Multi-hipertabló az automatikus tételbizonyításban

Multi-Hyper Tableaux in Automated Theorem Proving

Doktori (Ph.D.) értekezés tézisei

Kovásznai Gergely

Debreceni Egyetem
Informatikai Kar
Debrecen, 2007.

TARTALOMJEGYZÉK

1. BEVEZETÉS

A disszertációban a klasszikus elsőrendű logika tételbizonyítással kapcsolatos kérdéseivel foglalkozunk. Az automatikus tételbizonyító algoritmusok alapját képezik a tételbizonyító szoftvereknek, melyek fontos alkotóelemei a széles körben használt szakértői rendszereknek és az azokon alapuló komplex alkalmazásoknak [24, 26, 27, 28]. Hasonlóképp a hátterét jelentik a logikai programozási nyelveknek. A disszertációban a *hipertabló* [3] (és közvetve a hiperrezolúció [21]) teljes automatizálhatóságának kérdésével foglalkozunk. Ennek megfelelően ismertetjük saját kalkulusunkat, a *multi-hipertabló* kalkulust [25] a célul kitűzött problémák megoldására. A disszertáció négy fejezetből, egy angol nyelvű összefoglalásból és függelékből áll.

Az első fejezetben röviden áttekintjük az elsőrendű logika azon fogalmait, melyekre szükségünk van a továbbiakban.

A második fejezetben azon kalkulusokat vesszük sorra, melyek valamilyen értelemben az általunk kidolgozott kalkulus előzményeinek tekinthetők. Az analitikus tabló [23, 11, 13] és a rezolúció [20, 2, 21] különböző változatait írjuk le, majd eljutunk a *hipertablóhoz* [3], melyre leginkább támaszkodunk az értekezésünkben. A hipertabló komoly automatizálási gondokkal küzd. Az ilyen problémák lehetséges orvoslására a *rigid hipertabló* [14] kalkulust hozzuk fel példaként. Erről a kalkulusról ugyanakkor bebizonyítjuk, hogy nem helyes, illetve rámutatunk, hogy a kalkulus teljessége nincs bizonyítva.

A harmadik fejezet tartalmazza elért eredményeinket. A fejezet első részében részletes leírását adjuk a *multi-hipertabló* kalkulusnak [25], bebizonyítjuk a helyességét, majd egy külön részben a teljességét is. A következő szakasz a klózgenerálás témakörével foglalkozik; itt ismertetjük az általunk kidolgozott eljárást is, a *klózgeneráló tablót* [25]. Ezen módszer – az irodalomban ismert más klózgeneráló módszerekhez hasonlóan – exponenciális bonyolultságú, ezért DAG-ok felhasználásával linearizáljuk. A fejezet negyedik részében a *redundancia* témakörével foglalkozunk. Három – logikailag hasonló – célra használunk fel redundanciával kapcsolatos vizsgálatokat. Először a *hipertabló* Baumgartner által megadott *redundancia kritériumát* [3] vezetjük le. Majd

klózok egyszerűsítését végezzük el redundanciavizsgálat segítségével. Utoljára azt vizsgáljuk meg, hogy tetszőleges *rigid literálos tablóhoz* (és így a multi-hipertablóhoz is) hogyan adható meg redundanciafogalom. A fejezet utolsó részében a *heurisztikus tételbizonyítás* lehetőségeivel foglalkozunk a multi-hipertabló kapcsán, majd ehhez kapcsolódva megvizsgáljuk, hogy a *klózok zártságára* vonatkozó kikötésnek az eltörlése vajon milyen következményekkel jár a multi-hipertablóra nézve.

A disszertáció utolsó fejezete az elért eredmények összefoglalása mellett röviden értékeli is azokat, valamint rávilágít azon elemekre és kérdésekre, melyek továbbfejlesztése és megválaszolása a jövőbeli kutatásoknak irányt mutat.

A függelékben algoritmikus megvalósítását írjuk le néhány, a disszertációban részletezett eljárásnak. Előbb egy saját unifikációs algoritmust ismertetünk, majd klózok redundancián alapuló egyszerűsítését valósítjuk meg. Végül magának a multi-hipertabló kalkulusnak adjuk algoritmikus megvalósítását.

2. ALAPFOGALMAK

A disszertációban kizárólag az *elsőrendű klasszikus logika* keretei között dolgozunk. A kapcsolódó fogalmakat az irodalomban megszokott módon értelmezzük és használjuk, ha attól el kívánunk térni, akkor azt mindig jelezni fogjuk.

Egy F formula vagy formulahalmaz esetén $\mathcal{FV}(F)$-fel jelöljük az F *szabad változóinak halmazát.* Az F-et *zártnak* nevezzük, ha $\mathcal{FV}(F) = \emptyset$; egyébként F *nyitott.* Az F formula (univerzális) *lezártját* $\forall F$-fel jelöljük.

2.1. Definíció. (Változóidegenség)

Az A és B formulákat akkor nevezzük *változóidegennek*, ha $\mathcal{FV}(A) \cap \mathcal{FV}(B) = \emptyset$. Egy formulahalmaz változóidegen, ha annak formulái páronként változóidegenek. □

A disszertációban az irodalomban megszokott módon definiáljuk a *(term)-helyettesítések* fogalmát. Az üres helyettesítést ϵ-nal jelöljük. Ugyancsak az irodalomban megszokott módon értelmezzük egy F formulán vagy formulahalmazon megengedett helyettesítéseket, illetve ezek elvégzését F-en.

2.2. Definíció. (Példány)

Egy A formula akkor *példánya* egy B formulának, ha létezik olyan σ helyettesítés, hogy $A = B\sigma$. □

A következő két definíció a helyettesítések egy speciális osztályára, az ún. átnevezésekre vonatkozik.

2.3. Definíció. (Átnevezés)

Az *átnevezés* egy olyan σ helyettesítést, ahol

(1) minden $x \in \mathcal{D}om(\sigma)$ esetén $\sigma(x)$ változó, és

(2) bármely $x, y \in \mathcal{D}om(\sigma)$ esetén ha $x \neq y$, akkor $\sigma(x) \neq \sigma(y)$. □

2.4. Definíció. (Átnevezett)

Az A formula akkor *átnevezettje* a B formulának, ha létezik olyan σ átnevezés, hogy $A = B\sigma$. □

A disszertációban túlnyomórészt *klózokon működő kalkulusokkal* dolgozunk, ezért különös fontossággal bír a literálok és a klózok pontos definiálása.

2.5. Definíció. (Literál)

Literálnak nevezünk egy A vagy $\neg A$ formulát, ahol A atomi formula. Megkülönböztetünk *pozitív és negatív literálokat* a következő definíció szerint:

(1) A akkor és csak akkor pozitív, ha $A \neq \bot$;

(2) $\neg A$ akkor és csak akkor pozitív, ha A negatív.

Egy L literál esetén az L *alapját* $\underline{L}$-lel jelöljük, és a következőképpen definiáljuk:

$$\underline{A} = \underline{\neg A} \quad = \quad \begin{cases} \top & \text{, ha } A = \bot \\ A & \text{, egyébként} \end{cases}$$ □

2.6. Definíció. (Klóz)

Klóznak nevezünk egy $C = \forall x_1 \ldots \forall x_k (L_1 \vee L_2 \vee \ldots \vee L_n)$ *zárt* formulát, ahol minden L_i literál, $k \geq 0$, $n \geq 0$. A C *magja* alatt az $L_1 \vee L_2 \vee \ldots \vee L_n$ formulát értjük, melyet $\langle C \rangle$-vel jelölük. □

Felhívjuk a figyelmet arra, hogy az előző definícióban kikötöttük, hogy minden klóz zárt. Ezért nem fog félreértéshez vezetni, ha kvantoros előtagjait nem írjuk le; azaz a fenti C klóz helyett magját, $L_1 \vee L_2 \vee \ldots \vee L_n$-t is használhatjuk. Ugyanezen klózt szokás csupán *literáljai multihalmazaként*, azaz $\{L_1, L_2, \ldots, L_n\}$-ként is jelölni és használni.

Az *üres klózt* (azaz mikor $n = 0$) $\bot$-val jelöljük.

Egy C klóz akkor *pozitív* (*negatív*), ha minden $L \in C$ literál pozitív (negatív).

A klózokon működő kalkulusok gyakran használt művelete a klózok új példányainak előállítása, erre vonatkozik a következő definíció:

2.7. Definíció. (Klóz új példánya)

Egy C *klóz új példánya* – adott $\mathcal{V}$ változóhalmaz mellett – C-nek egy olyan $\langle C \rangle \sigma$ példánya, ahol

(1) σ átnevezés,

(2) $\mathcal{D}om(\sigma) = \mathcal{FV}\big(\langle C \rangle\big)$, és

(3) $\mathcal{R}ange(\sigma) \cap \mathcal{V} = \emptyset$. □

Annak jelölésére, hogy egy F formula egy $\mathcal{M}$ *modellben igaz*, az $\mathcal{M} \models F$ jelölést fogjuk alkalmazni. *Nyitott* F esetén $\mathcal{M} \models F$ akkor és csak akkor, ha $\mathcal{M} \models \forall F$.

Két A és B formulát *ekvivalensnek* nevezünk, ha bármely $\mathcal{M}$ modell esetén $\mathcal{M} \models A$ akkor és csak akkor, ha $\mathcal{M} \models B$; ennek jelölése: $A \sim B$.

A $P_1, \ldots, P_n$ $(n \geq 0)$ formuláknak akkor és csak akkor *logikai következménye* a K formula, ha minden olyan $\mathcal{M}$ modellben, ahol minden P_i esetén $\mathcal{M} \models P_i$, az $\mathcal{M} \models K$ is teljesül. Jelölése: $P_1, \ldots, P_n \models K$.

3. TÉTELBIZONYíTÓ MÓDSZEREK

A disszertáció 2. fejezetében azon kalkulusokat vesszük sorra, melyekre munkánk során támaszkodtunk.

3.1. Tablókalkulusok

A disszertáció 2.1. fejezetében Smullyan *analitikus tablójával* [23] foglalkozunk. Tabló alatt – az irodalomban megszokott módon – formulákkal címkézett fát értünk. A tabló ágait sok esetben – ahol ez tömörebb megfogalmazást tesz lehetővé – mint csúcsai *formuláinak multihalmazát* ragadjuk meg. Hasonlóképpen, a tablót sokszor mint *ágainak multihalmazát* értelmezzük.

