

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Logic E-Learning Assistant

propusă de

Ștefan-Claudiu Susan

Sesiunea: iulie, 2020

Coordonator științific

Conf. Dr. Andrei Arusoaie

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

Logic E-Learning Assistant

Ștefan-Claudiu Susan

Sesiunea: iulie, 2020

Coordonator științific

Conf. Dr. Andrei Arusoaie

Cuprins

Motivație	2
Introducere	3
1 Prezentare generală	4
1.1 Context	4
1.2 Contribuții	4
1.3 Arhitectura aplicației	6
2 Descrierea problemei	8
2.1 Prezentare generală	8
2.2 Soluții deja existente	9
2.2.1 FOL Evaluator	9
2.2.2 Natural Deduction	10
2.2.3 WolframAlpha	11
2.2.4 Tree Proof Generator	12
2.3 Neajunsuri ale soluțiilor existente	15
3 Prezentarea contribuțiilor	17
3.1 Prezentare generală	17
3.2 Parsarea unei formule	18
3.2.1 Aspecte teoretice	18
3.2.2 Verificarea sintactică	20
3.2.3 Construirea arborelui	21
3.3 Analiza Formulei	22
3.4 Deducție Naturală	25
3.4.1 Aspecte teoretice	26
3.4.2 Implementare	27

3.5	Forme Normale	28
3.5.1	Logica propozițională	28
3.5.2	Logica de ordinul I	32
3.6	Rezoluție	35
3.6.1	Aspecte teoretice	35
3.6.2	Implementare	36
3.7	Quiz	38
4	Evaluarea soluției	42
4.1	Logica Propozițională	42
4.1.1	Analiza Formulei	42
4.1.2	Deducție Naturală	44
4.1.3	Forme normale	45
4.1.4	Rezoluție	46
4.2	Logica de ordinul I	48
4.2.1	Analiza Formulei	48
4.2.2	Deducție Naturală	50
4.3	Forme Normale	52
4.4	Rezoluția de ordinul I	54
4.5	Observații pe baza rezultatelor	54
	Concluzii	56
	Bibliografie	57
	Appendix	60

Motivație

Motivația aplicației provine din numărul mic de tool-uri dedicate logicii de ordinul I și logicii propoziționale ce pot fi folosite de către începători sau al căror scop este cel de a ajuta în învățare. Nu există astfel de aplicații care să ofere un pachet complet și să fie destul de ușor de folosit pentru a facilita învățarea.

Pe lângă numărul mic, acestea prezintă și numeroase inconveniente care fac grea și neproductivă utilizarea acestora de către persoane fără experiență. Printre acestea se numără dificultatea utilizării, acest aspect făcându-le nepotrivite pentru cei care au primul lor contact cu logica. Un alt inconvenient este unul de natură teoretică, unele dintre acestea prezintă funcționalități similare (rezoluție, deducție, etc.), însă abordarea acestora nu este suficient de explicită, pașii urmați nu sunt scoși în evidență. De asemenea, componenta de verificare a demonstrațiilor scrise de utilizator este absentă. În plus, sintaxa utilizată de acestea este mai apropiată de cea a unui limbaj de programare și nu de limbajul utilizat în cadrul acestui domeniu (ex: semne diferite pentru operatori), iar acest lucru poate reprezenta un inconvenient pentru cei neexperimentați. Așadar, aplicațiile deja prezente ce vizează acest domeniu sunt incomplete sau nepotrivite pentru nevoile începătorilor.

Logic E-Learning Assistant propune o abordare orientată către partea de învățare. Oferă un pachet de funcționalități complet, făcând posibilă atât redactarea unor demonstrații în cadrul aplicației, cât și generarea automată a acestora pentru unele capitole, dar și verificarea unor demonstrații redactate în format text.

Introducere

Logica propozițională este logica propozițiilor, legate între ele prin conectori logici cum ar fi *sau*, *și* și *non*. O propoziție este o afirmație care este adevărată sau falsă [6].

Logica de ordinul I este o extensie a logicii propoziționale, extensie care aduce un plus de expresivitate. Expresivitatea adițională este necesară pentru a putea modela anumite afirmații care nu pot fi exprimate în logica propozițională. Logica de ordinul I aduce, în plus față de LP, noțiunea de cuantificator (existențial sau universal) și noțiunea de predicat. Un predicat este o afirmație a cărei valoare de adevăr depinde de zero sau mai mulți parametrii [13].

Capitolul 1

Prezentare generală

1.1 Context

Logica propozițională este cel mai simplu și cel mai utilizat tip de logică, acest lucru derivă din capacitatea acesteia de a exprima în mod eficient funcții booleene alături de metode de verificare și producere a demonstrațiilor.

Gradul de utilizare al acestui tip de logică a crescut dramatic la finalul secolului trecut odată cu implementarea unor algoritmi puternici de căutare. Aceasta este folosită pentru a reprezenta multe probleme NP-complete, probleme des întâlnite în **Inteligența Artificială**, cum ar fi decision-making, problem-solving și planning.

Extensiile acesteia, în ciuda expresivității sporite, nu prezintă aceiași algoritmi eficienți și scalabili. Existența acestor algoritmi și eficiența lor reprezintă motivul principal al utilizării intense de care se bucură logica propozițională[16].

Printre aplicațiile **Logicii de ordinul I** trebuie să amintim noțiunea de model-checking. Acesta reprezintă o metodă pentru a stabili dacă un model îndeplinește anumite specificații. Este generată o formulă ce aparține logicii de ordinul I și nu conține variabile libere, apoi se stabilește dacă o implementare finită poate constitui un model pentru formula respectivă.¹

1.2 Contribuții

Logic E-Learning Assistant are ca scop principal ajutarea studenților în învățarea logicii, oferă funcționalități pentru noțiuni elementare (înălțimea arborelui, subfor-

¹https://en.wikipedia.org/wiki/Model_checking

mule), dar și pentru capitole mai complexe (deducție naturală, forme normale). Folosește un limbaj apropiat de cel utilizat în cadrul logicii, iar conținutul aplicației este modelat în conformitate cu materia parcursă în cadrul facultății, aceasta fiind construită pentru a fi utilizată odată cu studierea capitolelor respective.

Dintre facilitățile oferite amintim următoarele :

- Analiza formulei

1. Verificarea sintactică a unei formule logice
2. Aflarea subformulelor și generarea explicației pentru calculul acestora
3. Aflarea variabilelor propoziționale și generarea explicației pentru calculul acestora
4. Transformarea formulei prin înlocuirea implicațiilor simple și duble
5. Generarea și desenarea arborelui
6. Dimensiunea arborelui
7. Înălțimea arborelui
8. Evaluarea unei formule (doar pentru formulele din logica de ordinul I)
9. Verificarea satisfiabilității
10. Verificare tautologie
11. Verificare contradicție
12. Generare tabel de adevăr (doar pentru formulele din logica propozițională)
13. Găsirea complementului (doar pentru formulele din logica propozițională)
14. Găsirea de substitutii (doar pentru formulele din logica de ordinul I)

- Deducție naturală

1. Scrierea și verificarea unei demonstrații pas cu pas
2. Verificarea unei demonstrații aflate în format text
3. Exportarea demonstrației în sistemul local de fișiere în format text

- Forme Normale

1. Scrierea și verificarea unei transformări pas cu pas
2. Generarea unei transformări cu pașii acesteia explicați detaliat

- Rezoluție
 1. Scrierea și verificarea unei demonstrații pas cu pas
 2. Generarea unei demonstrații
 3. Verificarea unei demonstrații aflate în format text
 4. Exportarea demonstrației în sistemul local de fișiere în format text
- Testarea cunoștințelor dobândite într-un format bazat pe nivele prin multiple exerciții din fiecare capitol

1.3 Arhitectura aplicației

Aplicația prezintă o arhitectură monolită (Figura 1.1) și utilizează unele tool-uri (JavaCC[5] pentru generarea de parsere) sau alte resurse exterioare (fișierul Python). Limbajul de programare utilizat în implementare este în mare parte Java, alături de Python pentru fișierul ce realizează evaluarea formulelor de ordinul I. De asemenea, folosește o bază de date pentru a reține informații necesare părții de testare a cunoștințelor acumulate de către utilizator (nivelul curent, subiectele).

