A \wedge B) \sim \neg A \vee \neg B$$
$$\neg\neg A \sim A$$

3. A disztributivitást felhasználva elérjük, hogy a konjunkciók és diszjunkciók megfelelő sorrendben kövessék egymást.

$$A \vee (B_1 \wedge ... \wedge B_n) \sim (A \vee B_1) \wedge ... \wedge (A \vee B_n)$$
$$(B_1 \wedge ... \wedge B_n) \vee A \sim (B_1 \vee A) \wedge ... \wedge (B_n \vee A)$$

4. Esetleg egyszerűsítünk.

3.1. példa. Az $(X \vee Y) \supset Z$ formula átalakítása konjunktív normálformába:

1. $(X \vee Y) \supset Z$
2. $\neg(X \vee Y) \vee Z$, az 1. szabályt alkalmazva
3. $(\neg X \wedge \neg Y) \vee Z$, a 2. szabályt alkalmazva
4. $(\neg X \vee Z) \wedge (\neg Y \vee Z)$, a 3. szabályt alkalmazva. Ez a formula már KNF alakú.

Egy n-változós logikai formula konjunktív normálformába alakítása akár 2^n méretű formulát is eredményezhet, mint ahogy a formula igazságtáblája is n változó esetén 2^n sorból áll. Nem tudjuk, létezik-e olyan módszer, amely egy formuláról eldönti, hogy az kielégíthető-e, és ehhez a legrosszabb esetben sem használ fel exponenciális időt.

3.3. Normálforma egyszerűsítése

A kanonikus konjunktív normálalak attól függően, hogy a formula hány esetben ad hamis értéket, elég terjedelmes lehet. Egy adott formulához azonban nem egyértelműen tartozik KNF alak, így nem feltétlenül ez a legegyszerűbb ilyen formula. A normálforma egyszerűsítésére több módszert is kidolgoztak, a legtöbbet idézett ezek közül a Quine-McCluskey-algoritmus.

3.4. A Quine-McCluskey-algoritmus

Az algoritmus Willard Van Orman Quine és Edward J. McCluskey amerikai kutatók nevéhez kötődik, 1956-ban jelent meg. McCluskey elsősorban áramkörök tervezésére szánta. Az eljárás eredetileg diszjunktív normálformában lévő formulák egyszerűsítésére íródott, azonban most tekintsük a konjunktív normálformákat egyszerűsítő változatát.

Tekintsünk egy n változós Boole-függvényt. Rögzítsük a formula változóinak sorrendjét, például az $X_1, X_2, ...X_n$ sorrendben. Ekkor minden egyes elemi diszjunkciót szemléltethetünk egy $\{0, 1, -\}^n$-beli vektorral. Ennek a vektornak az i. komponense legyen rendre $1, 0, -$ aszerint, hogy X_i az elemi diszjunkcióban negálatlanul szerepel, negált, illetve nem szerepel.

Például ha f teljes konjunktív normálformája

$$f(X_1, X_2, X_3, X_4) = (X_1 \vee X_2 \vee \neg X_3 \vee \neg X_4) \wedge (X_1 \vee \neg X_2 \vee \neg X_3 \vee X_4) \wedge (X_1 \vee X_2 \vee \neg X_3 \vee X_4) \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3 \vee \neg X_4) \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3 \vee X_4) \wedge (\neg X_1 \vee \neg X_2 \vee X_3 \vee \neg X_4) \wedge (X_1 \vee \neg X_2 \vee \neg X_3 \vee \neg X_4),$$

akkor az új jelöléssel

$$f = 1100 \wedge 1001 \wedge 1101 \wedge 0000 \wedge 0001 \wedge 0010 \wedge 1000.$$

Az algoritmus során f kitüntetett konjunktív normálformájából indulunk ki.

Rendezzük az ebben szereplő elemi diszjunkciókat az egyesek száma szerinti növekvő sorrendbe, és írjuk le őket egy oszlopba. Amikor az egyesek száma eggyel növekszik, egy vízszintes vonallal határoljuk el az új elemi diszjunkciót a régiektől. (Amennyiben az egyesek száma k-val növekszik, úgy $k - 1$ vízszintes vonalat húzunk.) Az eképpen felírt elemi diszjunkciókat ezáltal az egyesek száma szerint osztályoztuk, az osztályozás osztályai éppen két szomszédos vízszintes vonal közötti diszjunkciók halmazai (amennyiben ez nem üres halmaz). Ha két osztályt csak egy vonal választ el, akkor nevezzük e két osztályt *szomszédosaknak*. Szomszédos osztályokban lévő elemi konjunkciók esetén az egyesek száma éppen eggyel különbözik.

Ezután alkalmazzuk az összevonási szabályokat. Egy ilyen szabály csakis szomszédos osztályokba tartozó (és persze ugyanabban az oszlopban lévő) két elemi diszjunkcióra alkalmazható és pontosan akkor, ha a tekintett két elemi diszjunkciót megadó vektor csak egyetlen komponensében különbözik úgy, hogy az adott, i. komponens értéke az egyik (szükségképpen a felső) vektorban 0, a másikban pedig 1. Ez esetben az összevonás eredményeként az a vektor adódik, amelynek i-edik komponense $-$, a többi komponense pedig az összevont vektorok fennmaradó (azonos értékű) komponenseivel egyezik meg. Felülről lefelé haladva végezzük el a lehetséges összevonásokat, az összevonásokkal keletkező új elemi diszjunkciókat egy újabb (az előző jobb oldalára felírt) oszlopba írva. (Ha ugyanazon elemi diszjunkció többször is előáll összevonások eredményeként, akkor elég azt csak az első alkalommal leírni.)

Valahányszor lefelé újabb két osztály között kezdjük el az összevonásokat végezni, vagy egy üres osztályt átlépünk, a kialakítandó új oszlopban is egy vízszintes vonalat húzunk. (Ezáltal az újonnan létrejövő oszlopban is vízszintes vonalak határolják majd el a szomszédos osztályokat.) A régi oszlopban továbbá, ha egy elemi diszjunkció összevonás(ok)ban vesz részt, akkor azt (az első ilyen alkalommal) megjelöljük.

Miután az új oszlop kialakult, és a régiben bizonyos elemeket megjelöltünk, az új oszloppal folytatjuk az eljárást. Tehát most az ebben az oszlopban álló elemi diszjunkciókat összevonva kapjuk a következő oszlop elemi konjunkcióit, az összevonásban szereplő elemi diszjunkciókat pedig most is megjelöljük. Az újabb oszlopokat mindig jobbra írva az eljárást addig folytatjuk, amíg van mit összevonni.

Például az imént említett f esetén a módszer az alábbi táblázatot adja:

Elemi diszj.		Elemi diszj.		Elemi diszj.
0000	$\surd$	000−	$\surd$	−00−
0001	$\surd$	$00-0$		$1-0-$
0010	$\surd$	−000	$\surd$	
1000	$\surd$	−001	$\surd$	
1001	$\surd$	100−	$\surd$	
1100	$\surd$	$1-00$	$\surd$	
1101	$\surd$	$1-01$	$\surd$	
		110−	$\surd$	

Az eljárás végeztével rendelkezésünkre áll f egyszerűsített konjunktív normálformája, jelen esetben

$$f(X_1, X_2, X_3, X_4) = (\neg X_1 \vee \neg X_2 \vee \neg X_4) \wedge (\neg X_2 \vee \neg X_3) \wedge (X_1 \vee \neg X_3).$$

A minimalizáló eljárás egy lehetséges megvalósítása Java nyelven:

```
public void minimalizal() {
    String s, s1, s2;
    // Ha igaz, akkor megjelöltük, tudtuk egyszerűsíteni
    List<Boolean> termjel;
    // Ez lesz a következő oszlop
    List<String> termcopy;
    // A jelöletlen termek adják az eredményt
    Set<String> eredmeny = new HashSet<String>();
    // Amíg tudunk
```

```
// egyszerűsíteni
while (termek.size()>0) {
    termjel = new ArrayList<Boolean>();
    termcopy = new ArrayList<String>();
    for (int i = 0; i<termek.size(); ++i) {
        termjel.add(false);
    }
    for (int i = 0; i<termek.size(); ++i) {
        for (int j = 0; j<termek.size(); ++j) if (i!=j) {
            s1 = termek.get(i);
            s2 = termek.get(j);
            // Ha közöttük egy koordináta eltérés van
            if (hammingTavolsag(s1, s2) == 1) {
                // Megjelöljük őket
                termjel.set(i, true);
                termjel.set(j, true);
                for (int k = 0; k<s1.length(); ++k) {
                    if (s1.charAt(k)!=s2.charAt(k)) {
                        s = s1.substring(0,k) + "-" +
                        s1.substring(k+1,s1.length());
                        termcopy.add(s);
                    }
                }
            }
        }
    }
    // Amit nem jelöltünk meg, az a minimális formula
    // része lesz, vagyis bekerül az eredményhalmazba
    for (int i = 0; i<termek.size(); ++i) {
        if (!termjel.get(i)) {
            eredmeny.add(termek.get(i));
        }
    }
    // A termek változóba kerül a következő oszlop
```

```
            termek = new ArrayList<String>();
            for (int i = 0; i<termcopy.size(); ++i) {
                termek.add(termcopy.get(i));
            }
        }
}
```

Az igazságtáblázat azon soraiban, ahol a formula 0-t ad értékül, a kapott elemi diszjunkciók közül legalább egynek az értéke 0, így kapunk a konjunkcióval szintén 0-t.