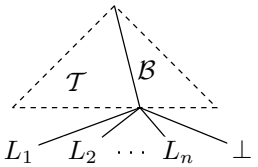
A 2.1.1. fejezetben Fitting *szabadváltozós tablóját* [11] írjuk le. Az utóbbi kalkulusban hivatkozunk először a Robinson által megadott unifikációs algoritmusra [20]. A legáltalánosabb unifikátor (MGU) előállítására az A.1. függelékben leírt, saját rekurzív unifikációs algoritmust használjuk. Ez az $\mathcal{U}()$ függvényt valósítja meg, mely az $(E_1, F_1), \ldots, (E_n, F_n)$ $n \geq 0$ db. kifejezéspárhoz rendeli azok MGU-ját, amennyiben az létezik.

A 2.1.2. fejezetben a *klóztabló* kalkulussal [13] foglalkozunk. Ez egy olyan speciális tablókalkulus, mely egy-egy levezetési lépésben egy-egy klózt csatol a tabló ágaihoz. Erre a műveletre bevezetjük a következő jelölést:

3.1. Definíció. (Klóz csatolása a tabló egy ágához)

Egy $C = \{L_1, L_2, \ldots, L_n\}$ *klóznak* a $\mathcal{T}$ tabló egy $\mathcal{B} \in \mathcal{T}$ ágához való *csatolása* alatt értjük a

$\mathcal{T} +^{\mathcal{B}} C \quad =$



tablót. □

3.2. Rezolúciós kalkulusok

A disszertáció 2.2. fejezetében Robinson (bináris) *rezolúciós kalkulusával* foglalkozunk [20, 2], a 2.2.1. fejezetben pedig a *lineáris inputrezolúciós* stratégiával, mely *nem teljes* az elsőrendű logikában (csak Horn-klózok esetén).

A 2.2.2. fejezetben leírjuk Robinson *hiperrezolúciós kalkulusát* [21], mely viszont *teljes* az elsőrendű logikában. Elvben a hiperrezolúció effektív, ám nehezen automatizálható.

3.3. Hipertabló kalkulusok

A hipertabló kalkulusok a hiperrezolúció automatizálhatóságával kapcsolatos kérdésekre keresik a választ, arra a felismerésre támaszkodva, hogy a *tabló* eljárás fa adatszerkezete jól felhasználható a hiperrezolvens leírásában. Így lehetővé válik, hogy a rezolúciós kalkulusokban használt sor adatszerkezethez képest sokkal strukturáltabb formában tároljuk a levezetés „history"-ját. Vagyis a hipertabló felfogható *a hiperrezolúció és a klóztabló* egyfajta egyesítésének.

A hipertabló kalkulusok napjaink tételbizonyítással kapcsolatos kutatásainak friss eredményei [6, 3, 4]. Bár a kalkulus lehetősége már a 70-es évek végén felvetődött [5], csak 1996-ban jelent meg kellő kidolgozottsággal a szakirodalomban (Baumgartner [3]). A disszertáció 2.3. fejezetében Baumgartner *hipertablóját* írjuk le, mellyel kapcsolatosan a következő összegzést tesszük:

- A hipertabló helyes és teljes [3].
- A faktorizáció nem szükséges a kalkulus teljességéhez (szemben a hiperrezolúcióval).
- A hipertablónak az ún. *tisztító helyettesítések* használatában rejlik a gyengéje, mivel azok előállítása nincs automatizálva.

A tisztító helyettesítések hipertablóból való eliminálásával több szerző is próbálkozott [4, 14, 10]. A disszertáció 2.3.1. fejezetében Kühn *rigid hipertablóját* [14] írjuk le példaként. Kühn felismerte, hogy a hipertabló tisztító helyettesítések nélkül csak Horn-klózokon teljes. Kühn a tisztító helyettesítések használatának kiküszöbölése érdekében a tablóban rigid változókat használ, illetve tablóból kinyert klózok (ún. ágklózok) igény szerinti ismétlésének lehetőségét építi be a kalkulusába. A rigid hipertablóval kapcsolatosan a következő összegzést tesszük:

- Az *ágklózok példányosítása nem szükséges* eleme a kalkulusnak.
- A rigid hipertabló *nem helyes*. Ezt a disszertáció 2.3.15. tételében bizonyítjuk.
- Nem bizonyított, hogy a rigid hipertabló *teljes-e*.

4. MULTI-HIPERTABLÓ

Saját fő eredményeinket a disszertációban a 3. fejezet tartalmazza. A disszertáció 3.1. fejezetében leírjuk a multi-hipertabló kalkulust, mely a [25] cikkben került publikálásra. A multi-hipertabló Baumgartner hipertablójának és Kühn rigid hipertablójának a továbbfejlesztése, mely

(1) kiküszöböli a hipertabló által használt tisztító helyettesítéseket,

(2) eliminálja a rigid hipertablóban használt ágklóz másolatok problémáját,

(3) a hipertabló és a rigid hipertabló effektivitási problémáira is megpróbál választ találni,

(4) bizonyítottan helyes és teljes kalkulus.

4.1. A multi-hipertabló kalkulus

A továbbiakban megadjuk a multi-hipertabló kalkulus pontos leírását. Először két definíciót adunk meg, melyek leírják, mit értünk egy $\mathcal{T}$ tabló valamely $\mathcal{B}$ ágának kiterjesztése alatt.

4.1. Definíció. (Kiterjesztés)

Kiterjesztésnek egy $[E, \sigma]$ párt nevezünk, ahol E egy klóz és σ egy helyettesítés. □

4.2. Definíció. (Kiterjesztés alkalmazása)

Az $[E, \sigma]$ *kiterjesztést* oly módon *alkalmazzuk* a $\mathcal{T}$ tabló egy $\mathcal{B} \in \mathcal{T}$ ágán, hogy előállítjuk a következő tablót:

$$(\mathcal{T} +^{\mathcal{B}} E)\sigma \quad . \qquad \square$$

Egy kiterjesztést egyszerre három dologhoz rendelhetünk hozzá: egy klózhoz, egy klóz-multihalmazhoz és egy literál-multihalmazhoz. Ezt a hozzárendelést írja le a következő definíció:

4.3. Definíció. (Kiterjesztés meghatározása)

(1) Legyen $C^{\ominus}$ egy klóz és legyen $\{L_1^{\ominus}, \ldots, L_k^{\ominus}\}$ a $C^{\ominus}$ összes negatív literáljának a halmaza, ahol $k \geq 1$.

(2) Legyen $\{C_1^{\oplus}, \ldots, C_k^{\oplus}\}$ pozitív klózoknak egy multihalmaza.

(3) Legyen $\{L_1^{\oplus}, \ldots, L_k^{\oplus} \mid L_i^{\oplus} \in C_i^{\oplus}\}$ (pozitív) literáloknak olyan multihalmaza, hogy $\sigma = \mathcal{U}\big((\underline{L_1^{\ominus}}, \underline{L_1^{\oplus}}), \ldots, (\underline{L_k^{\ominus}}, \underline{L_k^{\oplus}})\big)$ létezik.

Jelöljük E-vel a következő klózt:

$$C^{\ominus} \backslash \{L_1^{\ominus}, \ldots, L_k^{\ominus}\} \ \cup \ \bigcup_{i=1}^{k} C_i^{\oplus} \backslash \{L_i^{\oplus}\} \quad . \tag{4}$$

Ekkor a $C^{\ominus}$-hoz, $\{C_1^{\oplus}, \ldots, C_k^{\oplus}\}$-hoz és $\{L_1^{\oplus}, \ldots, L_k^{\oplus}\}$-hoz tartozó,

$$e\Big(C^{\ominus}, \{C_1^{\oplus}, \ldots, C_k^{\oplus}\}, \{L_1^{\oplus}, \ldots, L_k^{\oplus}\}\Big) \quad \text{-val}$$

hivatkozott kiterjesztés legyen $[E, \sigma]$. □

A következő definíció leírja, hogy egy ág és egy input klózhalmaz esetén mely klózokhoz, klóz-multihalmazokhoz és literál-multihalmazokhoz állíthatunk elő kiterjesztéseket.

4.4. Definíció. (Ág és klózhalmaz kiterjesztései)

Egy $\mathcal{B}$ *ághoz* és egy $\mathcal{C}$ *klózhalmazhoz* tartozó *kiterjesztések* a következő halmaz elemei:

$$\mathcal{E}(\mathcal{B},\mathcal{C}) = \left\{ e\Big(\widehat{C}^{\ominus}, \{\widehat{C}_1^{\oplus}, \ldots, \widehat{C}_k^{\oplus}\}, \{L_1^{\oplus}, \ldots, L_k^{\oplus}\}\Big) \;\middle|\; \begin{array}{lll} C^{\ominus} \in \mathcal{C}, & & (1) \\ C_i^{\oplus} \in \mathcal{B} \cup \mathcal{C}, & & (2) \\ L_i^{\oplus} \in \widehat{C}_i^{\oplus} & & (3) \end{array} \right\} \quad .$$

A fentiekben bármely C klóz esetén

$$\widehat{C} \quad = \quad \begin{cases} C \text{ új példánya}^1 & \text{, ha } C \in \mathcal{C}; \\ C & \text{, egyébként.} \end{cases}$$

□

[1] Ezen példányok a $\mathcal{V} = \mathcal{FV}(\mathcal{T})$ változóhalmaz mellett újak (lásd: 2.7. definíció), ahol $\mathcal{T}$ az aktuális tabló.

Az előzőekben leírt definíciókat használjuk fel most a multi-hipertabló kalkulus definiálásához:

4.5. Definíció. (Multi-hipertabló)

Adott egy $\mathcal{C}$ klózhalmaz. $\mathcal{C}$-hez a következő induktív definícióval adjuk meg a *pozitív multi-hipertabló* fogalmát:

(1) Inicializációs szabály:
$\{\{\top\}\}$ a $\mathcal{C}$-nek pozitív multi-hipertablója.

(2) Kiterjesztési szabály:

(a) Legyen $\mathcal{T}$ a $\mathcal{C}$-nek pozitív multi-hipertablója, és legyen $\mathcal{B} \in \mathcal{T}$.

(b) Legyen $e \in \mathcal{E}(\mathcal{B}, \mathcal{C})$ kiterjesztés.

Ekkor az e-nek a $\mathcal{T}$ tabló $\mathcal{B}$ ágán való alkalmazásával kapott tabló is $\mathcal{C}$-nek pozitív multi-hipertablója. □

4.6. Megjegyzés.

A *negatív multi-hipertabló* definíciója megkapható a 4.3., a 4.4. és a 4.5. definíciókból a „pozitív" és „negatív" jelzők felcserélésével. □

A multi-hipertabló helyessége a hiperrezolúció helyességéből meglehetősen egyszerűen bizonyítható.

4.7. Tétel. (Multi-hipertabló – Helyesség)

A multi-hipertabló kalkulus helyes.