Componentele mari ale aplicației de față sunt următoarele :

1. Clasele ce conțin logica aplicației
2. Interfața grafică
3. Generatorul de parsere
4. Fișierul Python utilizat la evaluarea formulelor din logica de ordinul I
5. Baza de date ce reține informații relevante pentru componenta de teste

Dintre pachetele aplicației Java (Figura 1.1) sunt evidențiate următoarele:

- application : Conține subpachete în care se găsesc fișiere FXML ce determină layout-urile ferestrelor din interfață și controller-ele acestora alături de diverse resurse (imagini, fișierul CSS etc.)
- Parsers : Clasele generate de către JavaCC pentru parsarea și verificarea sintactică a formulelor

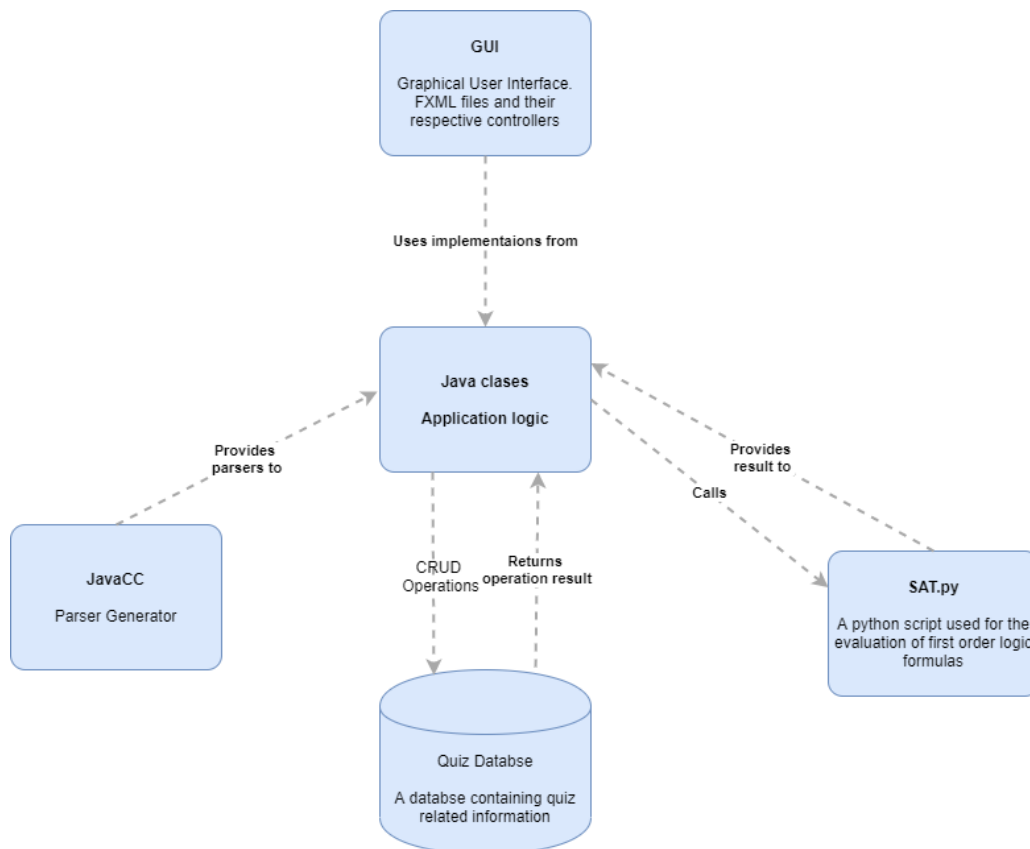


Figura 1.1: Diagrama aplicației

- **AbstractSyntaxTree** : Implementări pentru arborele abstract de sintaxă și un nod din acesta alături de alte metode ce le analizează
- **Formulas** : Clase ce modelează formulele
- **NaturalDeduction** : Conține funcționalitățile ce țin de partea de deducție naturală
- **NormalForms** : Conține funcționalitățile ce țin de partea de forme normale
- **Resolution** : Conține funcționalitățile ce țin de partea de rezoluție
- **DbManagement** : Facilitează comunicarea cu baza de date pentru Quiz.

Capitolul 2

Descrierea problemei

Logica propozițională și extensiile acesteia sunt subiecte ce necesită multă practică pentru a fi stăpânite și includ numeroase capitole ce necesită diverse noțiuni fundamentale sau unele capitole precedente. Astfel, studiul logicii trebuie să fie un proces incremental unde teoria se îmbină cu practica în cadrul fiecărui capitol. De exemplu, **rezoluția** nu poate fi studiată fără a avea cunoștințe cu privire la **forme normale**.

La fel ca matematica, logica necesită foarte mult exercițiu pentru a fi înțeleasă pe deplin. Acest exercițiu trebuie practicat și în afara timpului alocat studiului teoretic. Însă, nu au la dispoziție aplicații care să faciliteze lucrul individual, un tool care să îi ajute pe utilizatori să pună în practică noțiunile învățate și să vină drept un complement noțiunilor teoretice învățate. În absența unei astfel de unelte studiul individual poate prezenta o provocare.

2.1 Prezentare generală

În cazul de față problema este reprezentată de către dificultățile pe care începătorii le pot avea în studiul individual al logicii. Cauza problemei este absența tool-urilor care oferă toate funcționalitățile necesare, sunt ușor de folosit și sunt în conformitate cu materia studiată. Un echilibru între acestea 3 este foarte greu de găsit. Majoritatea variantelor existente nu sunt concepute pentru a fi utilizate de către începători sau pot fi utilizate doar în anumite cazuri sau curpind doar o parte din capitolele care trebuie studiate.

2.2 Soluții deja existente

Pentru început, trebuie să precizăm tool-uri de tipul Proof Assistant cum ar fi Isabelle [14] sau Coq [1]. Acestea oferă toate funcționalitățile necesare pentru a construi un sistem deductiv sau pentru alte operații. Însă, absența unui pachet complet de facilități deja implementate disponibile într-un limbaj uniform poate reprezenta o problemă.

Pe pagina cursului Facultății de Informatică (UAIC) ¹ se găsesc aplicații suport pentru unele operații de baza aplicate formulelor din logica propozițională: verificarea sintaxei, subformule, arborele abstract, înălțimea arborelui, dimensiunea arborelui și mulțimea de variabile propoziționale. Acestea sunt perfecte pentru a ajuta în învățarea noțiunilor fundamentale ce țin de sintaxa și semantica formulelor din logica propozițională.

Pentru a determina satisfiabilitatea unei formule din logica propozițională există numeroase implementări, acestea conțin în mod implicit și componenta de evaluare a formulei.

2.2.1 FOL Evaluator

Pentru evaluare formulelor din logica de ordinul I putem lua în considerare **FOL Evaluator**². Acesta oferă o evaluare a formulelor (Figura 2.1) puternic ancorată într-un model matematic cu unele limitări.

- Puncte pozitive :

1. Evaluarea detaliată, prezentarea fiecărei posibilități pentru variabilele cuantificate.
2. Scoaterea în evidență a zonei unde trebuie definit modelul

- Puncte negative :

1. Domeniul poate fi alcătuit doar din numere întregi
2. Valorile variabilelor și constantelor pot fi doar numere întregi
3. Funcțiile și predicatele pot fi doar operații matematice de bază
4. Sintaxa utilizată pentru funcții și predicate este diferită de cea clasică

¹<https://profs.info.uaic.ro/logica/logica-2019-2020>

²<https://mrieppel.github.io/fol>

```

AxExy=x=y is true on this model

Evaluation History:
-----

AxExy=x=y is true on {}
  Eyx=y is true on {"x":1}
    x=y is true on {"x":1,"y":1}
    x=y is false on {"x":1,"y":2}
    x=y is false on {"x":1,"y":3}
  Eyx=y is true on {"x":2}
    x=y is false on {"x":2,"y":1}
    x=y is true on {"x":2,"y":2}
    x=y is false on {"x":2,"y":3}
  Eyx=y is true on {"x":3}
    x=y is false on {"x":3,"y":1}
    x=y is false on {"x":3,"y":2}
    x=y is true on {"x":3,"y":3}

```

Figura 2.1: Evaluare formulei $\forall x \exists y x=y$ pentru domeniul $\{1,2,3\}$

2.2.2 Natural Deduction

Natural Deduction³ este un tool utilizat pentru deducție naturală în cadrul logicii propoziționale. Acesta generează automat demonstrații (Figura 2.2) pentru cazuri elementare. Demonstrația obținută este fluidă și clară. În ciuda lipsurilor este o variantă bună pentru exercițiile simple.

- Puncte pozitive :

1. Demonstrația este generată automat.
2. Ușor de utilizat.

- Puncte negative :

1. Oferă suport doar pentru 9 dintre regulile de inferență prezentate în cadrul cursului.
2. Sintaxa formulelor este puternic diferită de cea utilizată în domeniu
3. Aplicarea regulilor nu este clar semnalată

O variantă demnă de luat în considerare pentru deducție naturală în cadrul logicii de ordinul I este **Natural deduction proof editor and checker**⁴. Este complet, cu o interfață plăcută și posibilitatea de a rezolva unele exerciții puse la dispoziție. Cu

³<http://teachinglogic.liglab.fr/DN/index.php>

⁴<https://proofs.openlogicproject.org/>

```

-p & -q => -p

```

```

assume -p & -q.
  assume p + q.
    assume p.
      -p.
      F.
    therefore p => F.
  assume q.
    -q.
    F.
  therefore q => F.
  F.
therefore -(p + q).
therefore -p & -q => -(p + q).