X_1, X_2, X_3, X_4	$f(X_1, X_2, X_3, X_4)$	X_1, X_2, X_3, X_4	$f(X_1, X_2, X_3, X_4)$
0000	1	1000	1
0001	1	1001	1
0010	0	1010	1
0011	0	1011	1
0100	1	1100	1
0101	1	1101	0
0110	0	1110	0
0111	0	1111	0

Vizsgáljuk meg, hogy a változók mely értékei esetén hogyan alakulnak az egyes elemi diszjunkciók értékei:

X_1	1	1	0	1	0	0	0
X_2	1	1	1	1	1	0	0
X_3	0	1	1	1	1	1	1
X_4	1	1	0	0	1	0	1
$\neg X_1 \vee \neg X_2 \vee \neg X_4$	$\surd$	$\surd$					
$\neg X_2 \vee \neg X_3$		$\surd$	$\surd$	$\surd$	$\surd$		
$X_1 \vee \neg X_3$			$\surd$		$\surd$	$\surd$	$\surd$

Az oszlopok megfelelnek az eredeti formula igazságtáblájának a sorainak. $\surd$ jellel jelöltük a sorok azon oszlopait, ahol az adott elemi diszjunkció értéke 0. Minden oszlopban szerepelnie kell legalább egy $\surd$ jelnek. További egyszerűsítésre adhat lehetőséget, ha találunk olyan sort, amelyet kihagyhatunk anélkül, hogy bármelyik oszlop üresen maradna. Az ábráról leolvasható, hogy itt erre nincs mód, mindhárom elemi diszjunkció feltétlenül szükséges. Amennyiben a feladatunk bonyolultabb, több lehetőséget kaphatunk sorok elhagyására is, amely a változószám növelésével igen könnyen bekövetkezhet. Ilyen esetekben operációkutatási feladatot kapunk, melynek megoldására léteznek automatikus módszerek ugyan, de további költséges számításokat igényelnek.

Ehelyett választhatjuk az ún. *leggyorsabb esés* módszerét. Ez a közelítő módszer abból áll, hogy kiválasztjuk az (egyik) legtöbb oszlopot lefedő (azaz legtöbb $\surd$-t tartalmazó) sort. Ezután a kiválasztott sort és az általa lefedett oszlopokat elhagyjuk. És így tovább, előbb-utóbb a visszamaradó táblázat üres lesz. Az addig kiválasztott sorok halmaza lefedi a táblázatot. Az ilymódon nyert fedés nem feltétlenül a legrövidebb, de általában nem sokkal rosszabb annál. Tekintsünk egy példát, amelyben további egyszerűsítés lehetséges:

X	0	0	0	1	1
Y	0	0	1	0	1
Z	0	1	1	0	0
$X \vee Y$	$\surd$	$\surd$			
$Y \vee Z$	$\surd$			$\surd$	
$X \vee \neg Z$		$\surd$	$\surd$		
$\neg X \vee Z$				$\surd$	$\surd$

Az algoritmus minden egyes lépésében elhagyunk egy sort és egy, vagy több oszlopot. Az első elhagyott sor az $X \vee Y$, ami bekerül a végleges KNF-be. A hozzá tartozó oszlopok is eltűnnek.

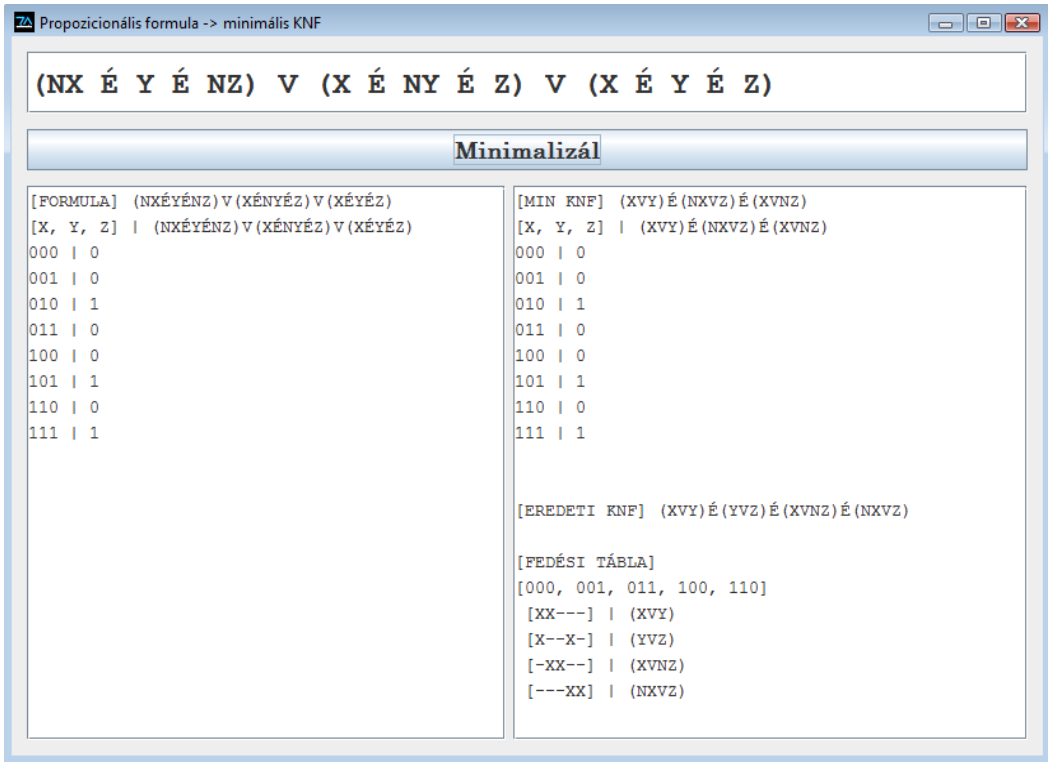
X	0	1	1
Y	1	0	1
Z	1	0	0
$Y \vee Z$		$\surd$	
$X \vee \neg Z$	$\surd$		
$\neg X \vee Z$		$\surd$	$\surd$

Innen a $\neg X \vee Z$-t, és a hozzá tartozó oszlopokat hagyjuk el.

X	0
Y	1
Z	1
$Y \vee Z$	
$X \vee \neg Z$	$\surd$

Az $X \vee \neg Z$, és egyetlen oszlopának elhagyásával kiürül a táblázat. A minimalizált KNF:

$$(X \vee Y) \wedge (\neg X \vee Z) \wedge (X \vee \neg Z).$$



A minimalizáló algoritmus a gyakorlatban: a lefedési tábla segítségével az eredetileg kapott négyből egy elemi diszjunkciótól megszabadulunk.

4. Kielégíthetetlenség-ellenőrzés DPM-mel

4.1. Davis és Putnam módszere

Martin Davis és Hilary Putnam amerikai matematikusok 1960-ban publikálták klózhalmaz kielégíthetetlenségét ellenőrző algoritmusukat, melynek nulladrendű változata az alábbi módon működik.

Legyen $\mathcal{S}$ a vizsgálandó formula, vagyis klózok halmaza. A Davis-Putnam-módszer (DPM) négy szabály alkalmazásával csökkenti $\mathcal{S}$ elemeinek számát. Ha $\mathcal{S}$ kielégíthető, akkor minden klóz törlődik. Ha kielégíthetetlen, akkor az üres klóz bekerül a halmazba.

1. **Tautológia szabály** (Tautology Rule): Egy klózt *tautológiának* nevezünk, ha tartalmaz komplemens literálpárt. Töröljük $\mathcal{S}$-ből az összes tautológiát. Ha a megmaradó $\mathcal{S}'$ klózhalmaz üres, akkor megállunk, egyébként az $\mathcal{S}'$ klózhalmazzal folytatjuk az eljárást.

2. **Egy-literál szabály** (One-Literal Rule): Ha $\mathcal{S}$-ben van L egységklóz, akkor $\mathcal{S}$-ből úgy kapjuk $\mathcal{S}'$-t, hogy $\mathcal{S}$-ből elhagyjuk az L-et tartalmazó klózokat. Ha $\mathcal{S}'$ üres, akkor megállunk, egyébként előállítjuk $\mathcal{S}''$-t az $\mathcal{S}''$-ből úgy, hogy töröljük $\neg L$-t $\mathcal{S}'$ klózaiból. Ha $\mathcal{S}$-ben volt $\neg L$ egységklóz, akkor ez a klóz $\neg L$ törlése után az üres klóz ($\square$) lesz, és ekkor is megállunk. Egyébként az $\mathcal{S}''$ halmazzal folytatjuk az eljárást.

3. **Tiszta-literál szabály** (Pure-Literal Rule): Azt mondjuk, hogy $\mathcal{S}$ egy klózában lévő L literál *tiszta* $\mathcal{S}$-ben, ha $\neg L$ nem fordul elő $\mathcal{S}$ egyetlen klózában sem. Ha L tiszta $\mathcal{S}$-ben, akkor $\mathcal{S}$-ből úgy kapjuk $\mathcal{S}'$-t, hogy $\mathcal{S}$-ből elhagyunk minden L-et tartalmazó klóztot. A megmaradó $\mathcal{S}'$ klózhalmazzal folytatjuk az eljárást (ha az nem üres).

4. **Szétvágási szabály** (Splitting Rule): Ezt a szabályt akkor alkalmazhatjuk, ha $\mathcal{S}$ klózai a következőképpen csoportosíthatók:

$$\{(A_1 \vee L), (A_2 \vee L), ..., (A_m \vee L)\}, \{(B_1 \vee L), (B_2 \vee L), ..., (B_n \vee L)\}, \mathcal{R},$$

ahol $\mathcal{R} = \{R_1, R_2, ..., R_s\}$, és az A_i, B_i, R_i klózok nem tartalmazzák sem L-t, sem $\neg L$-t. Legyen ekkor $\mathcal{S}_1 = \{A_1, A_2, ..., A_m\} \cup R$ és $\mathcal{S}_2 = \{B_1, B_2, ..., B_n\} \cup \mathcal{R}$. A kapott $\mathcal{S}_1$ és $\mathcal{S}_2$ klózhalmazokkal dolgozunk tovább.