A kalkulus teljességének bizonyítása viszont messze nem olyan egyszerű. Kühn rigid hipertablója egy-egy levezetési lépésben egyszerre akár több klózt is csatolhat a tablóhoz, úgy irányítva ezen műveletet, hogy az egymással rezolválható klózok kerüljenek a tablóban egymás alá a megfelelő ágakra, ily módon lezárva egy-egy ágat. Ezzel szemben a multi-hipertabló hiperrezolvenst számol (egy $[E, \sigma]$ kiterjesztés megfelel az $\langle E \rangle \sigma$ hiperrezolvensnek), és azt csatolja a tablóhoz. Ez a megközelítési mód áttekinthetőbb kalkulust eredményez, hiszen a tabló bővítése egyidejűleg *mindig csak egy klózzal* történik. De a legnagyobb előny mégis a teljesség bizonyításában nyilvánul meg: a multi-hipertabló teljességének bizonyításához fel tudjuk használni a hiperrezolúció bizonyított teljességét, azaz azt a tényt, hogy egy kielégíthetetlen klózhalmaznak létezik hiperrezolúciós cáfolata.

A teljesség bizonyítása két lépcsőben történik:

(1) A disszertáció 3.2.1. fejezetében egy megadott hiperrezolúciós cáfolathoz egy ún. *hiperrezolúciós gráfot* generálunk. A hiperrezolúciós gráf fogalma új a szakirodalomban, definiálásához számos saját fogalmat használunk fel, mint például a *csoportosított gráf*, a *levezető csúcscsoport* és az $\mathcal{N}$*-él*.

(2) A disszertáció 3.2.2. fejezetében a hiperrezolúciós gráfból *zárt multi-hipertablót* generálunk.

4.8. Tétel. (Multi-hipertabló – Teljesség)

A multi-hipertabló kalkulus teljes.

4.2. Klózgenerálás tablóval

A legtöbb klózgeneráló algoritmus közvetlenül a klózgenerálásra vonatkozó ekvivalens átalakításokat alkalmazza a formulán [16]. Elenyésző számban, de léteznek más, alternatív módszerek is. Ilyenek például a BDD-ken (Binary Decision Diagram) vagy más néven Shannon-gráfokon alapuló módszerek [16, 19]. A BDD-k alapvetően propozícionális logikában alkalmazhatóak. Sokszor párhuzamot vonnak a BDD-k és a tablók között [19]. Hasonlóan a tablókkal párhuzamosíthatóak a pp-kifejezéseken, a formulák duál formáján és egymásba ágyazott listákon alapuló algoritmusok [11, 17], ám ezek is csak nulladrendű logikában alkalmazhatóak.

A tablók és a normálformák kapcsolata régóta ismert a szakirodalomban, természetesen itt is csak propozícionális esetről beszélünk. Egy nulladrendű formulához generált befejezett *analitikus duál tablóból* a következőképpen nyerhető a formula klóz normálformája: minden ág esetén az adott ágon szereplő literáloknak vegyük a diszjunkcióját, és az így kapott formuláknak a konjunkcióját.

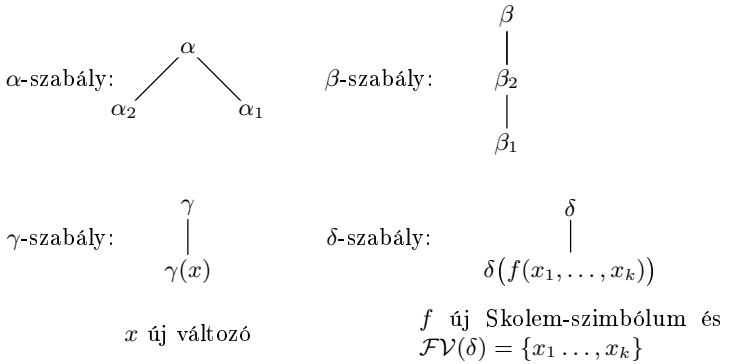
A disszertáció 3.3. fejezetében rávilágítunk, hogy egy elsőrendű logikában alkalmazott klózgeneráló tablómódszer szabályainak ki kell váltaniuk a formula prenexizálását és skolemizálását. Erre a célra a *szabadváltozós tabló* [11] γ- és δ-szabályait használjuk.

4.9. Definíció. (Klózgeneráló tabló)

Adott egy $\mathcal{F} = \{F_1, \ldots, F_n\}$ formulahalmaz. $\mathcal{F}$-hez a következő induktív definícióval adjuk meg a *klózgeneráló tabló* [25] fogalmát:

(1) $\big\{\{F_1\}, \ldots, \{F_n\}\big\}$ az $\mathcal{F}$ klózgeneráló tablója.

(2) Ha $\mathcal{T}$ az $\mathcal{F}$ klózgeneráló tablója, akkor $\mathcal{T}'$ is az, amennyiben $\mathcal{T}'$-t a 4.1. ábrán látható szabályok valamelyikének alkalmazásával nyertük $\mathcal{T}$-ből.□



4.1. ábra. Klózgeneráló tabló – levezetési szabályok

Fontos momentum, hogy minden formulát, amire már alkalmaztunk szabályt, törlünk az ágról mint címkét. Egy klózgeneráló tablót *befejezettnek* tekintünk, ha már csak literálokat tartalmaz. Azt kell bebizonyítanunk, hogy egy $\mathcal{F}$ formulahalmazhoz generált befejezett klózgeneráló tabló olyan klózhalmazt állít elő, mely pontosan akkor kielégíthető, ha $\mathcal{F}$ is az. Ehhez előbb definiálunk egy olyan függvényt, mely a tablóhoz egy formulát rendel, majd pedig bebizonyítunk egy lemmát.

4.10. Definíció.

Az ágakon és tablókon értelmezett $cnf()$ függvényt a következőképpen definiáljuk:

(1) Egy $\mathcal{B} = \{A_1, \ldots, A_k\}$ ág esetén $cnf(\mathcal{B}) = (A_1 \vee \ldots \vee A_k)$.

(2) Egy $\mathcal{T} = \{\mathcal{B}_1, \ldots, \mathcal{B}_k\}$ tabló esetén $cnf(\mathcal{T}) = cnf(\mathcal{B}_1) \wedge \ldots \wedge cnf(\mathcal{B}_k)$.□

4.11. Lemma.

Egy $\mathcal{T}$ klózgeneráló tabló esetén $cnf(\mathcal{T})$ akkor és csak akkor kielégíthető, ha valamely szabály alkalmazásával belőle nyert $\mathcal{T}'$ tabló esetén is kielégíthető $cnf(\mathcal{T}')$.

A lemma alapján könnyen bizonyítható a következő tétel:

4.12. Tétel.

Egy $\mathcal{F}$ formulahalmaz akkor és csak akkor kielégíthetetlen, ha a

$$\mathcal{C} \quad = \quad \{cnf(\mathcal{B}) \mid \mathcal{B} \in \mathcal{T}\}$$

kielégíthetetlen, ahol $\mathcal{T}$ az $\mathcal{F}$ klózgeneráló tablója.

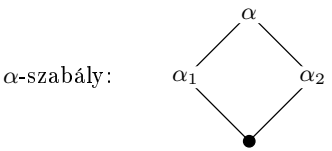
Vegyük észre, hogy a fenti tételben ha $\mathcal{T}$ *befejezett*, akkor $\mathcal{C}$ *klózhalmaz.* A $\mathcal{C}$ klózhalmaz tehát pontosan akkor kielégíthetetlen, ha $\mathcal{F}$ kielégíthetetlen.

4.2.1. Lineáris klózgenerálás

Mint a klózgeneráló algoritmusok szinte mindegyike, a klózgeneráló tabló is *exponenciális bonyolultságú.* A probléma az, hogy egyes formulák a tabló több ágán is előfordulhatnak. A problémát nem más, mint maga a tabló – mint *fa adatszerkezet* – okozza. Ezért a fának olyan módosítása lenne előnyös, amely tartalmazhat hurkokat – ez pedig az irodalomban a *DAG*-ként[2] ismert adatszerkezet.

A DAG-ok nem ismeretlenek a logikai alkalmazások számára. Robinson exponenciális unifikációs algoritmusát a 70-es években Paterson és Wegman [18] próbálta linearizálni, DAG-ok alkalmazásával. A DAG-okat a logika egyéb területeken is sikerrel alkalmazták, pl. modális tablómódszerekben [7].

A disszertáció 3.3.1. fejezetében DAG-ok bevezetésével linearizáljuk a klózgeneráló tablót, annak α-szabályát a 4.2. ábrán látható módon módosítva.



4.2. ábra. Klózgeneráló DAG – módosult α-szabály

Fontos momentum a szabályok alkalmazásának módja: az új csúcso(ka)t mindig közvetlenül a *feldolgozandó csúcs alá* szúrjuk be.

Vitathatatlan, hogy a vázolt algoritmus *lineáris bonyolultságú*, hiszen a befejezett klózgeneráló DAG megkonstruálásához szükséges szabályalkalmazások száma lineáris függvénye a kiindulási formulahalmazban szereplő logikai összekötőjelek számának.

[2] irányított körmentes gráf

4.2.2. Ágak zártságán alapuló klózhalmaz-egyszerűsítés

A disszertáció 3.3.2. fejezetében megmutatjuk, hogy egy befejezett klózgeneráló tablóban az *érvényes klózok* detektálása az *ágak lezárásával* milyen egyszerűen tehető meg. Egy $\mathcal{B}$ ág esetén:

> Ha van két olyan $L_1, L_2 \in \mathcal{B}$ különböző előjelű literál, hogy $\mathcal{U}(\underline{L_1}, \underline{L_2})$ létezik, akkor $\mathcal{B}$-t (mint klózt) nem vesszük fel a klózhalmazba.

4.3. Redundancia

A disszertáció 3.4. fejezetében definiáljuk a (formulákon értelmezett) redundancia fogalmát:

4.13. Definíció. (Redundancia)

Egy A *formula redundáns* egy B formulára és a $\circ$ szimmetrikus, bináris logikai összekötőjelre nézve, ha $A \circ B \sim B$. □

Ezt a redundanciafogalmat három, logikailag hasonló célra fogjuk felhasználni.

4.3.1. Redundancia hipertablóban

A disszertáció 3.4.1. fejezetében leírjuk és a fenti redundanciafogalomból levezetjük a Baumgartner-féle *hipertabló redundanciafogalmát* [3]. Ehhez bebizonyítjuk a következő tételt:

4.14. Tétel.

Adottak az A és a $B_1 \wedge \ldots \wedge B_k$ formulák. Ha valamely B_i esetén létezik olyan σ helyettesítés, hogy $A = B_i\sigma$, akkor A redundáns $B_1 \wedge \ldots \wedge B_k$-ra és a $\wedge$-ra nézve.