```

Figura 2.2: Demonstrație generată pentru $\{\neg p \wedge \neg q\} \vdash \neg(p \vee q)$

toate acestea, este destul de greu de utilizat și nu oferă o sintaxă clară pentru funcții și predicate și nu oferă posibilitatea de a verifica demonstrații în format text.

2.2.3 WolframAlpha

WolframAlpha[3] este un motor de căutare factual ce prezintă suport pentru diverse subiecte (matematică, istorie, științe, etc.). În cadrul secțiunii "Boolean Algebras" se găsesc diverse secțiuni ce oferă informații (Figura 2.3) clare despre o formulă dată. Este un tool foarte bine realizat și cu o mare varietate de funcționalități din mai multe domenii. Atipic față de alte motoare de căutare, acesta nu oferă link-uri către răspunsuri, ci direct răspunsuri. De asemenea, oferă diverse informații adiționale. Dispune de o bază de date de dimensiuni foarte mari cu informații din multiple domenii și viteza este garantată de hardware-ul puternic pe care rulează serverul.⁵

Pentru o formulă dată ce aparține logicii propoziționale poate furniza următoarele informații (printre altele):

1. Diverse forme normale
2. Tabelul de adevăr

⁵<https://en.wikipedia.org/wiki/WolframAlpha>

3. Circuitul logic

4. Diagrama Venn

- Puncte pozitive :

1. Informații foarte variate
2. Ușor de utilizat.
3. Interfața grafică plăcută

- Puncte negative :

1. În ciuda varietății lor, informațiile nu sunt detaliate
2. Procesul de utilizare nu este deloc interactiv, utilizatorul nu dispune de opțiunea de a realiza singur transformările

2.2.4 Tree Proof Generator

Tree Proof Generator⁶ este o aplicație ce încearcă să demonstreze faptul că o formulă din logica propozițională (Figura 2.4) sau logica de ordinul I (Figura 2.5) dată ca input este tautologie. În cazul în care nu poate demonstra că formula este validă atunci va furniza un contramodel (Figura 2.6). Pentru demonstrație folosește o combinație între transformări în forme normale și rezoluție. De asemenea, acesta va furniza și un arbore corespunzător demonstrației. Arborele furnizat poate fi salvat ulterior în format PNG.

- Puncte pozitive :

1. Funcționează pe mai multe tipuri de logică
2. Oferă un contraexemplu dacă nu poate dovedi că formula este validă
3. Demonstrațiile sunt generate automat
4. Ușor de folosit
5. Arborele rezultat poate fi salvat în format PNG

⁶<https://www.umsu.de/trees/>

Minimal forms:

DNF	$(P \wedge \neg Q \wedge \neg r) \vee q$
CNF	$(P \vee q) \wedge (q \vee \neg Q) \wedge (q \vee \neg r)$
ANF	$(P \wedge q) \vee (P \wedge Q) \vee (P \wedge r) \vee (P \wedge q \wedge Q) \vee (P \wedge q \wedge r) \vee (P \wedge Q \wedge r) \vee (P \wedge q \wedge Q \wedge r) \vee P \vee q$
NOR	$(P \bar{q}) \bar{q} (q \bar{q} \neg Q) \bar{q} (q \bar{q} \neg r)$
NAND	$(P \bar{\bar{r}} \neg Q \bar{\bar{r}} \neg r) \bar{\bar{r}} \neg q$
AND	$\neg(\neg P \wedge \neg q) \wedge \neg(\neg q \wedge Q) \wedge \neg(\neg q \wedge r)$
OR	$\neg(\neg P \vee Q \vee r) \vee q$

(assuming NAND and NOR are n-ary operators)

$$e_1 \vee e$$

$$e_1 \bar{q} e$$

$$e_1 \bar{\bar{r}} e_2$$

Other forms:

ESOP	$(P \wedge \neg q \wedge \neg Q \wedge \neg r) \vee q$
IMPLIES	$(P \Rightarrow \neg((Q \Rightarrow \neg q) \Rightarrow \neg(\neg Q \Rightarrow \neg(r \Rightarrow q)))) \Rightarrow \neg(\neg P \Rightarrow \neg q)$
ITE	$(P \wedge ((Q \wedge q) \vee (\neg Q \wedge ((r \wedge q) \vee \neg r)))) \vee (\neg P \wedge q)$

$p \Rightarrow q$ represents

Logic circuit:

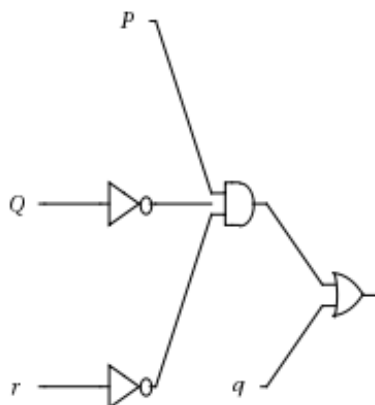


Figura 2.3: O parte din informațiile oferite pentru formula $(\neg p \wedge \neg Q \wedge \neg r) \vee q$

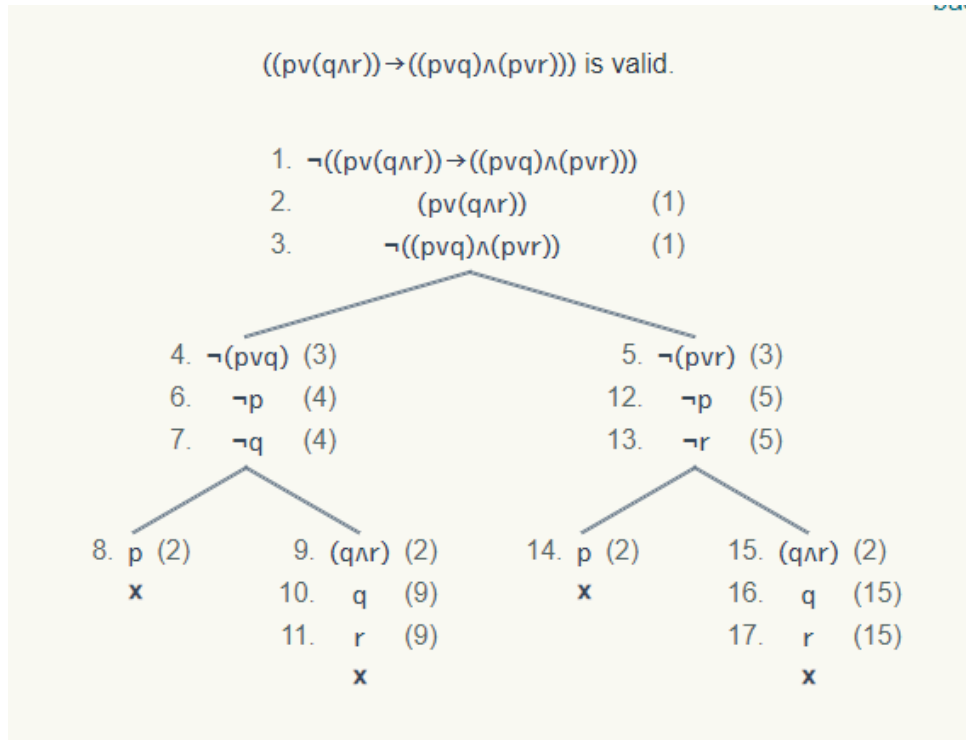


Figura 2.4: Arbore generat pentru formula $(p \vee (q \wedge r)) \rightarrow ((p \vee q) \wedge (p \vee r))$

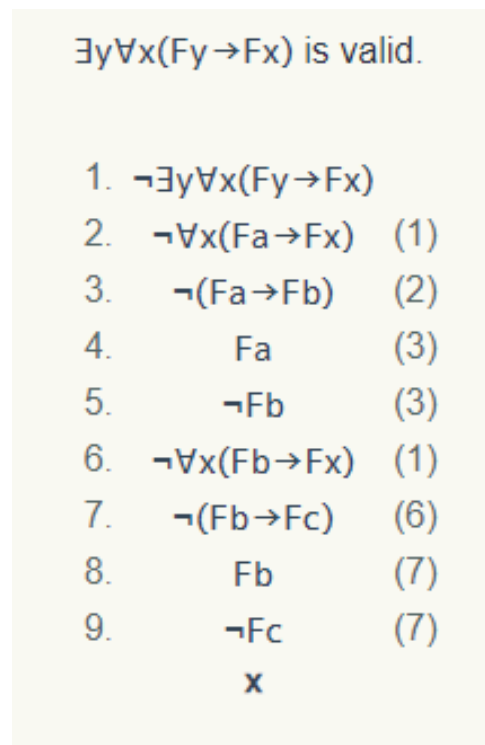


Figura 2.5: Arbore generat pentru formula $\exists y \forall x (F(y) \rightarrow F(x))$