A Davis-Putnam-eljárás helyes és teljes. Ez adódik a következő tételből.

4.1. tétel. *Egy $\mathcal{S}$ klózhalmaz akkor és csak akkor kielégíthetetlen, ha a Davis-Putnam-módszer szabályainak alkalmazásával $\mathcal{S}$-ből kapott klózhalmaz(ok) kielégíthetetlen(ek).*

Bizonyítás. Tekintsük a szabályokat egyesével.

1. A tautológiát minden hozzárendelés kielégíti, ezért $\mathcal{S}'$ pontosan akkor kielégíthetetlen, ha $\mathcal{S}$ kielégíthetetlen.

2. Ha $\mathcal{S}' = \{\}$, akkor $\mathcal{S}$ minden klóza tartalmazza az L egységklózt, így minden L-et igazzá tevő hozzárendelés kielégíti $\mathcal{S}$-t, így $\mathcal{S}$ kielégíthető.

   Meg kell mutatnunk továbbá, hogy az $\mathcal{S}''$ klózhalmaz akkor és csak akkor kielégíthetetlen, ha $\mathcal{S}$ kielégíthetetlen.

   (a) Ehhez először tegyük fel, hogy $\mathcal{S}''$ kielégíthetetlen. Ha $\mathcal{S}$ kielégíthető lenne, akkor lenne $\mathcal{S}$-t kielégítő $\mathcal{A}$ hozzárendelés. $\mathcal{A}$ tartalmazza L-t, mivel L egységklóz $\mathcal{S}$-ben. $\mathcal{A}$ kielégíti továbbá $\mathcal{S}$ minden, L-et nem tartalmazó klózát is. Mivel azonban $\mathcal{A}$-ban nem lehet benne $\neg L$, így kielégíti a $\neg L$-et tartalmazó klózokat is a $\neg L$ törlése után, így kielégíti $\mathcal{S}''$-t. Ezzel pedig ellentmondáshoz érkeztünk, $\mathcal{S}''$ nem lehetne kielégíthetetlen.

   (b) A másik irány vizsgálatához tegyük fel, hogy $\mathcal{S}$ kielégíthetetlen. Ha $\mathcal{S}''$ kielégíthető lenne, akkor lenne őt kielégítő A'' hozzárendelés. Mivel sem L, sem $\neg L$ nem szerepel $\mathcal{S}''$-ben, választhatunk olyan A''-t, hogy benne L szerepel. Ez az A'' kielégíti $\mathcal{S}''$-t. De ez ellentmond a feltételezésnek, mely szerint $\mathcal{S}$ kielégíthetetlen.

3. $\mathcal{S}'$ akkor és csak akkor kielégíthetetlen, ha $\mathcal{S}$ kielégíthetetlen.

   (a) Tegyük fel, hogy $\mathcal{S}'$ kielégíthetetlen. Ekkor $\mathcal{S}$ biztos, hogy kielégíthetetlen, mivel $S' \subseteq S$.

(b) A másik irányt vizsgálva tegyük fel, hogy $\mathcal{S}$ kielégíthetetlen. Ha $\mathcal{S}'$ kielégíthető lenne, akkor lenne $\mathcal{S}$-t kielégítő $\mathcal{A}$ hozzárendelés. Mivel $\mathcal{S}$-ben nem fordul elő sem L, sem $\neg L$, ezért $\mathcal{A}$-ban L vagy $\neg L$ is szerepelhet. Válasszuk azt az $\mathcal{A}$ hozzárendelést, melynek eleme L. De $\mathcal{S}'$-nek egy ilyen $\mathcal{A}$ hozzárendelése kielégíti $\mathcal{S}$-t is. Ezzel pedig ellentmondáshoz érkeztünk azzal szemben, hogy $\mathcal{S}$ kielégíthetetlen.

4. $\mathcal{S}$ akkor és csak akkor kielégíthetetlen, ha $\mathcal{S}_1$ és $\mathcal{S}_2$ is kielégíthetetlenek.

(a) Tételezzük fel, hogy $\mathcal{S}$ kielégíthetetlen. Ha $\mathcal{S}_1$ vagy $\mathcal{S}_2$ kielégíthető lenne, akkor létezne olyan $\mathcal{A}$ hozzárendelés, amely azt kielégíti. Mivel $\mathcal{S}_1$ és $\mathcal{S}_2$ klózai nem tartalmazzák sem L-t, sem $\neg L$-t, így az $\mathcal{A}$ hozzárendelésben L vagy $\neg L$ is szerepelhet. Ha $\mathcal{A}$ az $\mathcal{S}_1$-et elégíti ki, akkor L szerepeltetésével, ha az $\mathcal{S}_2$-t, úgy a $\neg L$ szerepeltetésével $\mathcal{A}$ kielégíti $\mathcal{S}$-t. Ez ellentmond annak, hogy $\mathcal{S}$ kielégíthetetlen, tehát $\mathcal{S}_1$ és $\mathcal{S}_2$ is kielégíthetetlenek.

(b) A másik lehetőség szerint tegyük fel, hogy $\mathcal{S}_1$ és $\mathcal{S}_2$ is kielégíthetetlenek. Ha $\mathcal{S}$ kielégíthető lenne, akkor lenne olyan $\mathcal{A}$ hozzárendelés, amely kielégíti $\mathcal{S}$-et. Amennyiben $\mathcal{A}$-nak eleme L, akkor $\mathcal{S}_1$-et, ha eleme $\neg L$, akkor pedig $\mathcal{S}_2$-t elégítené ki. Ez ellentmond annak, hogy $\mathcal{S}_1$ és $\mathcal{S}_2$ is kielégíthetetlenek. Így $\mathcal{S}$ kielégíthetetlen.

□

Tekintsünk néhány gyakorlati példát.

4.2. Alkalmazás

4.2. példa. Alkalmazzuk a módszert az $\mathcal{S} = \{\{X, Y, \neg Z\}, \{X, \neg Y\}, \{\neg X\}, \{Z\}, \{U\}\}$ klózhalmazra.

1. $\mathcal{S}' = \{\{Y, \neg Z\}, \{\neg Y\}, \{Z\}, \{U\}\}$ [2. szabály $\neg X$-szel]
2. $\mathcal{S}'' = \{\{\neg Z\}, \{Z\}, \{U\}\}$ [2. szabály $\neg Y$-nal]
3. $\mathcal{S}''' = \{\square, \{U\}\}$ [2. szabály $\neg Z$-vel]

$\mathcal{S}$ kielégíthetetlen, mivel $\mathcal{S}'''$ már tartalmazza az üres klózt.

4.3. példa. Vizsgáljuk az $\mathcal{S} = \{\{X, Y\}, \{\neg Y\}, \{\neg X, Y, \neg Z\}\}$ klózhalmazt.

1. $\mathcal{S}' = \{\{X\}, \{\neg X, \neg Z\}\}$ [2. szabály $\neg Y$-nal]
2. $\mathcal{S}'' = \{\{\neg Z\}\}$ [2. szabály X-szel]
3. $\mathcal{S}''' = \{\}$ [2. szabály $\neg Z$-vel]

$\mathcal{S}$ kielégíthető, mivel $\mathcal{S}'''$ üres.

4.4. példa. Vizsgáljuk az $\mathcal{S} = \{\{X, \neg Y\}, \{\neg X, Y\}, \{Y, \neg Z\}, \{\neg Y, \neg Z\}\}$ klózhalmazt.

1. $\mathcal{S}'_1 = \{\{\neg Y\}, \{Y, \neg Z\}, \{\neg Y, \neg Z\}\}$, [4. szabály X-szel]
   $\mathcal{S}'_2 = \{\{Y\}, \{Y, \neg Z\}, \{\neg Y, \neg Z\}\}$
2. $\mathcal{S}''_1 = \{\{\neg Z\}\}$, $\mathcal{S}''_2 = \{\{\neg Z\}\}$ [2. szabály Y-nal és $\neg Y$-nal]
3. $\mathcal{S}'''_1 = \{\}$, $\mathcal{S}'''_2 = \{\}$ [2. szabály $\neg Z$-vel]

$\mathcal{S}'_1$ és $\mathcal{S}'_2$ kielégíthető, így $\mathcal{S}$ is az.

4.5. példa. Legyen $\mathcal{S} = \{\{X, \neg Y\}, \{X, Y\}, \{Y, Z\}, \{\neg Y, Z\}\}$.

1. $\mathcal{S}' = \{\{Y, Z\}, \{\neg Y, Z\}\}$ [3. szabály $\neg X$-szel]
2. $\mathcal{S}'' = \{\}$ [3. szabály Z-vel]

Tehát $\mathcal{S}$ kielégíthető.

5. Következtetéshelyesség-ellenőrzés rezolúcióval

5.1. Rezolúciós kalkulus

Legyenek $A_1, A_2, ..., A_n, B$ $(n \geq 1)$ tetszőleges ítéletlogikai formulák. Ekkor az ítéletlogika eldöntésproblémájának kétféle megfogalmazása:

1. az $A_1 \supset A_2 \supset ... \supset A_n \supset B$ formula tautológia-e, illetve
2. az $\{A_1, A_2, ..., A_n, \neg B\}$ formula kielégíthetetlen-e.