Ezen a tételen alapul a hipertabló redundanciafogalma:

4.15. Definíció. (Redundancia hipertabló ágára nézve)

Egy D *klózt redundánsnak* mondunk egy hipertabló $\mathcal{B}$ *ágára* nézve, ha valamely $L_1 \in D$ és $L_2 \in \mathcal{B}$ esetén van olyan σ helyettesítés, hogy $L_1 = \widehat{L}_2\sigma$ [3]. □

[3] A $\widehat{L}$ jelölés az L literál egy új példányát jelenti (jobban mondva: $\widehat{L}$ a $\forall L$ klóz új példányának egyetlen literálja).

4.3.2. Redundancián alapuló klózegyszerűsítés

A disszertáció 3.4.2. fejezetében *klózoknak redundancián alapuló egyszerűsítését* írjuk le [25]. Ehhez nyújt alapot a következő tétel és az arra épülő következő definíció:

4.16. Tétel.

Adottak az A és a $B_1 \vee \ldots \vee B_k$ formulák. Ha

$$\mathcal{FV}(A) \cap \mathcal{FV}(B_1 \vee \ldots \vee B_k) \quad = \quad \emptyset$$

és valamely B_i esetén létezik olyan σ helyettesítés, hogy $A\sigma = B_i$, akkor A redundáns $B_1 \vee \ldots \vee B_k$-ra és a $\vee$-ra nézve.

4.17. Definíció. (Redundancia klózra nézve)

Egy D *klóz redundáns* a C *klózra* nézve, ha $\mathcal{FV}\big(\langle C \rangle\big) \cap \mathcal{FV}\big(\langle D \rangle\big) = \emptyset$ és létezik olyan σ helyettesítés, hogy $\langle D \rangle \sigma \subseteq C$. □

Ezen alapul a redundáns klóz fogalma:

4.18. Definíció. (Redundáns klóz)

Redundáns klóznak egy olyan C klózt nevezünk, melyre létezik olyan $D \subset C$, hogy D redundáns a $C \backslash D$ klózra nézve. □

A tárgyalandó egyszerűsítés lényege az, hogy egy adott klóz *redundanciáját megszüntetjük*, azaz előállítjuk a klóz irredundáns variánsát, melynek a következő a definíciója:

4.19. Definíció. (Klóz irredundáns variánsa)

Egy C *klóz irredundáns variánsa* egy olyan $C' \subseteq C$ klóz, ahol C' nem redundáns és $C' \sim C$. □

Erre vonatkozóan a következő tételt bizonyítjuk:

4.20. Tétel.

(1) *Bármely C klóznak létezik irredundáns variánsa.*

(2) *Ha C-hez több ilyen variáns létezik, akkor azok egymás átnevezettjei.*

A fejezet végén bevezetjük még a multi-hipertablóban használt kiterjesztésekre vonatkozóan is az irredundáns variáns fogalmát:

4.21. Definíció. (Kiterjesztés irredundáns variánsa)

Egy $\mathcal{T}$ multi-hipertabló $[E, \sigma]$ *kiterjesztésének irredundáns variánsa* alatt egy olyan $[E', \sigma']$ kiterjesztést értünk, hogy

(1) $E' \subseteq E$,

(2) $\sigma' = \{x/t \in \sigma \mid x \in \mathcal{FV}(\langle E' \rangle) \cup \mathcal{FV}(\mathcal{T})\}$ és

(3) $\langle E' \rangle \sigma'$ az $\langle E \rangle \sigma$-nak irredundáns variánsa. □

Az A.2. függelékben algoritmikus megvalósítását adjuk egy klóz, illetve egy kiterjesztés irredundáns variánsa előállításának.

4.3.3. Redundancia multi-hipertablóban

A disszertáció 3.4.3. fejezetében azzal foglalkozunk, hogy *rigid literálos tablóhoz*[4] (és így multi-hipertablóhoz is) hogyan adható meg olyan redundanciafogalom, mely pusztán az aktuális tablón és a tablóhoz csatolandó klózon alapul. A disszertációban rámutatunk, hogy *általánosságban ilyen redundanciafogalom nem létezik.* Viszont bebizonyítjuk a következő tételt:

4.22. Tétel.

Legyen $\mathcal{T}$ a 4.3. ábrán látható alakú rigid literálos tabló, ahol

- *jelölje $\mathcal{B}$ a $b_1 \cup \{L_1\} \cup b$ ágat,*
- *jelölje C az $L_1 \vee L_2 \vee \ldots \vee L_k$ klózt, és legyen $\mathcal{FV}(\langle C \rangle) \cap \mathcal{FV}(t) = \emptyset$.*

Ekkor:

(1) *Ha $\widehat{C}$ a C új példánya, akkor*

$$F(\mathcal{T}) \quad \sim \quad F\big(\mathcal{T} +^{\mathcal{B}} \widehat{C}\big) \quad .$$

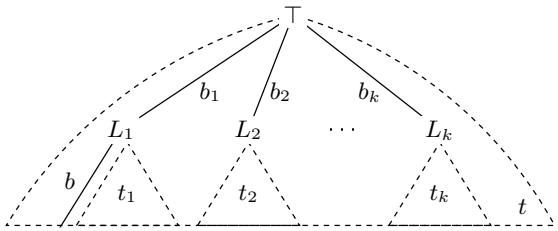
(2) *Ha D olyan klóz, hogy $\langle \widehat{C} \rangle \sigma \subseteq D$ valamely σ helyettesítésre, akkor*

$$F(\mathcal{T}) \quad \sim \quad F\big(\mathcal{T} +^{\mathcal{B}} D\big) \quad .$$

A fenti tétel C-vel jelölt klózának karakterizálására bevezetjük a következő fogalmat:

[4] Rigid változókat tartalmazó és csak literálokkal címkézett tabló.

[5] Az $L_1, \ldots, L_k$ literálok, a $b_1, \ldots, b_k, b$ részágak, a $t_1, \ldots, t_k, t$ pedig résztablók.



4.3. ábra. Rigid literálos tabló[5]

4.23. Definíció. (Független ágklóz)

Legyen $\mathcal{T}$ egy tabló. Legyenek $L_1, \ldots, L_k$ $(k \geq 1)$ olyan literálok a $\mathcal{T}$-ben, hogy

(1) $\mathcal{T}$ minden ága legfeljebb egy L_i-t tartalmaz, és

(2) $\mathcal{T}$ minden olyan $\mathcal{B}$ ága esetén, mely semelyik L_i-t sem tartalmazza:

$$\mathcal{FV}\big(\{L_1, \ldots, L_k\}\big) \cap \mathcal{FV}(\mathcal{B}) \quad = \quad \emptyset \quad .$$

Ekkor az $\{L_1, \ldots, L_k\}$ *független ágklóza* bármely valamely L_i-t tartalmazó ágnak. □

A fenti tétel alapján bármely olyan rigid literálos tablót építő tablókalkulus esetén, melynek valamely *levezetési szabálya biztosítja a független ágklózok új példányainak igény szerinti előállítását*, a következő redundanciafogalmat lehet kimondani:

Redundancia rigid literálos tabló ágára nézve:

A D klóz redundáns a rigid literálos $\mathcal{T}$ tabló $\mathcal{B}$ ágára nézve, ha van olyan $L \in \mathcal{B}$ és van olyan L-et tartalmazó C független ágklóza a $\mathcal{B}$-nek, hogy a D valamely részklóza példánya a C új példányának. □

A disszertációban megmutatjuk, hogy a hipertablóhoz tartozó (a disszertáció 3.4.1. fejezetében bemutatott) redundanciafogalom tulajdonképpen a fenti redundanciafogalomnak egy speciális változata. Azt is elmondjuk, hogy mivel a multi-hipertablónak nincs független ágklózok új példányait előállító levezetési

szabálya, multi-hipertabló esetén a fenti redundanciafogalom nem alkalmazható. A multi-hipertabló kalkulusnak olyan továbbfejlesztése lenne a kívánatos, mely független ágklózok új példányait tudja generálni anélkül, hogy a kalkulus elvesztené a teljességét.

4.4. Heurisztika

A disszertáció 3.5. fejezetében a heurisztikus tételbizonyítás lehetőségeivel foglalkozunk a multi-hipertabló kapcsán. Ennek érdekében egy $\mathcal{B}$ ághoz és egy $\mathcal{C}$ klózhalmazhoz tartozó kiterjesztések $\mathcal{E}(\mathcal{B},\mathcal{C})$ halmazán egy *teljes rendezést* definiálunk [25].

4.24. Definíció. (Heurisztika)

Adott egy $\mathcal{T}$ multi-hipertabló. Ennek egy ágának valamely $[E_1,\sigma_1]$ és $[E_2,\sigma_2]$ kiterjesztései esetén $[E_1,\sigma_1] \leq [E_2,\sigma_2]$, ha

(1) $|E_1| < |E_2|$, vagy

(2) $|E_1| = |E_2|$ és $|R_1| \leq |R_2|$, ahol $R_i = \{x \mid x \in \mathcal{D}om(\sigma_i) \cap \mathcal{FV}(\mathcal{T})\}$. □

4.4.1. Paraméteres tételek

A disszertáció 3.5.1. fejezetében azt vizsgáljuk meg, hogy a *klózok zártságára* vonatkozó kikötésnek az eltörlése vajon milyen következményekkel jár a multi-hipertablóra nézve. Bevezetjük a *paraméterváltozók* fogalmát, és a $\mathcal{P}ar(C)$ jelölést egy C klózban előforduló paraméterváltozók halmazának jelölésére. Erre a klózok *új példányainak* előállítása során lesz szükségünk: az új klózpéldányokat leíró 2.7. definíció módosítandó oly módon, hogy a benne szereplő

$$\mathcal{D}om(\sigma) = \mathcal{FV}(\langle C\rangle)$$

feltételt átírjuk a

$$\mathcal{D}om(\sigma) = \mathcal{FV}(\langle C\rangle)\backslash\mathcal{P}ar(C)$$

feltételre. Megfelelő módosítását adjuk a 4.24. definícióban leírt teljes rendezésnek:

4.25. Definíció. (Heurisztika - Paraméterváltozók használata)

Adott egy $\mathcal{C}$ klózhalmaz $\mathcal{T}$ multi-hipertablója. Ennek egy ágának valamely $[E_1,\sigma_1]$ és $[E_2,\sigma_2]$ kiterjesztései esetén $[E_1,\sigma_1] \leq [E_2,\sigma_2]$, ha

(1) $|P_1| < |P_2|$, ahol $P_i = \{x \mid x \in \mathcal{D}om(\sigma_i) \cap \mathcal{P}ar(\mathcal{C})\}$[6], vagy

(2) $|P_1| = |P_2|$ és $|E_1| < |E_2]$, vagy

(3) $|P_1| = |P_2|$ és $|E_1| = |E_2|$ és $|R_1| \leq |R_2|$, ahol

$$R_i \quad = \quad \{x \mid x \in \mathcal{D}om(\sigma_i) \cap \mathcal{FV}(\mathcal{T})\} \quad . \qquad \square$$

Ezen kívül a disszertációban részletesen kifejtjük az implementációs követelményeit egy olyan multi-hipertablón alapuló tételbizonyítónak, mely kezelni tudja a paraméterváltozókat. Ezen követelmények közé tartozik az ún. *visszaállítási pontok* és a *backtracking* használata. Az A.3. függelékben leírjuk a multi-hipertabló algoritmikus megvalósítását.