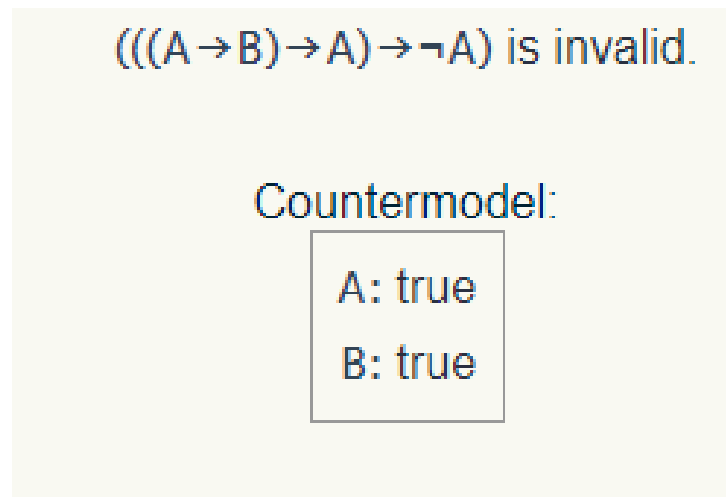


Figura 2.6: Contramodel generat pentru formula $((A \rightarrow B) \rightarrow A) \rightarrow \neg A$

- Puncte negative :
 1. Demonstrația nu este explicată
 2. Utilizatorul nu poate scrie demonstrații sau să editeze unele existente
 3. Nu prezintă o funcție de verificare a demonstrațiilor furnizate de către utilizator
 4. Modalitatea de demonstrare diferă în unele aspecte față de cea studiată în cadrul cursului

2.3 Neajunsuri ale soluțiilor existente

După cum am putut observa, majoritatea tool-urilor disponibile au și anumite inconveniențe care le fac incompatibile cu procesul complet de învățare. Nu se găsesc tool-uri care să fie ușor de folosit și să acopere toate capitolele studiate. Există aplicații dedicate pentru deducție naturală sau pentru analiza formulei de exemplu, însă nu conțin suport pentru alte aspecte ale logicii.

Printre neajunsurile soluțiilor deja existente se numără :

- lipsa de interacțiune din utilizarea acestora, majoritatea sunt capabile să producă un rezultat valid, însă utilizatorul nu poate participa în vreun fel la generarea demonstrației. Pe lângă asta, demonstrațiile produse sunt explicate sumar sau deloc. Aceste două trăsături le fac neproductive pentru procesul de învățare.

- imposibilitatea de a verifica demonstrații scrise de către utilizator. Lipsa acestei facilități afectează utilitatea aplicației din perspectiva unui începător.
- majoritatea variantelor disponibile nu prezintă un set de exerciții de verificare a cunostințelor dobândite.
- aceste aplicații diferă de materia studiată în cardul cursului, fie prin alfabetul sau sintaxa formulelor, fie prin unele incompatibilități teoretice.
- există totuși, unele variante ce oferă majoritate funcționalităților necesare. Cu toate acestea, sunt greu de utilizat, nu sunt destinate celor care au primul contact cu logica.

Capitolul 3

Prezentarea contribuțiilor

Logic E-Learning Assistant are ca scop principal ajutarea utilizatorilor în învățarea logicii, oferă funcționalități pentru noțiuni elementare (înălțimea arborelui, subformule), dar și pentru capitole mai complexe (deducție naturală, forme normale). Folosește un limbaj apropiat de cel utilizat în cadrul logicii, iar conținutul aplicației este modelat pentru a fi folosit incremental pe măsură ce utilizatorul acumulează noi cunoștințe.

3.1 Prezentare generală

Dintre facilitățile oferite amintim următoarele :

- Analiza formulei
 1. Verificarea sintactică a unei formule logice
 2. Aflarea subformulelor și generarea explicației pentru calculul acestora
 3. Aflarea variabilelor propoziționale și generarea explicației pentru calculul acestora
 4. Transformarea formulei prin înlocuirea implicațiilor simple și duble
 5. Generarea și desenarea arborelui
 6. Dimensiunea arborelui
 7. Înălțimea arborelui
 8. Evaluarea unei formule (doar pentru formulele din logica de ordinul I)
 9. Verificare satisfiabilitate
 10. Verificare tautologie

11. Verificare contradicție
 12. Generare tabel de adevăr (doar pentru formulele din logica propozițională)
 13. Găsirea complementului (doar pentru formulele din logica propozițională)
 14. Găsirea de substituții (doar pentru formulele din logica de ordinul I)
- Deducție naturală
 1. Scrierea și verificarea unei demonstrații pas cu pas
 2. Verificarea unei demonstrații aflate în format text
 3. Exportarea demonstrației în sistemul local de fișiere în format text
 - Forme Normale
 1. Scrierea și verificarea unei transformări pas cu pas
 2. Generarea unei transformări cu pașii acesteia explicați detaliat
 - Rezoluție
 1. Scrierea și verificarea unei demonstrații pas cu pas
 2. Generarea unei demonstrații
 3. Verificarea unei demonstrații aflate în format text
 4. Exportarea demonstrației în sistemul local de fișiere în format text
 - Testarea cunoștințelor dobândite într-un format bazat pe nivele prin multiple exerciții din fiecare capitol

3.2 Parsarea unei formule

Inițial, formula este primită ca String, dacă aceasta trece de verificarea sintactică va fi apelat algoritmul de parsare, în caz contrar, va fi aruncată o excepție.

3.2.1 Aspecte teoretice

Logica propozițională

Alfabetul logicii propoziționale este reuniunea următoarelor simboluri:

1. $A = \{p, q, r, p', \dots\}$ este o mulțime infinit numărabilă de variabile propoziționale pe care o fixăm la început
2. $\{\neg, \wedge, \vee\}$ este mulțimea conectorilor logici
3. $\{(,)\}$ este mulțimea simbolurilor auxiliare

Mulțimea formulelor propoziționale (LP) este cea mai mică mulțime de cuvinte care satisface următoarele proprietăți [11] :

1. Orice variabilă propozițională este în LP
2. Dacă $\varphi \in LP$, atunci $\neg\varphi \in LP$
3. Dacă $\varphi_1, \varphi_2 \in LP$, atunci $(\varphi_1 \vee \varphi_2) \in LP$
4. Dacă $\varphi_1, \varphi_2 \in LP$, atunci $(\varphi_1 \wedge \varphi_2) \in LP$

Logica de ordinul I

Alfabetul logicii de ordinul 1 este reuniunea următoarelor simboluri:

1. conectori logici : $\wedge \vee \neg \rightarrow \leftrightarrow$ la care se adaugă cunificatorii \forall și \exists
2. $X = \{x, y, z, x', y', z_1, \dots\}$ este o mulțime infinit numărabilă de variabile diferită de cea din cadrul logicii propoziționale
3. $"(", ")", ":", "$ și $"', "$ este mulțimea simbolurilor auxiliare
4. simboluri adiționale (predicative și funcționale în funcție de semnătură)

Un termen este fie o variabilă, fie o constantă (simbol funcțional de aritate 0), fie un șir de caractere de forma $F_1(t_1, t_2, \dots, t_n)$, unde F_1 este un simbol funcțional și t_1, t_2, \dots, t_n sunt termeni.

O formulă atomică este un șir de caractere de forma $P_1(t_1, t_2, \dots, t_n)$ unde P_1 este un simbol predicativ și t_1, t_2, \dots, t_n sunt termeni.

Mulțimea formulelor de ordinul I, notată $LP1$, este cea mai mică mulțime astfel încât :

1. Orice formulă atomică este formulă din $LP1$
2. Pentru orice formule $\varphi, \varphi_1, \varphi_2 \in LP1$ și pentru orice variabilă $x \in X$ avem că [13]:
 - $\neg\varphi \in LP1$
 - $(\varphi_1 \vee \varphi_2) \in LP1$
 - $(\varphi_1 \wedge \varphi_2) \in LP1$
 - $(\varphi_1 \rightarrow \varphi_2) \in LP1$
 - $(\varphi_1 \leftrightarrow \varphi_2) \in LP1$
 - $\forall x.\varphi \in LP1$
 - $\exists x.\varphi \in LP1$

3.2.2 Verificarea sintactică

Verificarea sintactică se realizează cu ajutorul parser-elor generate de către JavaCC. Acesta este un tool asemănător cu YACC, construiește parsere pe baza unei gramatici de tip $LL(1)$. În cadrul aplicației sunt definite două gramatici, una pentru logica propozițională și una pentru logica de ordinul I, acestea sunt asemănătoare cu o gramatică pentru operații matematice.