Írjuk át az utóbbi halmaz formuláit konjuktív normálformába. Az így kapott

$$\{KNF_{A_1}, KNF_{A_2}, ..., KNF_{A_n}, KNF_{\neg B}\}$$

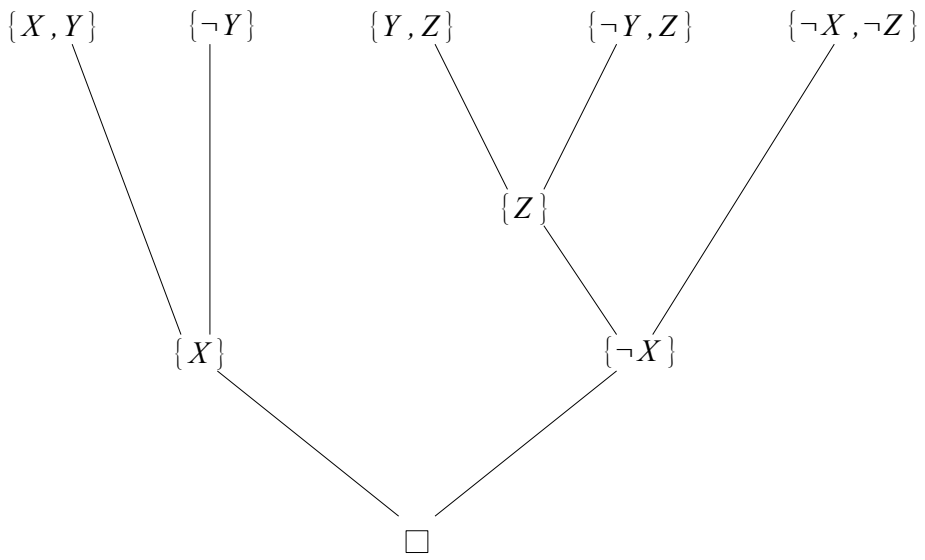
formulahalmaz pontosan akkor kielégíthetetlen, ha a halmaz formuláiban szereplő klózok halmaza kielégíthetetlen. A szemantikus eldöntésprobléma tehát klózhalmaz kielégíthetetlenségére vezethető vissza. Ez utóbbi UNSAT-probléma néven ismert.

A nulladrendű rezolúciós elv egy szintaktikai alapon működő döntési elv, amellyel igazolhatjuk egy klózhalmaz, illetve tetszőleges formula vagy formulahalmaz kielégíthetetlenségét, ezáltal megoldva a szemantikus eldöntésproblémát. A rezolúcióban használt klózokra vonatkozó egyszerűsítési szabály kimondja, hogy

$$(X \vee \mathcal{C}) \wedge (\neg X \vee \mathcal{C}) \sim \mathcal{C},$$

ahol X atomi formula, $\mathcal{C}$ pedig egy X-et nem tartalmazó klóz. Az X és a $\neg X$ klózok konjunkciójának egyszerűsítése az üres klóz ($\square$). A rezolúciós kalkulus eldöntésproblémája megállapodás szerint az, hogy levezethető-e $\mathcal{S}$-ből az üres klóz, vagyis hogy $\mathcal{S}$-nek van-e rezolúciós cáfolata.

5.1. példa. Kielégíthetetlen-e az $\{\{X, Y\}, \{Y, Z\}, \{\neg Y, Z\}, \{\neg X, \neg Z\}, \{\neg Y\}\}$ formula? Amennyiben igen, rajzoljuk fel az üres klóz egy levezetését.



Levezetési fa

A levezetési fa levélelemei találhatóak fent, gyökere, a levezetett klóz pedig lent. A levélelemek a formula klózhalmazának elemei. A rezolúciós levezetést adó klózsorozat így bármely olyan sorozata lesz az ábrában szereplő klózoknak, amelyben minden klózt megelőznek a gyermekei (ha vannak), például

$$\{X, Y\}, \{\neg Y\}, \{Y, Z\}, \{\neg Y, Z\}, \{\neg X, \neg Z\}, \{X\}, \{Z\}, \{\neg X\}, \square.$$

5.2. Előre- és visszakövetkeztetés

A rezolúciós kalkulus a tételbizonyítási feladatot a visszakövetkeztetés módszerével oldja meg, hiszen az általa használt klózhalmaz a tételformula negáltját tartalmazza. Ugyanakkor előrekövetkeztetés is végezhető a segítségével. Legyen $\{A_1, A_2, A_3, ..., A_n\}$ a feltételformulák halmaza és B a tételformula.

Visszakövetkeztetés: Amennyiben $\{A_1, A_2, A_3, ..., A_n, \neg B\}$-nek létezik rezolúciós cáfolata, úgy $\{A_1, A_2, A_3, ..., A_n\} \models B$.

Előrekövetkeztetés: A feltételformulák halmazából alkotott klózhalmazból való rezolúciós levezetés minden klóza szemantikus következménye az $\{A_1, A_2, A_3, ..., A_n\}$ halmaznak. A rezolúciós levezetés definíciója nem tartalmazott utalást arra, hogy milyen módon állítsuk elő azt. Tekintsünk párat a lehetséges stratégiák közül.

5.3. Lineáris rezolúciós kalkulus

5.2. definíció. Egy $\mathcal{C}$ klóznak $\mathcal{S}$ formulából történő *lineáris (rezolúciós) levezetésének vagy bizonyításának* egy $\langle \mathcal{C}_0, \mathcal{B}_0 \rangle, ..., \langle \mathcal{C}_n, \mathcal{B}_n \rangle$ klózpársorozatot nevezünk, ahol $\mathcal{C} = \mathcal{C}_{n+1}$ és

1. $\mathcal{C}_0$ és minden $\mathcal{B}_i$ eleme $\mathcal{S}$-nek, vagy valamely $\mathcal{C}_j$-nek, ahol $j < i$,
2. minden $\mathcal{C}_{i+1}$ (ahol $i \leq n$) $\mathcal{C}_i$ és $\mathcal{B}_i$ rezolvense.

Egy $\mathcal{C}$ klóz *lineárisan levezethető (bizonyítható)* $\mathcal{S}$-ből ($\mathcal{S} \vdash_L \mathcal{C}$), ha $\mathcal{C}$-nek létezik lineáris levezetése $\mathcal{S}$-ből. $\mathcal{S}$ *lineárisan cáfolható*, ha $\mathcal{S} \vdash_L \square$. Az $\mathcal{S}$-ből lineárisan levezethető összes klóz halmazát $\mathcal{L}(\mathcal{S})$-sel jelöljük.

5.3. példa. A

$$\mathcal{C} = \{\neg V\}$$

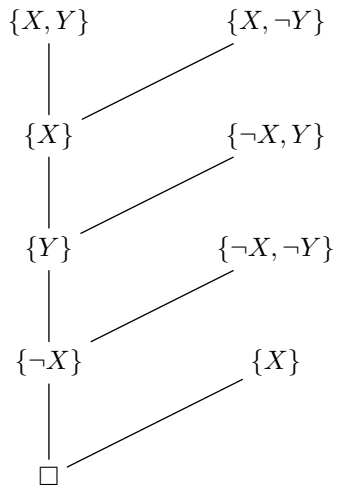
klóz levezetése az

$$\mathcal{S} = \{\ \{\neg X, Y\}, \{\neg Y, Z\}, \{X, V\}, \{\neg V, Y, Z\}, \{\neg Z\}\}$$

klózhalmazból:

1.	$\{\neg V, Y, Z\}$	$[\in \mathcal{S}]$
2.	$\{\neg Z\}$	$[\in \mathcal{S}]$
3.	$\{\neg V, Y\}$	[1, 2 rezolvense]
4.	$\{\neg Y, Z\}$	$[\in \mathcal{S}]$
5.	$\{\neg Y\}$	[2, 4 rezolvense]
6.	$\{\neg V\}$	[3, 5 rezolvense]

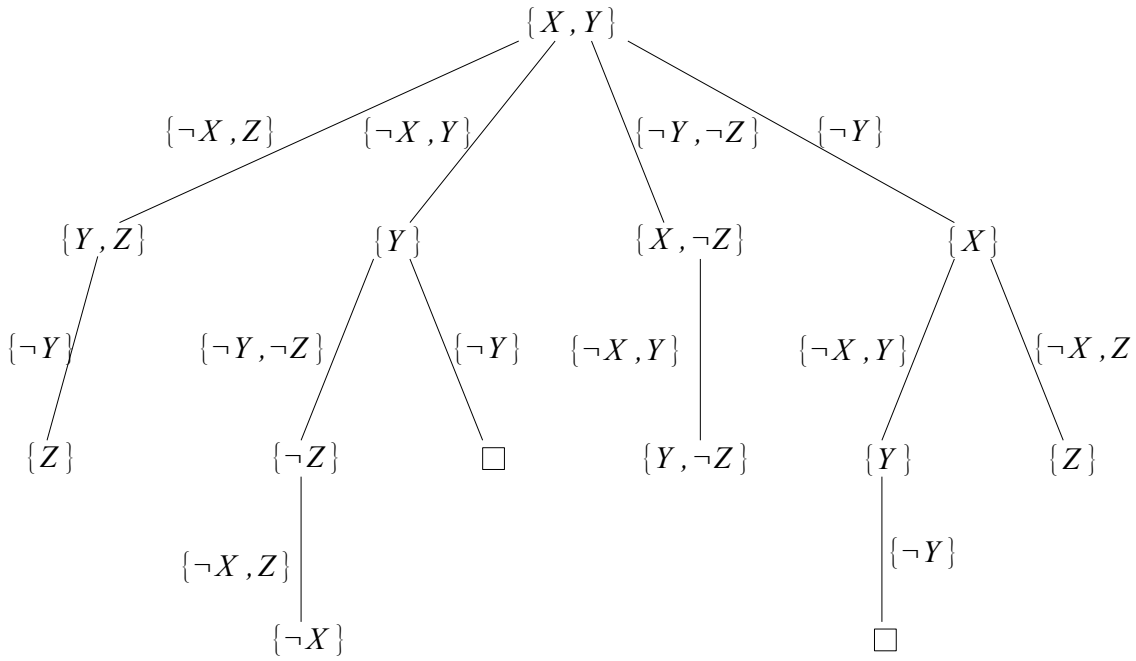
A lineáris rezolúciós kalkulus annyiban egyszerűsíti számunkra az eredeti rezolúciót, hogy amíg ott a levezetést egy fával ábrázolhattuk, és több ágról érkező klózokat rezolváltunk egymással, addig itt a levezetést klózok egymásból rezolvált sorozataként adjuk meg, hozzátéve, hogy melyik eredeti formulabeli vagy korábban kapott klóz segítségével történt a rezolválás. Tekintsünk egy példát:
Legyen $\mathcal{S} = \{\{X, Y\}, \{X, \neg Y\}, \{\neg X, Y\}, \{\neg X, \neg Y\}\}$. Ekkor $\mathcal{S}$ egy lineáris rezolúciós cáfolata az alábbi:



A bal oldalon szereplő klózokat *centrális klózoknak*, a jobb oldaliakat *mellékklózoknak* nevezzük.