[6] Egy $\mathcal{C}$ klózhalmaz esetén $\mathcal{P}ar(\mathcal{C}) = \bigcup_{C \in \mathcal{C}} \mathcal{P}ar(C)$.

1. INTRODUCTION

In the dissertation, we deal with some problem of theorem proving in classical first-order logic. Automated theorem proving methods constitute the basis of theorem proving softwares, experts systems, the complex applications based on expert systems [24, 26, 27, 28], and logical programming languages. In the dissertation, we deal with automating *hyper tableaux* [3] (and consequently, hyper-resolution [21]). In order to solve the problems being focused, we define the *multi-hyper tableau* calculus [25]. The dissertation consists of four sections and one appendix.

In the first section, we give a short overview on those concepts of classical first-order logic which are needed in the subsequent sections.

In the second section, the calculi that have given inspiration to our work are introduced. A few variants of tableau calculi [23, 11, 13] and resolution calculi [20, 2, 21] are presented, so is the *hyper tableau* calculus [3]. There are serious problems with automating hyper tableaux. As a solution for these problems, the *rigid hyper tableau* calculus is cited [14]. Nevertheless, we prove this calculus not to be sound, and point out the fact that the calculus has not been proved to be complete yet.

Our main results can be found in the third section. In the fist part of this section, we define the *multi-hyper tableau* calculus [25], and we prove this calculus to be sound and complete. After this, clause generation is focused; we also propose an own method called the *clause generating tableau* [25]. As all the clause generating methods in literature, this method is exponential. This is why we make it linear by using DAGs. In the fourth part of this section, the concept of *redundancy* is focused. We propose examinations based on redundancy for three similar purposes. First, Baumgartner's *redundancy criterion* for hyper tableaux [3] is obtained. Then, we *reduce clauses* by a redundancy check. Finally, it is detailed how to define an appropriate redundancy criterion for *rigid clausal tableaux* (and consequently, for multi-hyper tableaux as well). In the last part of this section, we deal with *heuristic theorem proving* by multi-hyper tableaux, and then we examine into the consequences of *letting*

clauses be open.

In the last section, we summarize the results of the dissertation, and we throw light upon which components could be improved and which questions could be answered in the future.

In the appendix, the algorithmic implementation of a few method is proposed. First, an own unifying algorithm is proposed. Then, a redundancy-based reduction of clauses is implemented. Finally, we implement the multi-hyper tableau calculus.

2. PRELIMINARIES

In the dissertation, we deal solely with classical first-order logic. Joint concepts are mostly interpreted as is usual in literature; the other ones are defined explicitly.

Given a formula or a formula set F, let $\mathcal{FV}(F)$ denote the *set of the free variables* in F. F is *closed* iff $\mathcal{FV}(F) = \emptyset$; otherwise, F is *open*. The (universal) *closure* of a formula F is denoted by $\forall F$.

Definition 2.1 (Variable Disjunct Formulas).

Two formulas A and B are *variable disjunct* iff $\mathcal{FV}(A) \cap \mathcal{FV}(B) = \emptyset$. A formula set is variable disjunct iff its formulas are pair-wise variable disjunct. □

In the dissertation, *(term) substitutions* are defined as is usual in literature. The empty substitution is denoted by ϵ. It is also defined in the usual way when and how to apply a substitution to a formula or a formula set.

Definition 2.2 (Instance).

A formula A is an *instance* of a formula B iff there is a substitution σ such that $A = B\sigma$. □

The following two definitions concern a special class of substitutions, namely the (variable) renamings.

Definition 2.3 (Renaming).

A *renaming* is a substitution σ where

(1) for any $x \in \mathcal{D}om(\sigma)$, $\sigma(x)$ is a variable;

(2) for any $x, y \in \mathcal{D}om(\sigma)$, if $x \neq y$ then $\sigma(x) \neq \sigma(y)$. □

Definition 2.4 (Renamed Variant).

A formula A is a *renamed variant* of a formula B iff there is a renaming σ such that $A = B\sigma$. □

Since we deal mostly with *calculi handling clauses* in the dissertation, it is expedient to define literals and clauses explicitly.

Definition 2.5 (Literal).

A *literal* is a formula A or $\neg A$ where A is an atom. *Positive and negative literals* are distinguished:

(1) A is positive iff $A \neq \bot$;

(2) $\neg A$ is positive iff A is negative.

The *base* of a literal L is denoted by $\underline{L}$, and is defined as follows:

$$\underline{A} = \underline{\neg A} \quad = \quad \begin{cases} \top & \text{, if } A = \bot; \\ A & \text{, otherwise.} \end{cases}$$ □

Definition 2.6 (Clause).

A *clause* is a *closed* formula $C = \forall x_1 \ldots \forall x_k (L_1 \vee L_2 \vee \ldots \vee L_n)$ where each L_i is a literal, $k \geq 0$, $n \geq 0$. The *core* of C is the formula $L_1 \vee L_2 \vee \ldots \vee L_n$, and is denoted by $\langle C \rangle$. □

Since all clauses are closed formulas, it does not give rise to a misinterpretation to eliminate the quantifiers; i.e., $L_1 \vee L_2 \vee \ldots \vee L_n$ (which is actually the core of C) can be used instead of C. The same clause can be denoted and used as a *multiset of its literals*, i.e., as $\{L_1, L_2, \ldots, L_n\}$.

The *empty clause* (i.e., when $n = 0$) is denoted by $\bot$.

A clause C is *positive* (*negative*) if each literal $L \in C$ is positive (negative).

The calculi handling clauses usually generate new instances of clauses, which are defined as follows:

Definition 2.7 (New Instance of a Clause).

A *new instance* of a clause C – w.r.t. a variable set $\mathcal{V}$ – is an instance $\langle C \rangle \sigma$ of C such that

(1) σ is a renaming;

(2) $\mathcal{D}om(\sigma) = \mathcal{FV}\big(\langle C \rangle\big)$;

(3) $\mathcal{R}ange(\sigma) \cap \mathcal{V} = \emptyset$. □

It is denoted by $\mathcal{M} \models F$ that a formula F is *true in a model* $\mathcal{M}$. If F is an open formula, $\mathcal{M} \models F$ iff $\mathcal{M} \models \forall F$.

Two formula A and B are *equivalent*: for any model $\mathcal{M}$, $\mathcal{M} \models A$ iff $\mathcal{M} \models B$. Notation: $A \sim B$.

A formula K is a *logical consequence* of some formulas $P_1, \ldots, P_n$ $(n \geq 0)$: for any model $\mathcal{M}$ such that $\mathcal{M} \models P_i$ for all P_i, it holds that $\mathcal{M} \models K$. Notation: $P_1, \ldots, P_n \models K$.

3. THEOREM PROVING METHODS

In Section 2 in the dissertation, a survey of the calculi to which our research refers can be found.

3.1. Tableaux

In Section 2.1 in the dissertation, Smullyan's *analytic tableaux* [23] are introduced. A tableau – as is usual in literature – is a tree labeled with formulas. In some cases (when the following notation is more advantageous), a branch of a tableau is regarded as a *multiset of the formulas* labeling the nodes in the given branch. Similarly, a tableau is often regarded as a *multiset of its branches*.

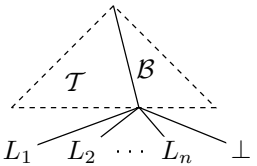
In Section 2.1.1, Fitting's *free-variable tableaux* [11] are detailed. In this calculus, we refer to the unifying algorithm invented by Robinson [20]. The most general unifier (MGU) is generated by an own recursive unifying algorithm proposed in Appendix A.1. This algorithm realizes the function $\mathcal{U}()$, which assigns to pairs of expressions $(E_1, F_1), \ldots, (E_n, F_n)$ (where $n \geq 0$) their MGU, if it exists.

In Section 2.1.2, *clause tableaux* [13] are introduced. This calculus is special in the sense that it attaches clauses to the branches of tableaux during the derivation. Let us introduced the following notation for this special operation:

Definition 3.1 (Attaching a Clause to a Branch of a Tableau).

Attaching a clause $C = \{L_1, L_2, \ldots, L_n\}$ to a branch $\mathcal{B} \in \mathcal{T}$ of a tableau $\mathcal{T}$ results in the following tableau:

$\mathcal{T} +^{\mathcal{B}} C \quad =$



□

3.2. Resolution

In Section 2.2 in the dissertation, Robinson's (binary) *resolution* [20, 2] is introduced. In Section 2.2.1, the strategy of *linear input resolution* is detailed, which is *not complete* in first-order logic (and it is only in the case of Horn clauses).

In Section 2.2.2, Robinson's *hyper-resolution* [21] is introduced, which is yet *complete* in first-order logic. Although hyper-resolution is effective in theory, it is difficult to automate.

3.3. Hyper Tableaux

Hyper tableau calculi intend to automate hyper-resolution upon realizing that the tree data structure used by *tableau* calculi can be applied with success to describing hyper-resolvents. In comparison with the queue data structure used by resolution calculi, it is now possible to store the „history" of the derivation in a more structured form. That is, hyper tableaux can be regarded as combinations of *hyper-resolution and clause tableaux.*

Hyper tableau calculi are new results of state-of-the-art research on theorem proving [6, 3, 4]. Although the feasibility of such a calculus had already been raised at the end of the 70's [5], a hyper tableau calculus in complete form was published only in 1996 (Baumgartner [3]). In Section 2.3 in the dissertation, Baumgartner's *hyper tableaux* are introduced, and we summarize them as follows:

- The hyper tableau calculus is sound and complete [3].
- Factoring is not needed for the completeness of the calculus (contrary with hyper-resolution).
- The use of so-called *purifying substitutions* is the weak point of hyper tableaux, since their generation is not automated.

Several authors have tried to eliminate purifying substitutions in hyper tableaux [4, 14, 10]. In Section 2.3.1 in the dissertation, Kühn's *rigid hyper tableaux* [14] are introduced by way of example. Kühn realized that hyper tableaux without purifying substitutions are complete only in the cases of Horn clauses. In order to eliminate purifying substitutions, Kühn uses rigid variables in tableaux, and incorporates in his calculus the possibility of repeating clauses extracted from the tableau (the so-called branch clauses). Let us summarize the rigid hyper tableau calculus in the following way:

- The *instantiation of branch clauses* is not a necessary component of the calculus.
- Rigid hyper tableaux are *not sound*. We prove this fact in Theorem 2.3.15 in the dissertation.
- It has not been proved yet that rigid hyper tableaux are *complete*.