Atunci când este furnizat un string se încearcă parsarea acestuia, în cazul unei parsări cu succes este furnizat mesajul "Ok", răspuns ce denotă o sintaxă validă, iar în cazul în care parsarea eșuează este furnizat un mesaj de eroare ce semnalează clar greșeala sintactică întâlnită.

Funcția de parsare corespunzătoare este apelată în constructorul claselor Formula și FOLFormula, în cazul în care mesajul nu este unul pozitiv va fi aruncată o excepție. Acest mecanism garantează validitatea sintactică a formulelor ce sunt reprezentate cu ajutorul uneia dintre cele două clase.

3.2.3 Construirea arborelui

Odată realizată verificarea sintactică, este realizată conversia șirului de caractere ce reprezintă formula în arborele corespunzător. Algoritmul utilizat este bazat pe Dijkstra's shunting yard algorithm¹(Algoritm 1). În continuare, vom folosi o stivă de noduri pentru a obține rădăcina arborelui (Algoritm 2).

Acesta este utilizat pentru parsarea expresiilor matematice aflate în notație infixată și a fost adaptat pentru formule logice.

Algorithm 1: Shunting-yard algorithm

Result: The postfix notation

while *there are tokens to be read* **do**

 read a token;

if *the token is a variable* **then**

 push it to the output queue;

else if *the token is an operator* **then**

while *there is an operator at the top of the operator stack with greater or equal precedence* **do**

 pop operators from the operator stack onto the output queue;

end

 push it onto the operator stack;

else if *the token is a left parenthesis* **then**

 push it onto the operator stack;

else if *the token is a right parenthesis* **then**

while *the operator at the top of the operator stack is not a left parenthesis* **do**

 pop the operator from the operator stack onto the output queue

end

if *there is a left parenthesis at the top of the operator stack* **then**

 pop the operator from the operator stack and discard it

end

if *there are no more tokens to read* **then**

while *there are still operator tokens on the stack* **do**

 pop the operator from the operator stack onto the output queue

end

¹https://en.wikipedia.org/wiki/Shunting-yard_algorithm

Algorithm 2: Building the syntax tree from the postfix notation

Result: The corresponding syntax tree

```
while there are token to read do
    read a token;
    if the token is a variable then
        | add it to the node stack;
    else if the token is an unary operator then
        | pop a node from the node stack;
        | add the operation result to node the stack;
    else if the token is a binary operator then
        | pop two nodes from the node stack;
        | add the operation result to the node stack;
end

return the node at the top of the stack;
```

Pentru formulele de ordin I abordarea este similară, însă este făcută o nouă parcurgere a șirului de caractere înainte de rularea algoritmului pentru a izola grupurile de caractere în funcție de semnificația lor (predicate, cuantificatori împreună cu variabila cuantificată). Algoritmul va primi o listă de string-uri ce reprezintă grupuri de caractere sau operatori în ordinea apariției lor în șirul de caractere inițial și variabilele propoziționale vor fi înlocuite de predicate, subarborele pentru acestea este construit separat atunci când sunt adăugate în stiva de noduri în etapa a doua.

3.3 Analiza Formulei

O parte din operațiile ce țin de analiza formulei (dimensiunea arborelului, înălțimea arborelului, etc.) sunt realizate cu ajutorul metodelor din clasa ce reprezintă arborele acesteia (Figura 3.1). Metodele respective realizează parcurgeri recursive ale arborelui și îl modifică sau adună informații în funcție de caz. Implementările acestora sunt adaptări ale funcțiilor definite pentru calcularea acestora la nivel teoretic [11]. De asemenea, explicațiile sunt generate (acolo unde este cazul) cu ajutorul unei liste de String-uri dată ca argument metodei respective, în această listă se adaugă treptat clarificări pe măsură ce este calculat rezultatul. Înlocuirea implicațiilor se realizează tot prin parcurgerea arborelui și înlocuirea nodurilor ce conțin implicații sau duble implicații.

Calculul înălțimii (Figura 3.2), al dimensiunii (Figura 3.3), al subformulelor (Figura 3.4), al variabilelor (Figura 3.5) și înlocuirea implicațiilor pentru formulele din LP este similar cu cel pentru formulele din LP1.

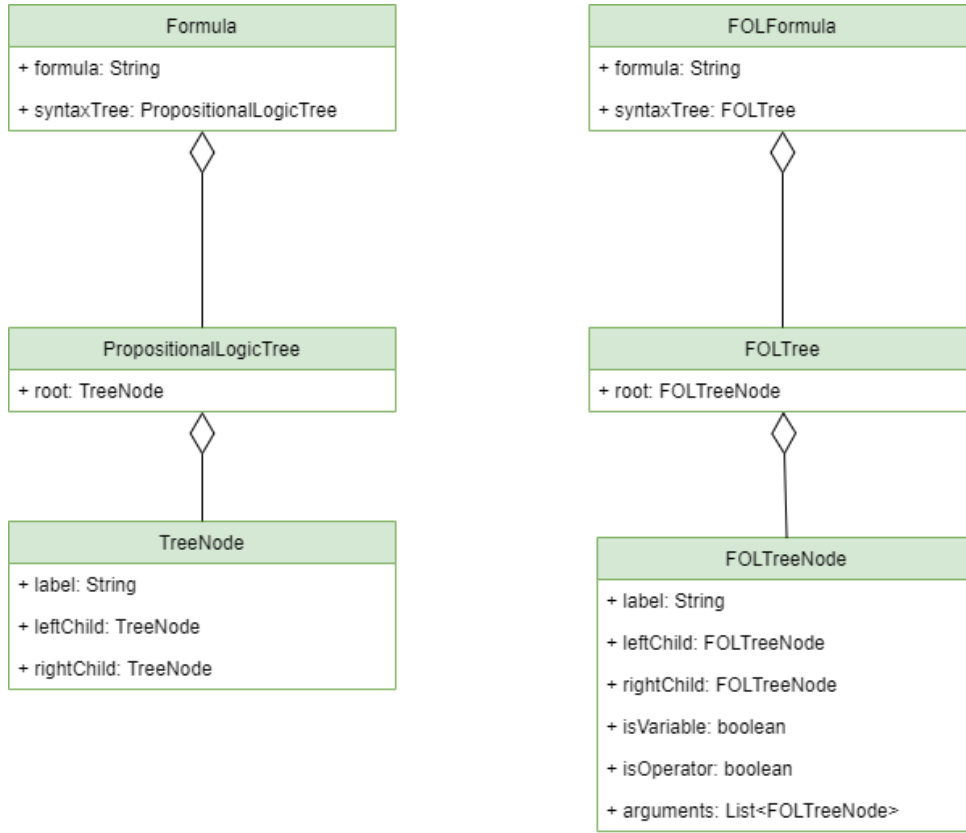


Figura 3.1: Diagrama claselor utilizate pentru reprezentarea unei formule

$$height(\varphi) = \begin{cases} 1, & \text{dacă } \varphi \in A; \\ 1 + height(\varphi'), & \text{dacă } \varphi = \neg\varphi' \text{ și } \varphi' \in LP; \\ 1 + \max(height(\varphi_1), height(\varphi_2)), & \text{dacă } \varphi = (\varphi_1 \wedge \varphi_2) \text{ și } \varphi_1, \varphi_2 \in LP; \\ 1 + \max(height(\varphi_1), height(\varphi_2)), & \text{dacă } \varphi = (\varphi_1 \vee \varphi_2) \text{ și } \varphi_1, \varphi_2 \in LP. \end{cases}$$

Figura 3.2: Modalitate de calcul a înălțimii arborelui unei formule din LP

Evaluarea formulelor din LP se realizează prin analiza recursivă a nodurilor din arbore (ex: pentru un nod etichetat cu " \wedge " se va returna conjuncția rezultatelor evaluărilor nodurilor fii). Pentru determinarea satisfiabilității și validității sunt utilizați algoritmi de tipul backtracking² și este utilizată optimizarea operațiilor booleane pentru a face prunarea rezultatelor care nu pot duce la soluții valide (ex: Dacă dorim să evaluăm $\varphi_1 \wedge \varphi_2$ și φ_1 este evaluată la 0 atunci φ_2 nu va mai fi evaluată deoarece rezultatul conjuncției va fi în mod sigur 0).