Szokás a lineáris rezolúciót úgy is ábrázolni, hogy a mellékklózokat nem külön csúcsokként, hanem a centrális klózok közötti élek címkéiként tüntetjük fel. Ezáltal az egy klózból induló összes lineáris rezolúciós levezetést egy fával ábrázolhatjuk, amelyet *teljes levezetési fának* nevezünk. Tekintsünk egy példát!

5.4. példa. Kielégíthetetlen-e az $\{\{X, Y\}, \{\neg X, Y\}, \{\neg Y, \neg Z\}, \{\neg X, Z\}, \{\neg Y\}\}$ formula? Rajzoljuk fel a teljes levezetési fának egy olyan részét, amely választ ad a kérdésre.



Részlet a teljes levezetési fából: az üres klózba (□) vezető utak egy-egy rezolúciós cáfolatot jelentenek

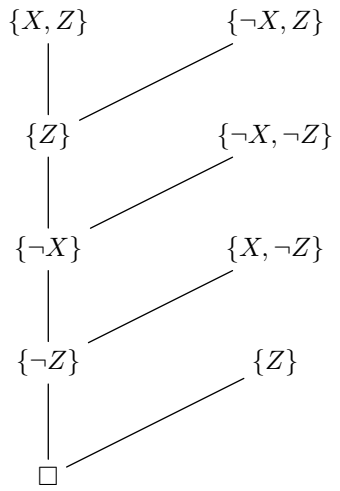
Mivel a rezolúciós kalkulus teljes, ezáltal a lineáris rezolúciós kalkulus is az. A Prolog, és más logikai programozási nyelvek azonban a rezolúciós levezetéshez egy másik, nem teljes stratégiát használnak. Lássuk, hogy miért.

5.4. Lineáris inputrezolúciós kalkulus

5.5. definíció. Egy $\mathcal{C}$ klóznak $\mathcal{S}$ formulából történő *lineáris inputrezolúciós levezetésének vagy bizonyításának* egy $\langle \mathcal{C}_0, \mathcal{B}_0 \rangle, ..., \langle \mathcal{C}_n, \mathcal{B}_n \rangle$ klózpársorozatot nevezünk, ahol $\mathcal{C} = \mathcal{C}_{n+1}$ és

1. $\mathcal{C}_0$ és minden $\mathcal{B}_i$ eleme $\mathcal{S}$-nek,
2. minden $\mathcal{C}_{i+1}$ (ahol $i \leq n$) $\mathcal{C}_i$ és $\mathcal{B}_i$ rezolvense.

5.6. példa (Ábrázolás). A szokásos jelölés szerint a lineáris rezolúciót úgy ábrázoljuk, hogy a kezdőpontot tesszük felülre, és a konklúzió kerül alulra.



A lineáris inputrezolúciós kalkulus annyi megkötést tesz a lineáris rezolúciós kalkulussal szemben, hogy a mellékklózok kizárólag a formula elemei lehetnek. Beláthatjuk, hogy ez a stratégia nem teljes. A rezolúciós cáfolathoz az üres klózt kell levezetnünk a kezdeti klózhalmazból. Az utolsó lépése ennek az, hogy egy egységklózzal rezolválunk. Mivel itt ezt csak a formulánkból vehetjük, amennyiben a formula nem tartalmaz egységklózt, ezzel a stratégiával nem tudjuk létrehozni a rezolúciós cáfolatot.

Tudunk ugyanakkor definiálni klózoknak egy olyan osztályát, melyre a lineáris inputrezolúciós kalkulus teljes.

5.7. definíció. Azt a klózt, amely legfeljebb egy pozitív literált tartalmaz, *Horn-klóznak* nevezzük.

5.8. definíció. Azt a klózt, amely pontosan egy pozitív literált tartalmaz, *programklóznak* nevezzük.
($\{X, \neg Y_1, \neg Y_2, ..., \neg Y_n\}$ Prolog jelöléssel: $X :- Y_1, Y_2, ..., Y_n.$).

5.9. definíció. Azt a programklózt, amelyben vannak negatív literálok is, *szabálynak* nevezzük.
(Az előző jelölésben ekkor $n > 0$).

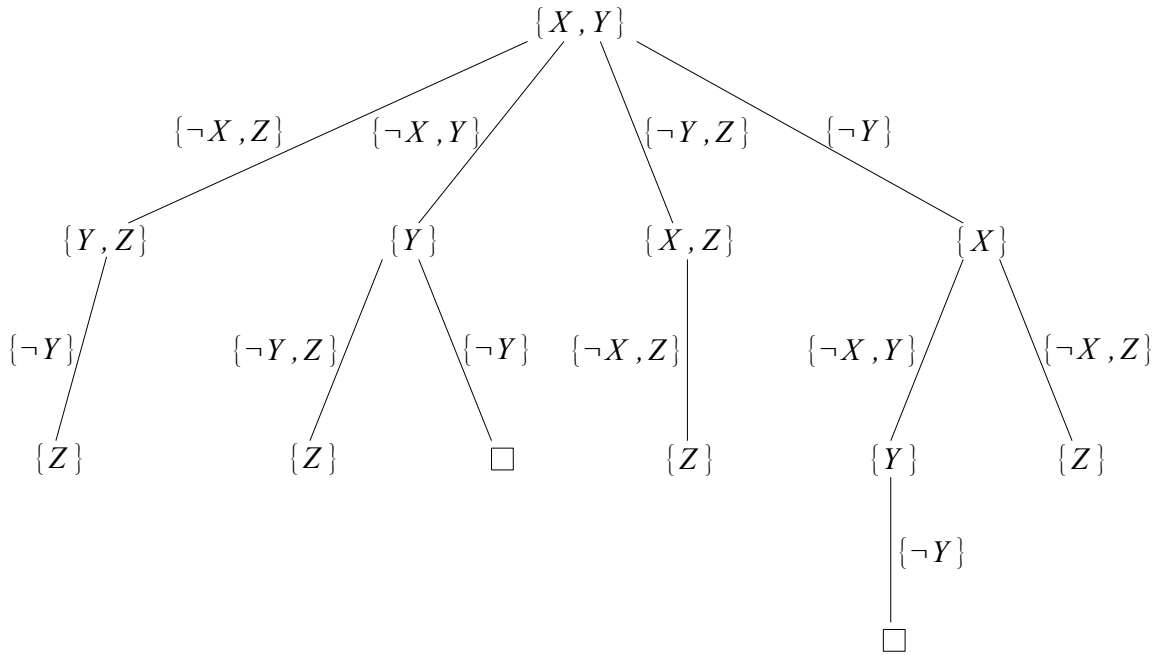
5.10. definíció. Azt a programklózt, amelyben nincs negatív literál, *ténynek vagy egységklóznak* nevezzük.
($\{X\}$ Prolog jelöléssel: $X.$ vagy $X :- .$).

5.11. definíció. Azt a klózt, amelyben nincs pozitív literál, *célklóznak* nevezzük. ($\{\neg Y_1, \neg Y_2, ..., \neg Y_n\}$-t Prologban kérdésként fogalmazzuk meg: $? - Y_1, Y_2, ..., Y_n.$).

5.12. definíció. Egy *Prolog program* olyan klózhalmaz, amelynek elemei programklózok (tények vagy szabályok).

Annak az oka, hogy a Prolog a lineáris inputrezolúciós stratégiát használhatja, az, hogy egyúttal azt is megköveteli, hogy Horn-klózokkal dolgozzunk. A lineáris inputrezolúciós kalkulus teljes a Horn-klózokra nézve. Tekintsünk egy példát, amely ilyen klózokkal dolgozik.

5.13. példa. Kielégíthetetlen-e a $\{\{Z\}, \{\neg X, Y\}, \{\neg Y, Z\}, \{\neg X, Z\}, \{\neg Y\}\}$ formula? Rajzoljuk fel a teljes levezetési fának egy olyan részét, amely választ ad a kérdésre.



Részlet a teljes levezetési fából: az üres klóz (□) itt is levezethető

A lineáris inputrezolúciós kalkulus implementálása során fel kell építenünk a teljes levezetési fát. Amennyiben üres klózhoz érünk, az eljárásnak vége.