4. MULTI-HYPER TABLEAUX

Our main results can be found in Section 3 in the dissertation. In Section 3.1, multi-hyper tableaux are introduced, which have been published in the paper [25]. The multi-hyper tableau calculus is an improvement of Baumgartner's hyper tableaux and Kühn's rigid hyper tableaux. It

(1) eliminates purifying substitutions, which are used in hyper tableaux;

(2) eliminates the problem of branch clauses copies, which are used in rigid hyper tableaux;

(3) intends to solve the efficiency problems of hyper tableaux and rigid hyper tableaux;

(4) and is proved to be sound and complete.

4.1. The Multi-Hyper Tableau Calculus

In the followings, we define the multi-hyper tableau calculus. First of all, we give two definitions, which describe what is meant by an extension of a branch $\mathcal{B}$ of a tableau $\mathcal{T}$.

Definition 4.1 (Extension).

An *extension* is a tuple $[E, \sigma]$ where E is a clause and σ is a substitution. □

Definition 4.2 (Applying an Extension).

An *extension* $[E, \sigma]$ is *applied* to a branch $\mathcal{B} \in \mathcal{T}$ of a tableau $\mathcal{T}$ by generating the following tableau:

$$(\mathcal{T} +^{\mathcal{B}} E)\sigma \quad .$$ □

An extension is assigned to three objects at the same time: to a clause, a clause multiset, and a literal multiset. The assignment is described by the following definition:

Definition 4.3 (Determining an Extension).

(1) Let $C^{\ominus}$ be a clause, and let $\{L_1^{\ominus}, \ldots, L_k^{\ominus}\}$ be the set of all the negative literals in $C^{\ominus}$, where $k \geq 1$.

(2) Let $\{C_1^{\oplus}, \ldots, C_k^{\oplus}\}$ be a multiset of positive clauses.

(3) Let $\{L_1^{\oplus}, \ldots, L_k^{\oplus} \mid L_i^{\oplus} \in C_i^{\oplus}\}$ be a multiset of (positive) literals, where $\sigma = \mathcal{U}\big((\underline{L_1^{\ominus}}, \underline{L_1^{\oplus}}), \ldots, (\underline{L_k^{\ominus}}, \underline{L_k^{\oplus}})\big)$ exists.

Let E denote the following clause:

$$C^{\ominus} \backslash \{L_1^{\ominus}, \ldots, L_k^{\ominus}\} \ \cup \ \bigcup_{i=1}^{k} C_i^{\oplus} \backslash \{L_i^{\oplus}\} \quad . \tag{4}$$

Then, the extension assigned to $C^{\ominus}$, $\{C_1^{\oplus}, \ldots, C_k^{\oplus}\}$, and $\{L_1^{\oplus}, \ldots, L_k^{\oplus}\}$ is referred as

$$e\Big(C^{\ominus}, \big\{C_1^{\oplus}, \ldots, C_k^{\oplus}\big\}, \big\{L_1^{\oplus}, \ldots, L_k^{\oplus}\big\}\Big) \quad ,$$

and is $[E, \sigma]$. □

The following definition describes to which clauses, clause multisets, and literal multisets extensions can be generated, in the case of a given branch and a given input clause set.

Definition 4.4 (Extensions for a Branch and a Clause Set).

For a *branch* $\mathcal{B}$ and a *clause set* $\mathcal{C}$, the *extensions* are the elements of the following set:

$$\mathcal{E}(\mathcal{B}, \mathcal{C}) = \left\{ e\Big(\widehat{C}^{\ominus}, \big\{\widehat{C}_1^{\oplus}, \ldots, \widehat{C}_k^{\oplus}\big\}, \big\{L_1^{\oplus}, \ldots, L_k^{\oplus}\big\}\Big) \;\middle|\; \begin{array}{ll} C^{\ominus} \in \mathcal{C}, & (1) \\ C_i^{\oplus} \in \mathcal{B} \cup \mathcal{C}, & (2) \\ L_i^{\oplus} \in \widehat{C}_i^{\oplus} & (3) \end{array} \right\} \quad .$$

Hereinbefore, for any clause C

$$\widehat{C} \quad = \quad \begin{cases} \text{a new instance}^{1} \text{ of } C & \text{, if } C \in \mathcal{C}; \\ C & \text{, otherwise.} \end{cases}$$

□

[1] These instances are new w.r.t. the variable set $\mathcal{V} = \mathcal{FV}(\mathcal{T})$ (c.f. Definition 2.7), where $\mathcal{T}$ is the current tableau.

Now, we use the previous definitions for defining the multi-hyper tableau calculus:

Definition 4.5 (Multi-Hyper Tableau).

Let $\mathcal{C}$ be a clause set. For $\mathcal{C}$, a *positive multi-hyper tableau* is defined as follows:

(1) Initialization Rule:
$\{\{\top\}\}$ is a positive multi-hyper tableau for $\mathcal{C}$.

(2) Extension Rule:

(a) Let $\mathcal{T}$ be a positive multi-hyper tableau for $\mathcal{C}$, and let $\mathcal{B} \in \mathcal{T}$.

(b) Let $e \in \mathcal{E}(\mathcal{B}, \mathcal{C})$ be an extension.

Then, the tableau resulted by applying e to $\mathcal{B}$ in $\mathcal{T}$ is a positive multi-hyper tableau for $\mathcal{C}$. □

Remark 4.6.

The definition of a *negative multi-hyper tableau* can be obtained by exchanging the words „positive" and „negative" in Definition 4.3, Definition 4.4, and Definition 4.5. □

The soundness of multi-hyper tableaux can be easily proved on the basis of the soundness of hyper-resolution.

Theorem 4.7 (Multi-Hyper Tableaux – Soundness).

The multi-hyper tableau calculus is sound. □

It is much more difficult to prove multi-hyper tableaux to be complete. In Kühn's rigid hyper tableaux, even more than one clause can be attached to the tableau in a single derivation step, and the clauses that can be resolved with each other are tried to be attached to the same branches in order to close these branches. Contrarily, the multi-hyper tableau calculus computes a hyper-resolvent in a derivation step (an extension $[E, \sigma]$ corresponds to the hyper-resolvent $\langle E \rangle \sigma$), and attaches it to the tableau. This latter approach yields a simple and transparent calculus, since the tableau gets extended with *only one clause* at once. The more important advantage of this approach manifests itself in the proof of completeness: the completeness of multi-hyper tableaux can be proved on the basis of the completeness of hyper-resolution.

Completeness is proved in two steps:

(1) In Section 3.2.1. in the dissertation, we show how to construct a so-called *hyper-resolution graph* from a given hyper-resolution refutation. The concept of hyper-resolution graph is new in literature, we define it by means of several own concepts, like *grouped graphs*, *inference node-groups*, and $\mathcal{N}$*-edges*.

(2) In Section 3.2.2 in the dissertation, a *closed multi-hyper tableau* is generated from a given hyper-resolution graph.

Theorem 4.8 (Multi-Hyper Tableaux – Completeness).

The multi-hyper tableau calculus is complete. □

4.2. Clause Generation by Tableaux

Most of the clause generating algorithms rewrites directly the formulas according to well-known logical equivalences [16]. As rare exceptions, there are other alternative methods. Some of them are based on BDDs (Binary Decision Diagrams), also known as Shannon-graphs [16, 19]. BDDs can basically used in propositional logic. Often, logicians draw a parallel between BDDs and tableaux [19]. Similarly, a parallel can be drawn between tableaux and the clause generating algorithms based on pp-expressions, the dual forms of formulas, and embedded lists [11, 17]. However, these methods can only be used in propositional logic, either.

The correspondence between tableaux and normal forms is well-known in literature, but only in propositional logic. Given the finished *analytic dual tableau* generated for a propositional formula, one can obtain the clause normal form of the formula in the following way: for each branch, generate the disjunction of the literals occurring in the branch; and then, generate the conjunction of the resulting disjunctive formulas.

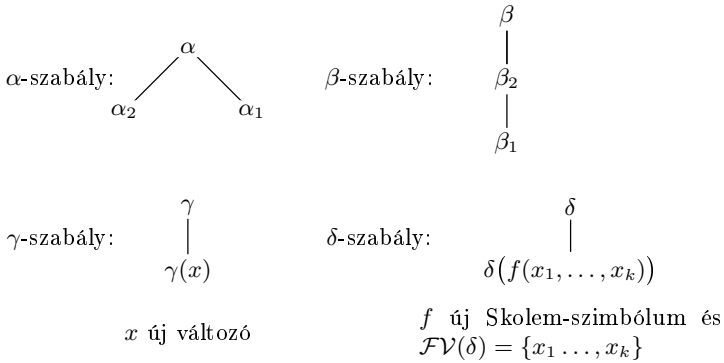
As pointed out in Section 3.3 in the dissertation, the derivation rules of a clause generating tableau method in first-order logic have to take out the prenexing and the skolemizing of formulas. For this purpose, we use the γ-rules and the δ-rules of *free-variable tableaux* [11].

Definition 4.9 (Clause Generating Tableaux).

Let $\mathcal{F} = \{F_1, \ldots, F_n\}$ be a formula set. For $\mathcal{F}$, a *clause generating tableau* [25] is inductively defined as follows:

(1) $\big\{\{F_1\},\ldots,\{F_n\}\big\}$ is a clause generating tableau for $\mathcal{F}$.

(2) If $\mathcal{T}$ is a clause generating tableau for $\mathcal{F}$ then so is $\mathcal{T}'$, if $\mathcal{T}'$ is obtained by applying any rule that can be seen in Figure 4.1 to $\mathcal{T}$. □



4.1. ábra. Klózgeneráló tabló – levezetési szabályok

It is important that each formula to that a rule has been applied gets erased (as a label) from the branch. A clause generating tableau is said *finished* if it consists of solely literals. It is to prove that a finished clause generating tableau for a formula set $\mathcal{F}$ generates a clause set which is satisfiable iff so is $\mathcal{F}$. Thereto, we define a function that assigns a formula to a tableau, and then we prove a lemma.

Definition 4.10.

The function $cnf()$ on the set of all branches and tableaux is defined as follows:

(1) Given a branch $\mathcal{B} = \{A_1,\ldots,A_k\}$, $cnf(\mathcal{B}) = (A_1 \vee \ldots \vee A_k)$.