²<https://en.wikipedia.org/wiki/Backtracking>

$$size(\varphi) = \begin{cases} 1, & \text{dacă } \varphi \in A; \\ 1 + size(\varphi'), & \text{dacă } \varphi = \neg\varphi' \text{ și } \varphi' \in LP; \\ 1 + size(\varphi_1) + size(\varphi_2), & \text{dacă } \varphi = (\varphi_1 \wedge \varphi_2) \text{ și } \varphi_1, \varphi_2 \in LP; \\ 1 + size(\varphi_1) + size(\varphi_2), & \text{dacă } \varphi = (\varphi_1 \vee \varphi_2) \text{ și } \varphi_1, \varphi_2 \in LP. \end{cases}$$

Figura 3.3: Modalitate de calcul a dimensiunii arborelui unei formule din LP

$$subf: LP \rightarrow 2^{LP}, \text{ definită prin:}$$

$$subf(\varphi) = \begin{cases} \{\varphi\}, & \text{dacă } \varphi \in A; \\ \{\varphi\} \cup subf(\varphi'), & \text{dacă } \varphi = \neg\varphi' \text{ și } \varphi' \in LP; \\ \{\varphi\} \cup subf(\varphi_1) \cup subf(\varphi_2), & \text{dacă } \varphi = (\varphi_1 \wedge \varphi_2) \text{ și } \varphi_1, \varphi_2 \in LP; \\ \{\varphi\} \cup subf(\varphi_1) \cup subf(\varphi_2), & \text{dacă } \varphi = (\varphi_1 \vee \varphi_2) \text{ și } \varphi_1, \varphi_2 \in LP. \end{cases}$$

Figura 3.4: Modalitate de calcul a subformulelor unei formule din LP

Pentru determinarea complementului [2] (Figura 3.6) unei formule din LP este utilizată o metoda dintr-o clasă din pachetul **NormalForms** ce face înlocuirile necesare parcurgând arborele formulei.

Evaluarea formulelor din LP1 se face cu ajutorul fișierului Python ce se găsește în folderul cu resurse al aplicației și este apelat cu de către clasa **Runtime**, rezultatul este apoi preluat din output-ul acestuia. Primește următoarele argumente :

- Arborele formulei reprezentat în format JSON
- Calea fișierului care conține asignarea
- Calea fișierului care conține implementările funcțiilor și predicatelor
- Calea fișierului care conține domeniile variabilelor legate

În cadrul codului Python, fișierul JSON ce conține arborele formulei este memorat într-un dicționar, apoi variabilele din fișierul ce conține asignarea sunt declarate și inițializate dinamic cu valorile corespunzătoare, pentru variabilele legate sunt inițializate dinamic colecțiile ce reprezintă domeniile acestora, iar implementările funcțiilor și predicatelor sunt prelucrate pentru a fi transformate în funcții Python și sunt executate dinamic. Acestea din urmă sunt generate cu ajutorul funcției builtin **exec**. După ce sunt inițializate aceste elemente este realizată o parcurgere a arborelui similară cu cea din cadrul logicii propoziționale, iar predicatele sunt evaluate folosind funcția builtin **eval**.

$$prop(\varphi) = \begin{cases} \{\varphi\}, & \text{dacă } \varphi \in A; \\ prop(\varphi'), & \text{dacă } \varphi = \neg\varphi' \text{ și } \varphi' \in LP; \\ prop(\varphi_1) \cup prop(\varphi_2), & \text{dacă } \varphi = (\varphi_1 \wedge \varphi_2) \text{ și } \varphi_1, \varphi_2 \in LP; \\ prop(\varphi_1) \cup prop(\varphi_2), & \text{dacă } \varphi = (\varphi_1 \vee \varphi_2) \text{ și } \varphi_1, \varphi_2 \in LP. \end{cases}$$

Figura 3.5: Modalitate de calcul a variabilelor propoziționale ale unei formule din LP

1. $a^c = \neg a$, pentru orice $a \in A$;
2. $(\neg\varphi)^c = \varphi$, pentru orice $\varphi \in LP$;
3. $(\varphi_1 \vee \varphi_2)^c = (\varphi_1^c \wedge \varphi_2^c)$, pentru orice $\varphi_1, \varphi_2 \in LP$;
4. $(\varphi_1 \wedge \varphi_2)^c = (\varphi_1^c \vee \varphi_2^c)$, pentru orice $\varphi_1, \varphi_2 \in LP$.

Figura 3.6: Determinarea complementului unei formule $\varphi \in LP_{\neg\vee\wedge}$

Pentru determinarea satisifiabilității și a validității sunt calculate închiderea existențială și închiderea universală corespunzătoare formulei și sunt evaluate aceste închideri.

Procesul de găsim al unei substituții care va transforma formula φ_1 într-o formulă φ_2 dată are două etape : verificarea existenței unei substituții și calculul acesteia.

Pentru partea de verificare sunt comparați arborii celor două formule până la nodurile ce conțin simboluri predicative (inclusiv). Dacă sunt găsite două noduri corespondente care au etichete diferite atunci nu există o substituție.

Calculul substituției se face parcurgând arborele și comparând recursiv termenii. Dacă sunt găsite probleme (ex: în prima formulă un nod este simbol funcțional și corespondentul său din a doua formulă este variabilă) atunci va fi returnată o substituție marcată ca fiind invalidă.

3.4 Deducție Naturală

Aplicația oferă suport pentru realizarea manuală a demonstrațiilor prin deducție naturală și pentru verificarea demonstrațiilor aflate în format text. Atunci când utilizatorul încearcă o aplicare incorectă a unei reguli de inferență sau aplică o regulă de inferență după ce a fost demonstrată secvența setată drept obiectiv va fi atenționat și acțiunea sa va fi respinsă.

3.4.1 Aspecte teoretice

O **secvență** este o pereche formată dintr-o mulțime de formule $\{\varphi_1, \dots, \varphi_n\}$ și dintr-o formulă φ , notată $\{\varphi_1, \dots, \varphi_n\} \vdash \varphi$.

O **regulă de inferență** este un tuplu format din :

1. o mulțime de secvențe care se numesc ipotezele regulii
2. o secvență care se numește concluzia regulii
3. o posibilă condiție de aplicare a regulii
4. un nume

Regulile de inferență cu 0 ipoteze se numesc axiome. De asemenea, este posibil să lipsească condiția de aplicare.

O mulțime de reguli de inferență (Figura 3.7) alcătuiesc un sistem deductiv. O demonstrație formală într-un sistem deductiv este o listă de secvențe cu proprietatea că fiecare secvență S_i este justificată de o regulă de inferență a sistemului deductiv aplicată secvențelor anterioare [12] [7].

$$\begin{array}{c}
 \begin{array}{ccc}
 \wedge i \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \varphi'}{\Gamma \vdash (\varphi \wedge \varphi')}, & \wedge e1 \frac{\Gamma \vdash (\varphi \wedge \varphi')}{\Gamma \vdash \varphi}, & \wedge e2 \frac{\Gamma \vdash (\varphi \wedge \varphi')}{\Gamma \vdash \varphi'}, \\
 \rightarrow e \frac{\Gamma \vdash (\varphi \rightarrow \varphi') \quad \Gamma \vdash \varphi}{\Gamma \vdash \varphi'}, & \rightarrow i \frac{\Gamma, \varphi \vdash \varphi'}{\Gamma \vdash (\varphi \rightarrow \varphi')}, & \vee i1 \frac{\Gamma \vdash \varphi_1}{\Gamma \vdash (\varphi_1 \vee \varphi_2)}, \\
 \vee i2 \frac{\Gamma \vdash \varphi_2}{\Gamma \vdash (\varphi_1 \vee \varphi_2)}, & \vee e \frac{\Gamma \vdash (\varphi_1 \vee \varphi_2) \quad \Gamma, \varphi_1 \vdash \varphi' \quad \Gamma, \varphi_2 \vdash \varphi'}{\Gamma \vdash \varphi'}, & \\
 \neg e \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \neg \varphi}{\Gamma \vdash \perp}, & \neg i \frac{\Gamma, \varphi \vdash \perp}{\Gamma \vdash \neg \varphi}, & \perp e \frac{\Gamma \vdash \perp}{\Gamma \vdash \varphi}, \\
 \text{IPOTEZĂ} \frac{}{\Gamma \vdash \varphi} \varphi \in \Gamma, & \text{EXTINDERE} \frac{\Gamma \vdash \varphi}{\Gamma, \varphi' \vdash \varphi}, & \neg \neg e \frac{\Gamma \vdash \neg \neg \varphi}{\Gamma \vdash \varphi}. \\
 \forall e \frac{\Gamma \vdash (\forall x. \varphi)}{\Gamma \vdash \varphi[x \mapsto t]} & & \\
 \exists e \frac{\Gamma \vdash (\exists x. \varphi) \quad \Gamma \cup \{\varphi[x \mapsto x_0]\} \vdash \psi \quad x_0 \notin \text{vars}(\Gamma, \varphi, \psi)}{\Gamma \vdash \psi} & & \\
 \forall i \frac{\Gamma \vdash \varphi[x \mapsto x_0] \quad x_0 \notin \text{vars}(\Gamma, \varphi)}{\Gamma \vdash (\forall x. \varphi)} & & \exists i \frac{\Gamma \vdash \varphi[x \mapsto t]}{\Gamma \vdash (\exists x. \varphi)}
 \end{array}
 \end{array}$$

Figura 3.7: Regulile de inferență din deducția naturală pentru logica de ordinul I. Deducția naturală pentru logica propozițională include aceste reguli fără cele pentru cuantificatori.