A mellékelt *rezolucio* program az egyes állapotok közötti operátorokként értelmezi az adódó rezolválási lehetőségeket. A fa felépítéséhez a program Horn-klózait (melyeket úgy adunk meg, hogy a bennük szereplő egyetlen negált literállal kezdődnek) ABC-sorrendbe rendezi. A kérdésben szereplő klózt ezután az egyes literálok szerinti blokkok alapján próbálja rezolválni.

5.5. Alkalmazás

A rezolúciós kalkulust tételbizonyításra tudjuk használni, bizonyos feltételek felírása után. A feltételeket és a tételt formalizálnunk kell, a feltételformulákat illetve a tételformula negáltját ezután konjunktív normálformára kell alakítani. A konjunktív normálformákkal egy klózhalmazt kapunk. Amennyiben ebből a rezolúció segítségével le tudjuk vezetni az üres klózt, a tételt bebizonyítottuk. A levezetést kezdjük azzal a klózzal, melyet a tételformula negáltjából kaptunk.
Lássunk most egy példát a *rezolucio* program használatával megoldható feladatra, mely inputrezolúciós kalkulus segítségével hoz döntést. A feladatot fel kell tudnunk írni Horn-klózok segítségével. A szoftver képernyőjén a program a felső szövegdobozba kerül, a kérdést pedig a középső sorban kell feltenni.

5.14. példa. A feltételeink legyenek a következőek:

1. Bori akkor megy a koncertre, ha Attila és Flóri is mennek.
2. Attila akkor megy a koncertre, ha Flóri és Tomi is mennek.
3. Peti akkor megy, ha Csaba is megy.
4. Tomi megy, ha Flóri vagy Csaba is elmegy.
5. Flóri mindenképpen ott lesz a koncerten.

Döntsük el, hogy igaz-e az alábbi két tétel a fenti feltételek alapján:

Bori megy a koncertre.

Peti és Tomi is mennek a koncertre.

Látható, hogy a feltételek implikációit olyan klózokként tudjuk felírni, melyekben egy pozitív literál szerepel:

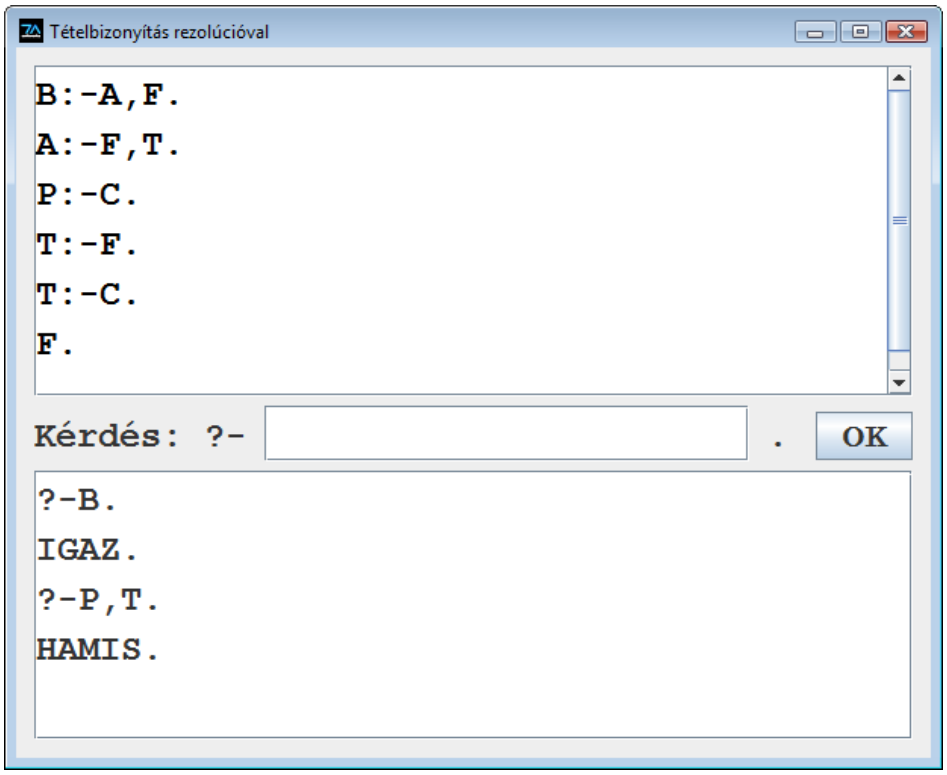
1. $B \vee \neg A \vee \neg F$
2. $A \vee \neg F \vee \neg T$
3. $P \vee \neg C$

A 4. feltételből két klózt kapunk:

4. $T \vee \neg F$
5. $T \vee \neg C$

Az utolsó feltétel pedig tényként szerepel:

6. F



A program megoldása: az első tételt be tudjuk bizonyítani, a másodikat nem.

6. Összefoglalás

Szakdolgozatomban nulladrendű logikai formulák halmazának kielégíthetetlenségét bizonyító, illetve cáfoló módszereket vizsgáltam. Ezek segítségével megmutathatjuk logikai következtetéseink helyességét. A módszerek előkészítéséhez szükség volt a formulák konjunktív normálformájú alakjaira, melyeket normálformára hozó algoritmusokkal állítottunk elő. Bemutattam a kanonikus konjunktív normálforma létrehozásának algoritmusát, amelyet a mellékelt programban is beépítettem. Részleteztem a szintaktikai átalakításhoz szükséges lépéseket is. A normálforma minimalizálására a Quine-McCluskey-algoritmust választottam és implementáltam, mint a legtöbbet idézett egyszerűsítési eljárást.

Bemutattam a Davis-Putnam-algoritmus működését, ami ma is gyakran használt algoritmus klózhalmaz kielégíthetetlenségi problémájának eldöntésére. Ezen eljárás során a kezdeti klózhalmazt alakítjuk, és a halmaz kielégíthetetlenségét az jelzi, hogy az üres klóz bekerül abba.

A dolgozat harmadik témája a rezolúciós eljárás volt. Láttuk a rezolúciós kalkulus működési elvét, mely módszer végrehajtása során az üres klóz levezetése jelezte az eredeti formulahalmaz kielégíthetetlenségét. A levezetést sorozatként definiáltuk, ennek létrehozására pedig különböző stratégiákat használtunk, mint a lineáris rezolúció, illetve a gyakorlatban is tekintett lineáris inputrezolúciós kalkulus.

További tanulmányaim során tervezem ezen tételbizonyító eljárások vizsgálatát az elsőrendű logikában. Az elsőrendű logikában az univerzálisan és egzisztenciálisan kvantált formulák megjelenésével a szemantikai kérdések vizsgálata (így egy adekvát kalkulus definiálása) egy nagyságrenddel nehezebb. A rezolúciós kalkulushoz szükséges előkészítő algoritmusok (Skolem-normálforma előállítása), elsőrendű rezolúciós szabály alkalmazásához szükséges illesztő helyettesítések hatékony megkeresése terveim szerint az MSc diplomamunkám tárgyát képezhetik.

7. Köszönetnyilvánítás

Ezúton szeretném megköszönni **dr. Várterész Magda** egyetemi docensnek, hogy témavezetőmként tengernyi dolga között lelkiismeretesen időt szánt rám és foglalkozott velem, szakmai, tanulmányi, pályaválasztási tanácsokkal segített, könyvekkel, publikációkkal és egyéb kutatási ismeretanyagokkal látott el, és mentorként terelgette pályámat. Nélküle ez a szakdolgozat ilyen formában biztosan nem készülhetett volna el.

Köszönöm **Kósa Márknak**, hogy szakmai tanácsaival segítette a szakdolgozat nyomdatechnikai kivitelezését, és hogy bármikor lehet rá számítani.

Köszönöm **Espák Miklósnak**, hogy megtanított programozni C-ben és Javában, és mindig segítőkészen állt a problémáinkhoz. Nélküle a szakdolgozathoz csatolt CD-n lévő szoftver nem teljesítene ilyen fényesen.

Köszönöm az összes további oktatómnak, akik az elmúlt 3 évben tanítottak, hiszen mindannyian hozzájárultak az egyes tárgyak és tudományterületek megismeréséhez, a közöttük lévő kapcsolatok megvilágításához, ahogy erkölcsi és szakmai fejlődésünkhöz is.

Köszönöm **Tomán Henriettának**, hogy a mi nyelvünkre fordította a diszkrét matematikát, **dr. Daróczy Zoltánnak**, hogy egy reggel 8 órai kalkuluselőadást is fel tudott dobni a történeteivel, **dr. Juhász Istvánnak**, hogy gondolatébresztőivel tette még érdekesebbé a programozás órákat, és **dr. Tornai Róbertnek**, hogy ennyi érdekes dolgot bele tudott zsúfolni egyetlen kurzusba.

Köszönöm továbbá **édesapámnak**, hogy finanszírozta és támogatta a tanulmányaimat, **édesanyámnak** és **nagymamámnak**, hogy hazavártak és finomakat sütöttek-főztek. Köszönöm **nővéremnek**, hogy erkölcsileg támogatott, és hogy 4 évesen megtanított olvasni, megalapozva ezzel tudományos pályámat.

Végül, de nem utolsósorban köszönöm a programtervező informatikus barátaimnak, hogy együtt könnyebben tudtuk venni az akadályokat. Köszönöm **Nagy István Zoltánnak**, hogy megannyiszor szakmai segítséget nyújtott, köszönet továbbá **Andrejcsik Tamásnak**, **Zelenák Zoltánnak**, **Szauervein Szabolcsnak**, **Csillag Péternek** és **Máté Balázsnak** a segítségükért.

Irodalomjegyzék

[1] Anil Nerode–Richard A. Shore: *Logic for Applications*, New York, 1996, Springer.