(2) Given a tableau $\mathcal{T} = \{\mathcal{B}_1,\ldots,\mathcal{B}_k\}$, $cnf(\mathcal{T}) = cnf(\mathcal{B}_1) \wedge \ldots \wedge cnf(\mathcal{B}_k)$.□

Lemma 4.11.

Given a clause generating tableau $\mathcal{T}$, $cnf(\mathcal{T})$ is satisfiable iff so is $cnf(\mathcal{T}')$ where the tableau $\mathcal{T}'$ has been obtained by applying a tableau rule to $\mathcal{T}$. □

On the basis of the previous lemma, it is easy to prove the following theorem:

Theorem 4.12.
A formula set $\mathcal{F}$ is unsatisfiable iff so is

$$\mathcal{C} \quad = \quad \{cnf(\mathcal{B}) \mid \mathcal{B} \in \mathcal{T}\} \quad ,$$

where $\mathcal{T}$ is a clause generating tableau for $\mathcal{F}$. □

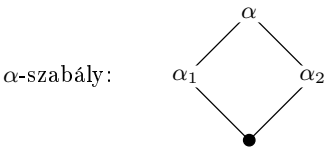
Notice that if $\mathcal{T}$ is *finished* in the above theorem then $\mathcal{C}$ is a *clause set*. That is, the clause set $\mathcal{C}$ is unsatisfiable iff so is $\mathcal{F}$.

4.2.1. Linear Clause Generation

As all the clause generating algorithms in literature, the method of clause generating tableaux is *exponential*. The problem is that some formulas may occur in several branches of the tableau. This problem is caused by the tableau – as a *tree data structure* – itself. Hence, it would be advantageous to modify trees not to contain any loop; in literature, this kind of data structure is known as DAG[2].

DAGs are well-known by logical applications. Robinson's exponential unifying algorithm was linearized in the 70's by Paterson and Wegman [18] by the use of DAGs. DAGs are also applied successfully in other fields of logic, e.g., in modal tableau methods [7].

In Section 3.3.1 in the dissertation, we linearize the clause generation tableau method by applying DAGs, by modifying its α-rule as can be seen in Figure 4.2.



4.2. ábra. Klózgeneráló DAG – módosult α-szabály

It is important how to apply a rule of clause generation tableaux: the new node(s) get(s) inserted just *below the node* to which the rule is being applied.

It is obvious that this method has *linear complexity* since the number of the rule applications needed for constructing the finished clause generating DAG is linear with the number of the logical connectives in the input formula set.

[2] directed acyclic graph

4.2.2. Reducing Clauses by Closing Branches

In Section 3.3.2 in the dissertation, we show how easy to detect *valid clauses* in a finished clause generation tableau simply by *closing branches*. In the case of a branch $\mathcal{B}$:

> If there are two oppositely signed literals $L_1, L_2 \in \mathcal{B}$ such that $\mathcal{U}(\underline{L_1}, \underline{L_2})$ exists then $\mathcal{B}$ (as a clause) does not get to the clause set.

4.3. Redundancy

In Section 3.4 in the dissertation, we define the concept of redundancy (on formulas):

Definition 4.13 (Redundancy).

A *formula* A is *redundant* w.r.t. a formula B and a symmetrical binary logical connective $\circ$ iff $A \circ B \sim B$. □

We use this redundancy concept for three similar purposes.

4.3.1. Redundancy in Hyper Tableaux

In Section 3.4.1 in the dissertation, Baumgartner's *redundancy concept for hyper tableaux* [3] is introduced and obtained from the above-mentioned redundancy concept. For this reason, we prove the following theorem:

Theorem 4.14.

Let A and $B_1 \wedge \ldots \wedge B_k$ be formulas. If there is a substitution σ for any B_i such that $A = B_i\sigma$ then A is redundant w.r.t. $B_1 \wedge \ldots \wedge B_k$ and $\wedge$. □

On the previous theorem, the redundancy concept of hyper tableaux is based:

Definition 4.15 (Redundancy w.r.t. a Hyper Tableau Branch).

A *clause* D is *redundant* w.r.t. a *branch* $\mathcal{B}$ of a hyper tableau iff there is a substitution σ for any $L_1 \in D$ and $L_2 \in \mathcal{B}$ such that $L_1 = \widehat{L}_2\sigma$ [3]. □

[3] The notation $\widehat{L}$ refers a new instance of the literal L (or to be more precise, $\widehat{L}$ is the single literal of a new instance of the clause $\forall L$).

4.3.2. Clause Reduction Based on Redundancy

In Section 3.4.2 in the dissertation, we propose a kind of *clause reduction based on redundancy* [25]. For this purpose, let us introduce the following theorem and definition:

Theorem 4.16.

Let A and $B_1 \vee \ldots \vee B_k$ be formulas. If

$$\mathcal{FV}(A) \cap \mathcal{FV}(B_1 \vee \ldots \vee B_k) \quad = \quad \emptyset$$

and there is a substitution σ for any B_i such that $A\sigma = B_i$ then A is redundant w.r.t. $B_1 \vee \ldots \vee B_k$ and $\vee$. □

Definition 4.17 (Redundancy w.r.t. a Clause).

A *clause* D is redundant w.r.t. a clause C iff $\mathcal{FV}(\langle C \rangle) \cap \mathcal{FV}(\langle D \rangle) = \emptyset$ and there is a substitution σ such that $\langle D \rangle \sigma \subseteq C$. □

The concept of a redundant clause is based on the previous definition:

Definition 4.18 (Redundant Clause).

A clause C is a *redundant clause* iff there is a clause $D \subset C$ such that D is redundant w.r.t. the clause $C \backslash D$. □

The point of the reduction being discussed is to *eliminate the redundancy* of the given clause, i.e., to generate the so-called irredundant variant of the clause.

Definition 4.19 (Irredundant Variant of a Clause).

An *irredundant variant* of a clause C is a clause $C' \subseteq C$ such that C' is not redundant and $C' \sim C$. □

For such variants, the following theorem is proved:

Theorem 4.20.

(1) Of any clause C, there is an irredundant variant.

(2) If more than one such variants exist then they are renamed variants of each other. □

At the end of the section, the concept of an irredundant variant is extended to the extensions used in multi-hyper tableaux:

Definition 4.21 (Irredundant Variant of an Extension).

By an *irredundant variant of the extension* $[E, \sigma]$ for a multi-hyper tableau $\mathcal{T}$, we mean an extension $[E', \sigma']$ such that

(1) $E' \subseteq E$;

(2) $\sigma' = \{x/t \in \sigma \mid x \in \mathcal{FV}(\langle E' \rangle) \cup \mathcal{FV}(\mathcal{T})\}$;

(3) and $\langle E' \rangle \sigma'$ is an irredundant variant of $\langle E \rangle \sigma$. □

In Appendix A.2, it is algorithmically implemented to generate an irredundant variant of a clause or an extension.

4.3.3. Redundancy in Multi-Hyper Tableaux

In Section 3.4.3 in the dissertation, it is discussed how to introduce a redundancy concept for *rigid clausal tableaux*[4] (and consequently, for multi-hyper tableaux) only by considering the current tableau and the clause being attached to the tableau. In the dissertation, we point out that such a *redundancy concept does not exist in general*. However, we prove the following theorem:

Theorem 4.22.

Let $\mathcal{T}$ be a rigid clausal tableau in the form that can be seen in Figure 4.3, where

- let $\mathcal{B}$ denote the branch $b_1 \cup \{L_1\} \cup b$;
- let C denote the clause $L_1 \vee L_2 \vee \ldots \vee L_k$, and let $\mathcal{FV}(\langle C \rangle) \cap \mathcal{FV}(t) = \emptyset$.

The following facts hold:

(1) if $\widehat{C}$ is a new instance of C then

$$F(\mathcal{T}) \quad \sim \quad F\big(\mathcal{T} +^{\mathcal{B}} \widehat{C}\big) \quad .$$

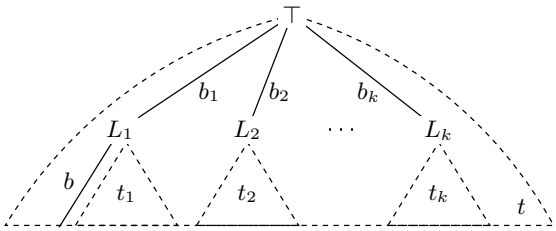
(2) If D a clause such that $\langle \widehat{C} \rangle \sigma \subseteq D$ for a substitution σ then

$$F(\mathcal{T}) \quad \sim \quad F\big(\mathcal{T} +^{\mathcal{B}} D\big) \quad . \qquad \square$$

In order to characterize the clause C in the previous theorem, let us introduce the following concept:

[4] Tableaux that contain rigid variables and are labeled only with literals.

[5] $L_1, \ldots, L_k$ are literals, $b_1, \ldots, b_k, b$ are subbranches, and $t_1, \ldots, t_k, t$ are subtableaux.



4.3. ábra. Rigid literálos tabló[5]

Definition 4.23 (Separate Branch Clause).

Let $\mathcal{T}$ be a tableau. Let $L_1, \ldots, L_k$ $(k \geq 1)$ be literals in $\mathcal{T}$ such that

(1) any branch of $\mathcal{T}$ contains at most one L_i;

(2) for all branch $\mathcal{B}$ of $\mathcal{T}$ such that $\mathcal{B}$ does not contain any L_i, it holds that

$$\mathcal{FV}\big(\{L_1, \ldots, L_k\}\big) \cap \mathcal{FV}(\mathcal{B}) \quad = \quad \emptyset \quad .$$

$\{L_1, \ldots, L_k\}$ is a *separate branch clause* of any branch containing any L_i. □

According to the above theorem, the following redundancy concept can be formulated for any rigid clausal tableau calculus which has such a *derivation rule that provides the generation of new instances of separate branch clauses.*

Redundancy w.r.t. a Rigid Clausal Tableau Branch:

A clause D is redundant w.r.t. a branch $\mathcal{B}$ of a rigid clausal tableau $\mathcal{T}$ iff there is an $L \in \mathcal{B}$ and is a separate branch clause C of $\mathcal{B}$ such that C contains L and the following statement holds: some subclause of D is an instance of a new instance of C. □

In the dissertation, we show that the redundancy concept for hyper tableaux (c.f. Section 3.4.1 in the dissertation) is actually a specialization of the above redundancy concept. We also point out that the multi-hyper tableau calculus has no derivation rule providing the generation of new instances of separate branch clauses, therefore the above redundancy concept cannot be applied for multi-hyper tableaux. It would be expedient to improve the multi-hyper tableau calculus to be able to generate new instances of separate branch clauses, without losing completeness.

4.4. Heuristics

In Section 3.5 in the dissertation, we deal with the potential of heuristic theorem proving in connection with multi-hyper tableaux. With this object, we define a *total ordering* on the set $\mathcal{E}(\mathcal{B},\mathcal{C})$ of extensions for a given branch $\mathcal{B}$ and clause set $\mathcal{C}$ [25].