3.4.2 Implementare

O componentă de bază a implementării deducției naturale în cadrul aplicației sunt interfețele `InferenceRule` (pentru logica propozițională) și `InferenceRuleFOL` (pentru logica de ordinul I). Una dintre acestea este implementată de toate clasele care definesc reguli de inferență pentru deducția naturală și conține următoarele metode (toate metodele primesc ca argument un număr variabil de instanțe de tip `Object` deoarece regulile de inferență au argumente diferite) :

- `canApply` : primește ca input argumentele pentru aplicarea unei reguli de inferență și returnează `True` dacă regula respectivă se poate aplica cu argumentele date sau `False` în caz contrar
- `apply` : primește aceleași argumente ca și funcția `apply` și returnează un obiect ce reprezintă rezultatul aplicării propriu-zise a regulii respective
- `appliedCorrectly` : primește ca input o secvență ce reprezintă rezultatul aplicării unei reguli și un număr de alte secvențe ce reprezintă argumentele primite de către regula respectivă, returnează "Ok" dacă regula a fost aplicată corect și un mesaj de eroare sugestiv dacă aceasta a fost aplicată greșit.

Redactarea unei demonstrații

Sistemul deductiv este implementat cu ajutorul clasei `DeductiveSystem`. Aceasta conține o listă cu câte o instanță din fiecare regulă de inferență. Pentru inițializarea acestuia trebuie date ca argument niște formule ce vor fi folosite drept ipoteze și o secvență care reprezintă scopul demonstrației. De asemenea, acesta memorează o listă cu secvențele și explicațiile din demonstrația curentă.

Demnă de menționat este metoda `apply`, aceasta primește un număr variabil de argumente ce reprezintă numărul regulii aplicate și argumentele pentru aplicarea acesteia (argumentele numerice sunt transformate în secvențele de la index-ul corespunzător din cadrul demonstrației, dacă acest index este invalid va fi aruncată o excepție). Metoda va căuta regula cu numărul corespunzător și o va aplica cu argumentele date după care va memora rezultatul în demonstrația curentă și numele regulii alături de parametrii aplicării în justificarea pentru secvența nou adăugată. În cazul în care este

întâmpinată vreo eroare în găsirea regulii sau aplicarea este invalidă atunci va fi aruncată o excepție.

Evaluarea unei demonstrații

Pentru a fi evaluată, o demonstrație trebuie redactată în format text. Verificarea unei demonstrații se realizează cu ajutorul claselor `ProofChecker` (și `ProofCheckerFOL`) și `ProofReader` (și `ProofReaderFOL`).

Clasa `ProofChecker` conține o listă cu câte o instanță din fiecare regulă de inferență corespunzătoare deducției naturale pentru tipul de logică vizat. De asemenea, conține două liste unde vor fi adăugate secvențele și justificările lor. Verificarea semantică se face prin intermediul metodei `checkProof` din această clasă, pentru a testa validitatea aplicării unei reguli va fi folosită metoda `appliedCorrectly` din regula respectiva. Pentru a determina argumentele cu care va fi apelată această metodă, dar și regula din care o vom apela, va fi analizată perechea formată din secvența curentă și justificarea acesteia. În caz că evaluarea este pozitivă va fi returnat mesajul "OK", altfel va fi returnat mesajul de eroare furnizat de către `appliedCorrectly` alături de linia corespunzătoare.

Clasa `ProofReader` realizează citirea demonstrației din fișierul text și verificarea ei sintactică. Demonstrația este citită linie cu linie și fiecare este verificată la nivel sintactic, dacă totul este în regulă aceasta va fi despărțită în regula de inferență și justificare și acestea 2 vor fi adăugate într-un obiect de tip `ProofChecker`. Dacă întreaga demonstrație a fost citită fără a se semnala erori sintactice va fi apelată metoda `checkProof` din `ProofChecker` pentru a face evaluarea semantică a demonstrației.

3.5 Forme Normale

3.5.1 Logica propozițională

Aspecte teoretice

O formulă φ se numește literal dacă există o variabilă propozițională $a \in A$ astfel încât $\varphi = a$ sau $\varphi = \neg a$.

O formulă φ se numește clauză dacă există n literali $\varphi_1, \dots, \varphi_n$ astfel încât :

$$\varphi = \varphi_1 \vee \varphi_2 \vee \dots \vee \varphi_n.$$

O formulă φ este în Formă Normală Conjunctivă dacă există n clauze $\varphi_1, \dots, \varphi_n$ astfel încât :

$$\varphi = \varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n$$

O formulă este în FNC dacă este o conjuncție de disjuncții de literali, se mai numește și formă normală clauzală.

O formulă este în FND(Formă Normală Disjunctivă) dacă este o disjuncție de conjuncții de literali.

Pentru orice formulă φ există formulele φ_1 și φ_2 astfel încât:

1. φ_1 este în FNC și $\varphi \equiv \varphi_1$
2. φ_2 este în FND și $\varphi \equiv \varphi_2$

Pentru aducerea unei formule în FNC sau în FND se pot aplica anumite transformări, acestea sunt reprezentate de către următoarele reguli[2] :

1. $(\varphi_1 \leftrightarrow \varphi_2) \equiv ((\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1))$
2. $(\varphi_1 \rightarrow \varphi_2) \equiv (\neg \varphi_1 \vee \varphi_2)$
3. $(\varphi_1 \vee (\varphi_2 \wedge \varphi_3)) \equiv ((\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3))$ (utilă doar pentru transformarea în FNC)
4. $((\varphi_2 \wedge \varphi_3) \vee \varphi_1) \equiv ((\varphi_2 \vee \varphi_1) \wedge (\varphi_3 \vee \varphi_1))$ (utilă doar pentru transformarea în FNC)
5. $(\varphi_1 \wedge (\varphi_2 \vee \varphi_3)) \equiv ((\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \varphi_3))$ (utilă doar pentru transformarea în FND)
6. $((\varphi_2 \vee \varphi_3) \wedge \varphi_1) \equiv ((\varphi_2 \wedge \varphi_1) \vee (\varphi_3 \wedge \varphi_1))$ (utilă doar pentru transformarea în FND)
7. $(\varphi_1 \vee (\varphi_2 \vee \varphi_3)) \equiv ((\varphi_1 \vee \varphi_2) \vee \varphi_3)$
8. $(\varphi_1 \wedge (\varphi_2 \wedge \varphi_3)) \equiv ((\varphi_1 \wedge \varphi_2) \wedge \varphi_3)$
9. $\neg(\varphi_1 \vee \varphi_2) \equiv (\neg \varphi_1 \wedge \neg \varphi_2)$
10. $\neg(\varphi_1 \wedge \varphi_2) \equiv (\neg \varphi_1 \vee \neg \varphi_2)$
11. $\neg \neg \varphi \equiv \varphi$

Implementare

Pentru verificarea apartenenței unei formule la FNC sau la FND sunt folosite clase derivate din clasa abstractă `NormalForm`, aceasta conține următoarele metode:

- `containsImplications` : verifică dacă formula în cauză conține implicații, dacă acest lucru este adevărat atunci nu este în FNC sau FND
- `operatorHierarchy(abstract)` : Metodă ce trebuie implementată de către sub-clase, testează faptul că ierarhia operatorilor în arborele de sintaxă este cea dorită pentru forma normală respectivă
- `checkFormula` : verifică apartenența unei formule la o formă normală folosindu-se de către rezultatele oferite de celelalte 2 metode

Pentru regulile de transformare a fost definită o interfață ce trebuie implementată de clasele ce reprezintă regulile propriu-zise. Acesta interfață este `NormalFormTransformationRule` și conține 2 metode :

- `canApply` : verifică dacă regula poate fi aplicată pe formula dată
- `apply` : realizează aplicarea propriu-zisă a regulii

Transformarea unei formule este realizată cu ajutorul claselor derivate din clasa abstractă `NormalFormTransformer`, aceasta conține o listă de reguli ce va fi inițializată conform regulilor pentru fiecare formă normală și o metodă abstractă ce va primi index-ul regulii care va fi aplicată și formula pe care se va aplica.

Generarea automată unei transformări este făcută prin intermediul unei clase numite `NormalFormTransformationProof`. Aceasta conține 2 obiecte menite să verifice apartenența unei formule la FNC sau FND în funcție de caz și o metodă care face conversia la forma normală dorită.

Algoritmul (Algoritmul 3) încearcă să găsească secvențial câte o regulă care poate fi aplicată pe o subformulă a formulei curente. Pașii sunt explicați clar în transformarea generată, sunt specificate regula aplicată, formula inițială și formula rezultată.