[2] dr. Pásztorné Varga Katalin, dr. Várterész Magda: *A matematikai logika alkalmazásszemléletű tárgyalása*, Budapest, 2003, Panem.

[3] Czédli Gábor: *Boole-függvények*, Szeged, 1995, Polygon.

[4] Dirk Louis, Peter Müller: *Java 5*, Budapest, 2006, Panem.

[5] dr. Várterész Magda: *Matematikai logika jegyzet.*

[6] dr. Pásztorné Varga Katalin, dr. Várterész Magda: *Számítástudomány, logika, informatikaoktatás* 2005.

[7] Robert Kowalski: *http://www.doc.ic.ac.uk/∼rak/history.html.*

A. Függelék - Kódok

A.1. minimal

A **minimal** programhoz tartozó Java osztályok.

```
public class Formula {
    private LogElem fa; // a felépítendő fa gyökere
    private String fedes = ""; // lekérdezhető fedési tábla
    private String f, // a formula
            elsoF; // a fedési tábla nélkül kapott
    // minimalizált alak
    private List<String> termek = new ArrayList<String>();
    private boolean tau, // Tautológia,
            ell; // ellentmondás ellenőrzésére.
    public String getFedes() {return fedes;}
    public String getElsoF() {return elsoF;}
    public String getF() {return f;}
    public void setF(String f) {this.f = f;}
    public Formula() {super(); fa = null;}
    public Formula(String formula) {this(); this.f = formula;}
    public void minimalizal() {
        String s, s1, s2;
        List<Boolean> termjel; // Ha igaz, akkor megjelöltük
        List<String> termcopy; // Ez lesz a következő oszlop
        List<String> termek2; // A további minimalizáláshoz mentjük
        List<String> lt = new ArrayList<String>(); // lefedési tábla
        List<String> eredmeny = new ArrayList<String>();
        // A jelöletlen termek adják az eredményt
        // Másolatot készítünk a termekről
        termek2 = new ArrayList<String>();
        for (int i = 0; i < termek.size(); ++i) {
            termek2.add(termek.get(i));
        }
        // Amíg tudunk egyszerűsíteni
        while (termek.size() > 0) {
            termjel = new ArrayList<Boolean>();
            termcopy = new ArrayList<String>();
            for (int i = 0; i < termek.size(); ++i) {
                termjel.add(false);
            }
            for (int i = 0; i < termek.size(); ++i) {
                for (int j = 0; j < termek.size(); ++j) {
                    if (i != j) {
                        s1 = termek.get(i);
                        s2 = termek.get(j);
                        // Ha a két term között egy
                        // koordináta eltérés van
```

```
                    if (hammingTavolsag(s1, s2) == 1) {
                        // Megjelöljük őket
                        termjel.set(i, true);
                        termjel.set(j, true);
                        for (int k = 0; k < s1.length(); ++k) {
                            if (s1.charAt(k) != s2.charAt(k)) {
                                if (k + 1 == s1.length()) {
                                 s = s1.substring(0, k) + "-";
                                } else {
                                 s = s1.substring(0, k) + "-" +
                                 s1.substring(k+1, s1.length());
                                }
                                termcopy.add(s);
                            }
                        }
                    }
                }
            }
        }
        // Amit nem jelöltünk meg, az a
        // minimális formula része lesz,
        // vagyis bekerül az eredményhalmazba
        for (int i = 0; i < termek.size(); ++i) {
            if (!termjel.get(i)) {
                if (!eredmeny.contains(termek.get(i))) {
                    eredmeny.add(termek.get(i));
                }
            }
        }
        // A termek változót feltöltjük
        // a következő oszlop tartalmával
        termek = new ArrayList<String>();
        for (int i = 0; i < termcopy.size(); ++i) {
            termek.add(termcopy.get(i));
        }
    }
    f = "";
    boolean elso;
    String ed; // elemi diszjunkció
    fedes = ""; // nullázzuk a fedési táblát
    fedes = Arrays.deepToString(termek2.toArray()) + "\n";
    if (tau) {
        elsoF = f = "1";
        return;
    } else if (ell) {
        elsoF = f = "0";
        return;
    } else {
        for (int i = 0; i < eredmeny.size(); ++i) {
            ed = "";
```

```
            ed += "(";
            elso = true;
            for (int j = 0; j < LogElem.betuk.size(); ++j) {
                if (eredmeny.get(i).charAt(j) != '-') {
                    if (!elso) {
                        ed += ((char) LogElem.CVAGY);
                    }
                    if (eredmeny.get(i).charAt(j) == '1') {
                        ed += "N";
                    }
                    ed += LogElem.betuk.get(j);
                    elso = false;
                }
            }
            ed += ")";
            lt.add(ed); // 2 új sor a lefedési táblába
            lt.add("");
            // Értékeljük ki az elemi diszjunkciót
            try {
                this.fa = this.fatEpit(ed);
            } catch (SyntaxException ex) {
            }
            for (int j = 0; j < termek2.size(); ++j) {
                // Változóhozzárendelés
                LogElem.setErtek(termek2.get(j));
                lt.set(lt.size() - 1, lt.get(lt.size() - 1) +
                ((!fa.erteke()) ? "X" : "-"));
            }
            fedes+="\t[" + lt.get(lt.size()-1) + "]\t|\t" +ed+ "\n";
            if (i < eredmeny.size() - 1) {
                ed += (char) LogElem.CES;
            }
            f += ed;
        }
    }
    elsoF = f;
    eredmeny = new ArrayList<String>();
    int leg = 0, legh;  // megkeressük a legtöbb X-szel jelölt
                        // sort a lefedési táblában
    while (!(lt.isEmpty() || lt.get(1).equals(""))) {
        legh = 0;
        for (int i = 0; i < lt.size() - 1; i += 2) {
            int xek = 0;
            for (int j = 0; j < lt.get(i + 1).length(); ++j) {
                if (lt.get(i + 1).charAt(j) == 'X') {
                    xek++;
                }
            }
            if (xek > legh) {
                legh = xek;
```

```
                leg = i;
            }
        }
        eredmeny.add(lt.get(leg));
        // Kitöröljük a leghosszabb sor által lefedett oszlopokat
        for (int i = 0; i < lt.size() - 1; i += 2) {
            if (i != leg) {
                String uj = "";
                for (int j=0; j<lt.get(i + 1).length(); ++j) {
                    if (lt.get(leg + 1).charAt(j) != 'X') {
                        uj += (char) lt.get(i + 1).charAt(j);
                    }
                }
                lt.set(i + 1, uj);
            }
        }
        // Kitöröljük a sort (amit 2 sorral reprezentálunk)
        lt.remove(leg);
        lt.remove(leg);
    }
    // Felírjuk az eredményformulát
    f = "";
    for (int i = 0; i < eredmeny.size(); ++i) {
        f += eredmeny.get(i) + ((i == eredmeny.size() - 1) ?
             "" : (char) LogElem.CES);
    }
    // Újraépítjük a fát
    try {
        fa = fatEpit(f);
    } catch (SyntaxException ex) {
    }
    // Eltüntetjük a fölösleges zárójeleket,
    // ha úgy alakult, hogy keletkeztek
    while (folosZarojel(f)) {
        f = f.substring(1, f.length() - 1);
    }
}
private static boolean folosZarojel(String s) {
    if (!(s.startsWith("(") && s.endsWith(")"))) {
        return false;
    }
    int szint = 1;
    for (int i = 1; i < s.length() - 1; ++i) {
        if (s.charAt(i) == '(') {szint++;}
        if (s.charAt(i) == ')') {szint--;}
        if (szint == 0) {return false;}
    }
    return true;
}
private int hammingTavolsag(String s1, String s2) {
```

```
        int t = 0;
        for (int i = 0; i < s1.length(); ++i) {
            if (i < s2.length() && s1.charAt(i) != s2.charAt(i)) {
                t++;
            }
        }
        return t;
    }
    /**
     * A formula Quine tábláját adja vissza.
     *
     * @return a formula 2^n + 1 soros Quine-táblát adja vissza.
     */
    public String quine() {
        int n = LogElem.betuk.size();
        boolean e; String s = "", k;
        s = LogElem.betuk.toString() + "␣|␣" + f + "\n";
        // Tautológiára és ellentmondásra külön jelölés lesz.
        tau = ell = true;
        for (int i = 0; i < pow(2, n); ++i) {
            // Meghatározzuk a változók értékeit i alapján.
            k = kettes(i, n);
            LogElem.setErtek(k);
            // Lekérdezzük a formula logikai értékét.
            e = fa.erteke();
            if (e) {ell = false;} else {tau = false;}
            // Ha hamis, akkor a termekhez adjuk.
            if (!e) {
                termek.add(k);
            }
            s = s + k + "␣|␣" + (e ? "1" : "0") + "\n";
        }
        return s;
    }
    /**
     * Kettes számrendszerbeli String-et ad vissza
     *
     * @param szam a konvertálandó szám
     * @param hossz az eredmény kívánt hossza
     * @return a szam kettes számrendszerbeli alakja String-ként
     */
    private static String kettes(int szam, int hossz) {
        int jegy = szam % 2;
        if (hossz == 1) {
            return "" + jegy;
        } else {
            return kettes((szam - jegy) / 2, hossz - 1) + jegy;
        }
    }
    /**
```

```
     * Ellenőriz, majd megpróbálja felépíteni a fát.
     *
     */
    public void epit() throws SyntaxException {
        if (helyese()) {
            fa = this.fatEpit(f);
        }
    }
    /**
     * Megadja, hogy a formula kezdeti
     * vizsgálata során mindent rendben talált-e.
     *
     * @return Hamis, ha a formula ellenőrzése során
     * hibát talált, igaz egyébként.
     */
    private boolean helyese() throws SyntaxException {
        // Leellenőrizzük a zárójeleket.
        //(...)
        // Létrehozzuk a változók (betűk) tömbjét.
        LogElem.betuTorles();
        for (int i = 0; i < f.length(); ++i) {
            if (!LogElem.jelek.containsKey(f.charAt(i))) {
                LogElem.ujbetu(f.charAt(i));
            }
        }
        // Kitöröljük a szóközöket.
        f = f.replaceAll("␣", "");
        // Ha ezzel üres kifejezést kaptunk, az is hiba.
        if (f.length() == 0) {
            throw new SyntaxException("Üres␣kifejezés.");
        }
        return true;
    }
    /**
     * Fát épít a megadott formulából.
     *
     * @param s a formula String-je.
     * @return a felépített fa (gyökéreleme).
     */
    private LogElem fatEpit(String s) throws SyntaxException {
        int szint = 0, h;
        // Zárójelek eltüntetése
        while (folosZarojel(s)) {
            s = s.substring(1, s.length() - 1);
        }
        // Ha csak egy változó maradt
        if (s.length()==1 && LogElem.betuk.contains(s.charAt(0))) {
            return new LogElem(LogElem.betuk.indexOf(s.charAt(0)));
        }
        if (s.equals("1")) {
```

```
        return new LogElem(LogElem.TAU);
    }
    if (s.equals("0")) {
        return new LogElem(LogElem.ELL);
    }
    for (h = 0; h < s.length(); ++h) {
        if (s.charAt(h) == '(') {szint++;}
        if (s.charAt(h) == ')') {szint--;}
        if (szint == 0) {
            if (s.charAt(h) == LogElem.CEKV) {
                if (h == s.length() - 1) {
                    throw new SyntaxException
                    ("Az ekvivalenciának nincs utótagja.");
                } else {
                    if (h == 0) {
                        throw new SyntaxException
                        ("Az ekvivalenciának nincs előtagja.");
                    } else {
                        return new LogElem(LogElem.EKV,
                        fatEpit(s.substring(0, h)),
                        fatEpit(s.substring(h + 1, s.length())));
                    }
                }
            }
        }
    }
    for (h = 0; h < s.length(); ++h) {
        if (s.charAt(h) == '(') {szint++;}
        if (s.charAt(h) == ')') {szint--;}
        if (szint == 0) {
            if (s.charAt(h) == LogElem.CIMPL) {
                if (h == s.length() - 1) {
                    throw new SyntaxException
                    ("Az implikációnak nincs utótagja.");
                } else {
                    if (h == 0) {
                        throw new SyntaxException
                        ("Az implikációnak nincs előtagja.");
                    } else {
                        return new LogElem(LogElem.IMPL,
                        fatEpit(s.substring(0, h)),
                        fatEpit(s.substring(h + 1, s.length())));
                    }
                }
            }
        }
    }
    for (h = 0; h < s.length(); ++h) {
        if (s.charAt(h) == '(') {szint++;}
        if (s.charAt(h) == ')') {szint--;}
```

```
        if (szint == 0) {
            if (s.charAt(h) == LogElem.CES) {
                if (h == s.length() - 1) {
                    throw new SyntaxException
                    ("A konjunkciónak nincs utótagja.");
                } else {
                    if (h == 0) {
                        throw new SyntaxException
                        ("A konjunkciónak nincs előtagja.");
                    } else {
                        return new LogElem(LogElem.ES,
                        fatEpit(s.substring(0, h)),
                        fatEpit(s.substring(h + 1, s.length())));
                    }
                }
            }
            if (s.charAt(h) == LogElem.CVAGY) {
                if (h == s.length() - 1) {
                    throw new SyntaxException
                    ("A diszjunkciónak nincs utótagja.");
                } else {
                    if (h == 0) {
                        throw new SyntaxException
                        ("A diszjunkciónak nincs előtagja.");
                    } else {
                        return new LogElem(LogElem.VAGY,
                        fatEpit(s.substring(0, h)),
                        fatEpit(s.substring(h + 1, s.length())));
                    }
                }
            }
        }
    }
    for (h = 0; h < s.length(); ++h) {
        if (s.charAt(h) == '(') {szint++;}
        if (s.charAt(h) == ')') {szint--;}
        if (szint == 0) {
            if (s.charAt(h) == LogElem.CNEG) {
                if (h == s.length()) {
                    throw new SyntaxException
                    ("A negáció után nem szerepel kifejezés");
                } else {
                    if (h != 0) {
                        throw new SyntaxException
                        ("A negációt nem előzheti meg változó.");
                    } else {
                        return new LogElem(LogElem.NEG,
                        fatEpit(s.substring(h + 1, s.length())));
                    }
                }
```

```
                }
            }
        }
        // Ha nem egy betű, és nincs benne
        // logikai összekötőjel, akkor hibás
        throw new SyntaxException("Hibás szintaxis");
    }
}
```