Definition 4.24 (Heuristics).

Let $\mathcal{T}$ be a multi-hyper tableau. Given two extension $[E_1,\sigma_1]$ és $[E_2,\sigma_2]$ for any branch of $\mathcal{T}$, $[E_1,\sigma_1] \leq [E_2,\sigma_2]$ holds iff

(1) $|E_1| < |E_2|$ or

(2) $|E_1| = |E_2|$ and $|R_1| \leq |R_2|$, where $R_i = \{x \mid x \in \mathcal{D}om(\sigma_i) \cap \mathcal{FV}(\mathcal{T})\}$.

□

4.4.1. Parametrized Theorems

In Section 3.5.1 in the dissertation, it is discussed what consequences the elimination of stipulating *clauses to be closed* has w.r.t. multi-hyper tableaux. We introduce the concept of *parametric variables*, and the notation $\mathcal{P}ar(C)$ for the set of parametric variables in a clause C. We need this concept and notation when generating *new instances* of clauses: the definition of new clause instances (Definition 2.7) must be modified such that the statement

$$\mathcal{D}om(\sigma) = \mathcal{FV}\big(\langle C\rangle\big)$$

is to be overwritten with the statement

$$\mathcal{D}om(\sigma) = \mathcal{FV}\big(\langle C\rangle\big)\backslash\mathcal{P}ar(C) \quad .$$

We also modify the total ordering that has been defined in Definition 4.24:

Definition 4.25 (Heuristics - Using Parametric Variables).

Let $\mathcal{C}$ be a clause set, and $\mathcal{T}$ a multi-hyper tableau for $\mathcal{C}$. Given two extension $[E_1,\sigma_1]$ és $[E_2,\sigma_2]$ for any branch of $\mathcal{T}$, $[E_1,\sigma_1] \leq [E_2,\sigma_2]$ holds iff

(1) $|P_1| < |P_2|$, where $P_i = \{x \mid x \in \mathcal{D}om(\sigma_i) \cap \mathcal{P}ar(\mathcal{C})\}$[6], or

[6] Given a clause set $\mathcal{C}$, $\mathcal{P}ar(\mathcal{C}) = \bigcup_{C\in\mathcal{C}} \mathcal{P}ar(C)$.

(2) $|P_1| = |P_2|$ and $|E_1| < |E_2]$, or

(3) $|P_1| = |P_2|$ and $|E_1| = |E_2|$ and $|R_1| \leq |R_2|$, where

$$R_i \quad = \quad \left\{ x \;\middle|\; x \in \mathcal{D}om(\sigma_i) \cap \mathcal{FV}(\mathcal{T}) \right\} \quad . \qquad \square$$

In the dissertation, the implementational requirements for a theorem prover based on multi-hyper tableaux that handles parametric variables are also detailed. The use of *backtracking* and so-called *rollback points* is one of these requirements. In Appendix A.3, we give the algorithmic implementation of the multi-hyper tableau calculus.

IRODALOMJEGYZÉK

[1] H. Andréka, I. Németi, J. van Benthem, „*Modal languages and bounded fragments of predicate logic*". Journal of Philosophical Logic, Vol. 27, p. 217-274, 1998.

[2] L. Bachmair, H. Ganzinger, „*Resolution Theorem Proving*". Handbook of Automated Reasoning, by J. A. Robinson and A. Voronkov, Vol. 1, Chapter 2, p. 19-99, Elsevier and MIT Press, 2001.

[3] P. Baumgartner, U. Furbach, I. Niemelä, „*Hyper Tableaux*". Lecture Notes in Computer Science, Vol. 1126, p. 1-17, 1996.

[4] P. Baumgartner, „*Hyper Tableaux – The Next Generation*". Lecture Notes in Artificial Intelligence, Vol. 1397, p. 60-76, 1998.

[5] F. M. Brown, „*Towards the Automation of Set Theory and its Logic*". Artificial Intelligence, Vol. 10, p. 281-316, 1978.

[6] F. Bry, A. Yahya, „*Positive Unit Hyperresolution Tableaux and Their Application to Minimal Model Generation*". Journal of Automated Reasoning, Vol. 25, p. 35-82, 2000.

[7] M. A. Castilho, L. F. del Cerro, O. Gasquet, A. Herzig, „*Modal Tableaux with Propagation Rules and Structural Rules*". Fundamenta Informaticae, Vol. 32, p. 281-297, 1997.

[8] C. L. Chang, R. C. T. Lee, „*Symbolic Logic and Mechanical Theorem Proving*". Academic Press, 1973.

[9] A. Church, „*A Note on the Entsheidungsproblem*". Journal of Symbolic Logic, Vol. 1, p. 40-41, 101-102, 1936.

[10] J. van Eijck, „*Constrained Hyper Tableaux*". Lecture Notes in Computer Science, Vol. 2142, p. 232-246, 2001.

[11] M. Fitting, „*First-Order Logic and Automated Theorem Proving*". Springer-Verlag, 1996.

[12] J. H. Gallier, „*Logic for Computer Science – Foundations of Automatic Theorem Proving*". Wiley & Sons, 2003.

[13] R. Hähnle, „*Tableaux and Related Methods*". Handbook of Automated Reasoning, by J. A. Robinson and A. Voronkov, Vol. 1, Chapter 3, p. 100-178, Elsevier and MIT Press, 2001.

[14] M. Kühn, „*Rigid Hypertableaux*". Lecture Notes in Artificial Intelligence, Vol. 1303, p. 87-98, 1997.

[15] R. Letz, K. Mayr, C. Goller, „*Controlled Intergration of the Cut Rule into Connection Tableau Calculi*". Journal of Automated Reasoning, Vol. 13, p. 297-328, 1994.

[16] A. Nonnengart, C. Weidenbach, „*Computing Small Clause Normal Forms*". Handbook of Automated Reasoning, by J. A. Robinson and A. Voronkov, Vol. 1, Chapter 6, p. 335-367, Elsevier and MIT Press, 2001.

[17] K. Pásztorné Varga, M. Várterész, „*A Generalized Approach to the Theorem Proving Methods*". 5th International Conference on Applied Informatics (ICAI 2001), p. 191-200, Hungary, 2001.

[18] M. S. Paterson, M. N. Wegman, „*Linear Unification*". Annual ACM Symposium on Theory of Computing, p. 181-186, 1976.

[19] J. Posegga, P. H. Schmitt, „*Automated Deduction with Shannon Graphs*". Journal of Logic and Computation, Vol. 5, p. 697-729, 1995.

[20] J. A. Robinson, „*A Machine-Oriented Logic Based on the Resolution Principle*". Journal of the ACM, Vol. 12, p. 23-41, 1965.

[21] J. A. Robinson, „*Automated Deduction with Hyper-Resolution*". International Journal of Computer Mathematics, Vol. 1, p. 227-234, 1965.

[22] T. Skolem, „*Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit matematischer Sätze nebst einem Theoreme über dichte Mengen*". Videnskabsakademiet i Kristiania, Skrifter I, No. 4, p. 1-36, 1919.

[23] R. M. Smullyan, „*First-Order Logic*". Springer-Verlag, 1968.

KOVÁSZNAI GERGELY PUBLIKÁCIÓI

Referált folyóiratban megjelent publikációk

[24] G. Kovásznai, C. Kotropoulos, I. Pitas, „*CAML – A Universal Configuration Language for Dialogue Systems*". Lecture Notes in Computer Science, Vol. 2736, p. 896 - 906, Springer, 2003.

[25] G. Kovásznai, „*HyperS Tableaux – Heuristic Hyper Tableaux*". Acta Cybernetica, Vol. 17, p. 325-338, 2005, (ZBL#1099.68096, MR#2183822).

Egyéb publikációk

[26] G. Kovásznai, C. Kotropoulos, I. Pitas, „*Partly-Specified Priority Patterns in Natural Language Parsing within Dialogue Systems*". 1st International Workshop on Interactive Rich Media Content Production: Architectures, Technologies, Applications, Tools (Richmedia 2003), p. 161-168, Lausanne, Switzerland, 2003.

[27] G. Kovásznai, C. Kotropoulos, I. Pitas, „*A Novel Universal Language for Configurating Dialogue Systems*". 9th Panhellenic Conference on Informatics (PCI '03), p. 36-45, Thessaloniki, Greece, 2003.

[28] G. Kovásznai, „*Algorithmic Improvements in Natural Language Parsing within Dialogue Systems: Priority Patterns and Wildcards*". 6th International Conference on Applied Informatics (ICAI 2004), Vol. 2, p. 129-138, Eger, Hungary, 2004.

Előadások

[29] G. Kovásznai, „*A SOFIA tételbizonyító*" (in Hungarian). XXV. OTDK, Eger, Hungary, 2001.

[30] G. Kovásznai, K. Veréb, „*Mathematical Morphology in Image Processing by SLD Resolution*". CSCS 2002, Szeged, Hungary, 2002.

[31] G. Kovásznai, C. Kotropoulos, I. Pitas, „*CAML – A Universal Configuration Language for Dialogue Systems*". 14th Conference on Database and Expert System Applications (DEXA 2003), Prague, Czech Republic, 2003.

[32] G. Kovásznai, C. Kotropoulos, I. Pitas, „*Partly-Specified Priority Patterns in Natural Language Parsing within Dialogue Systems*". Richmedia 2003, Lausanne, Switzerland, 2003.

[33] G. Kovásznai, C. Kotropoulos, I. Pitas, „*A Novel Universal Language for Configurating Dialogue Systems*". PCI '03, Thessaloniki, Greece, 2003.

[34] G. Kovásznai, „*Universal Configuration Language and Core-Architecture for Dialogue Systems (Univerzális konfigurációs nyelv és architektúra párbeszédes rendszerekhez)*" (in Hungarian). Magyar számítógépes nyelvészeti konferencia (MSZNY 2003), Szeged, Hungary, 2003.

[35] G. Kovásznai, „*Párbeszédes rendszerek*" (in Hungarian). Magyar Tudományos Akadémia, Távközlési Rendszerek Bizottsága, Budapest, Hungary, 2003.

[36] G. Kovásznai, „*Algorithmic Improvements in Natural Language Parsing within Dialogue Systems: Priority Patterns and Wildcards*". ICAI 2004, Eger, Hungary, 2004.

[37] G. Kovásznai, „*Unification for Effective and Finite Semantic Tableaux in First-order Logic: the SOFIA Prover*". CSCS 2004, Szeged, Hungary, 2004.

[38] G. Kovásznai, „*Redundancy for Rigid Clausal Tableaux*". ICAI 2007, Eger, Hungary, 2007.