A.2. rezolucio

A **rezolucio** programhoz tartozó Java osztályok.

```
public class Rezolucio {
    private List<String> program = new ArrayList<String>();
    public static Set<Character> betuk;
    public Rezolucio() {super();
    betuk = new HashSet<Character>();}
    public boolean addSzabaly(String szabaly) {
        try {
            szabaly = ellenoriz(szabaly);
        } catch (Exception ex) {
            return false;
        }
        betuFelvetel(szabaly);
        program.add(szabaly);
        return true;
    }
    private static String ellenoriz(String s) throws Exception {
        //   A:-B,C.  vagy  A.  alakú legyen
        String r = ""+s;
        r.replaceAll(" ", "");
        if (r.length()<2) throw new Exception();
        if (!Character.isUpperCase(r.charAt(0)))
                throw new Exception();
        if (r.charAt(1)!='.') {
            if (r.length()<5 || r.length()%2==0)
                throw new Exception();
            if (!r.substring(1, 3).equals(":-") ||
                        !r.endsWith(".")) {
                throw new Exception();
            }
            for (int i=3;i<r.length()-2;i+=2) {
                if (!Character.isUpperCase(r.charAt(i)) ||
                    (i<r.length()-2 && r.charAt(i+1)!=',')) {
                        throw new Exception();
                }
            }
        }
```

```
        // Csak a betűket hagyjuk meg
        // az egyszerűbb feldolgozhatóságért
        r = r.replaceAll(":|-|,|\\.", "");
        return r;
    }
    public String bizonyit(String s) {
        String fokloz = "";
        for (char i = 0; i<26; ++i)
            if (s.contains(Character.toString((char)(i+65))))
                fokloz += "H"; else fokloz += "-";
        rez.Rez r = new Rez(fokloz);
        r.programTorol();
        for (int i = 0; i<program.size(); ++i) {
            r.addProgramSor(program.get(i));
        }
        BacktrackKereso k = new BacktrackKereso(r,
                BacktrackKereso.CELALLAPOTHOZ_VEZETO_UT +
                BacktrackKereso.KORFIGYELES);
        k.Keres();
        for (kereso.Csucs bcs: k.getTerminalisok())
                k.kiir(bcs);
        return (k.getTerminalisok().isEmpty()?"HAMIS.":"IGAZ.");
    }
    private static void betuFelvetel(String s) {
        for (int i=0; i<s.length(); ++i) {
                betuk.add(s.charAt(i));
        }
    }
}
```

A backtrack kereséshez használt állapot osztálya.

```
public class Rez extends Allapot {
    private String fokloz;
    private List<String> program;
    public String getFokloz() {return fokloz;}
    public List<String> getProgram() {return program;}
    public void programTorol() {
        program = new ArrayList<String>();
        opok = new HashSet<Operator>();
    }
    public void addProgramSor(String sor) {
        program.add(sor);
        if (opok == null) opok = new HashSet<Operator>();
        opok.add(new Rezolval(program.size() - 1));
        // ABC-sorrendbe rendezzük a program sorait
        Collections.sort(program);
    }
    public Rez() {program = new ArrayList<String>();}
    public Rez(String fokloz) {this(); this.fokloz = fokloz;}
    public Rez(Rez allapot) {this(allapot.getFokloz());
```

```
        program.addAll(allapot.getProgram());}
    private static String betucsere(String s,
                    int index, char ujbetu) {
        return new String(s.substring(0,index) +
                ((Character)ujbetu) + s.substring(index+1));
    }
    @Override
    public Allapot alkalmaz(Operator op) {
        Rez uj = new Rez(this);
        if (op instanceof Rezolval) {
            Rezolval m = (Rezolval) op;
            String s = program.get(m.getMivel());
            uj.fokloz = betucsere(uj.fokloz, s.charAt(0)-65, '-');
            for (int i=1;i<s.length();++i) {
                uj.fokloz = betucsere(uj.fokloz,s.charAt(i)-65,'H');
            }
        return uj;
        }
        return null;
    }
    @Override
    public Boolean celallapot_e() {
        for (int i = 0; i < 26; ++i) {
        if (fokloz.charAt(i)!='-') return false;
        }
        return true;
    }
    @Override
    public Boolean elofeltetel(Operator op) {
        if (op instanceof Rezolval) {
         int mivel = ((Rezolval)op).getMivel();
         String s = program.get(mivel);
         if (fokloz.charAt(s.charAt(0)-65)!='H') return false;
         for (int i = 1; i < s.length(); ++i) {
          if (fokloz.charAt(s.charAt(i)-65)=='I') return false;
         }
         return true;
        }
        return false;
    }
    //(...)
    @Override
    public boolean equals(Object obj) {
    return (obj!=null &&
    (obj instanceof Rez) &&
    program.equals(((Rez)obj).getProgram()) &&
    fokloz.equals(((Rez)obj).getFokloz()));
    }
}
```