Algorithm 3: Normal form transformation

Result: The transformed formula and the transformation steps

if *the formula is already in the desired normal form* **then**

 | return the formula;

remove the implications;

if *the new formula is not the same as the initial one* **then**

 | add removing implications to the transformation steps;

while *the formula is not in the desired normal form* **do**

 | initialize a list of all the subformulas;

for *rule in transformationRules* **do**

 | **for** *subf in subformula* **do**

 | **if** *the current rule can be applied on the current subformula* **then**

 | apply the rule;

 | replace the transformed subformula in the initial formula;

 | add the rule application to the transformation steps;

 | **end**

 | **end**

end

return the transformation steps and the transformed formula;

3.5.2 Logica de ordinul I

Aspecte Teoretice

O formulă φ este în formă normală Prenex dacă:

$$\varphi = Q_1x_1.Q_2x_2.....Q_nx_n\varphi'$$

unde:

1. $Q_i \in \{\forall, \exists\}$
2. φ' nu conține cuantificatori

Practic, o formulă este în formă normală Prenex (FNP), dacă toți cuantificatorii sunt "în fața formulei". Pentru orice formulă există o formulă echivalentă aflată în formă normală Prenex.

Lema redenumirii : Fie $\varphi \in LP1$ o formulă și $x, y \in X$ două variabile cu proprietatea că $y \notin free(\varphi)$.

Atunci au loc echivalențele :

$$\forall x.\varphi \equiv \forall y.(\sigma^b(\varphi)) \text{ și } \exists x.\varphi \equiv \exists y.(\sigma^b(\varphi))$$

unde $\sigma = \{x \rightarrow y\}$

Pentru a transforma o formulă în corespondența sa aflată în formă normală Prenex se aplică următoarele echivalențe :

1. $(\forall x.\varphi_1) \wedge \varphi_2 \equiv \forall x.(\varphi_1 \wedge \varphi_2)$, dacă $x \notin free(\varphi_2)$
2. $(\forall x.\varphi_1) \vee \varphi_2 \equiv \forall x.(\varphi_1 \vee \varphi_2)$, dacă $x \notin free(\varphi_2)$
3. $(\exists x.\varphi_1) \wedge \varphi_2 \equiv \exists x.(\varphi_1 \wedge \varphi_2)$, dacă $x \notin free(\varphi_2)$
4. $(\exists x.\varphi_1) \vee \varphi_2 \equiv \exists x.(\varphi_1 \vee \varphi_2)$, dacă $x \notin free(\varphi_2)$
5. $\neg\forall x.\varphi \equiv \exists x.\neg\varphi$
6. $\neg\exists x.\varphi \equiv \forall x.\neg\varphi$

În cazul în care una dintre primele patru echivalențe nu poate fi aplicată din cauza restricției $x \in free(\varphi)$, trebuie să aplicăm mai întâi lema redenumirii pentru a redenumi convenabil variabila legată x .

De asemenea, vom folosi la nevoie comutativitatea conectorilor \wedge și \vee și vom înlocui echivalențele[8].

O formulă φ este în formă normală Skolem (prescurtat, FNS) dacă

$$\varphi = \forall x_1, \dots, \forall x_n. \varphi'$$

unde :

1. φ' nu conține cunatificatori
2. $free(\varphi') \subseteq \{x_1, \dots, x_n\}$

Cu alte cuvinte, o formulă este în formă normală Skolem dacă aceasta conține doar cuantificatori universali aflați la începutul formulei, iar toate aparițiile variabilelor din formulă sunt apariții legate (avem cuantificatori pentru toate variabilele ce apar în formulă).

Lema de skolemizare : Fie $\varphi = \forall x_1. \forall x_2. \dots \forall x_n. \exists x \varphi'$, unde $k \geq 0$, $\varphi' \in LP1$ (φ' nu conține alți cunatificatori). Cu alte cuvinte, există k cuantificatori universali înaintea primului cuantificator existențial.

Fie $f \in F_k$ un simbol funcțional de aritate k care nu apare în φ . Avem că φ este echisatisfiabilă cu :

$$\forall x_1. \forall x_2. \dots \forall x_n. (\sigma^b(\varphi))'$$

unde $\sigma = \{x \rightarrow f(x_1, \dots, x_n)\}$

Pentru orice formulă $\varphi \in LP1$ există o formulă $\varphi' \in LP1$ astfel încat : φ' este în formă normală Skolem și φ și φ' sunt echisatisfiabile.

O formulă oarecare $\varphi \in LP1$ poate fi transformată în formă normală Skolem urmând următorii pași :

1. Calculăm o formulă φ_1 , aflată în FNP și echivalentă cu formula φ .
2. Calculăm o formulă φ_2 , închiderea existențială a lui φ_1
3. Aplicăm în mod repetat lema de skolemizare pe formula φ_2

Rezultatul va fi o formulă aflată în FNS, echisatisfabilă cu formula de la care am plecat.

Noțiunea de formă normală conjunctivă este similară cu cea din cardul logicii propoziționale, doar că în cazul de față un literal reprezintă un simbol predicativ de aritate k ($k \geq 0$) aplicat pe k argumente sau negația acestuia. O formulă este în formă normală Skolem Clauzala (FNSC) dacă[9] :

1. φ este în formă normală Skolem
2. φ' este în formă normală clauzală(formă normală conjunctivă), unde $\varphi = \forall x_1. \dots \forall x_n \varphi'$

Implementare

Implementarea pentru forma normală Prenex este asemănătoare cu cele pentru FNC și FND de la logica propozițională. A fost definită o interfață pentru regulile de transformare și este implementată de câte o clasă pentru fiecare regulă.

Clasa `PrenexNormalFormTransformer` conține o listă cu regulile respective, aceasta are ca rol aplicarea regulilor de transformare pe o formulă din LP1 dată. Dacă este nevoie de comutativitatea operatorilor \wedge și \vee atunci aceasta va fi folosită automat. Acest lucru este valabil și pentru lema redenumirii, dacă echivalența nu poate fi aplicată din cauza condiției $x \notin \text{free}(\varphi_2)$ atunci variabila x va fi redenumită automat. Noul nume al acesteia va fi unul format din 1-3 litere mici în așa fel încât să nu existe în φ_1 sau φ_2 .

O transformare în formă normală prenex poate fi realizată automat cu ajutorul clasei `PrenexNormalFormTransformation`. Algoritmul de conversie automată și generare a explicației pentru fiecare transformare efectuată este similar cu cel de la FNC și FND pentru logica propozițională, diferă doar regulile aplicate.

Transformarea în formă normală Skolem se face în conformitate cu pașii enumerați în secțiunea precedentă. Pentru conversia în formă normală Prenex și calcularea închiderii existențiale există deja metode implementate precedent. Pentru a genera o listă cu cuantificatorii existențiali care trebuie scoși și o listă de substituții pentru variabilele cuantificate existențial a fost implementat următorul algoritm (Algoritmul 4).

Algorithm 4: Skolem lemma

Result: A list of substitutions for the existentially quantified variables

if the current node is a quantifier **then**

if the current node is an universal quantifier **then**

 add the quantified variable to the universally quantified variables list;

else if the current node is an existential quantifier **then**

 create a substitution for the existential quantified variable using the
 universally quantified variables list;

 add the quantifier to the existential quantifiers list;

 go to the next node;

După ce sunt calculate cele două liste trebuie scoși cuantificatorii vizați și aplicate substituțiile corespunzătoare, iar rezultatul va fi o formulă aflată în formă normală Skolem.

Pentru a converti în FNSC o formulă ce aparține FNS, este izolată formula de cuantificatorii universalii și este aplicat algoritmul de conversie la FNC de la logica propozițională (Algoritmul 3) (cu tipurile de date adaptate pentru LP1). Apoi sunt puși înapoi cuantificatorii universalii și formula obținută este în FNSC.

3.6 Rezoluție

3.6.1 Aspecte teoretice

Logica Propozițională

Rezoluția este un sistem deductiv cu o singură regulă de inferență (Figura 3.8). Ipotezele și concluzia regulii de inferență sunt clauze.

$$\text{REZOLUȚIE BINARĂ} \frac{C \vee a \quad D \vee \neg a}{C \vee D,}$$

Figura 3.8: Regula de inferență pentru rezoluție în cadrul logicii propoziționale

Rezolventul a două clauze nu este unic. De exemplu, clauzele $p \vee q$ și $\neg p \vee \neg q \vee r$ au doi rezolvenți: $p \vee \neg p \vee r$ și $q \vee \neg q \vee r$. Putem distinge între rezolvenți prin marcarea explicită a variabilei propoziționale a pe care am folosit-o pentru a aplica regula de rezoluție.

Clauza $C \vee D$ rezultată în urma aplicării rezoluției se numește rezolvent al clauzelor $C \vee a$ și $D \vee \neg a$ după/în funcție de literalul a . De exemplu, $p \vee \neg p \vee r$ este rezolventul după q al clauzelor $p \vee q$ și $\neg p \vee \neg q \vee r$, în timp ce $q \vee \neg q \vee r$ este rezolventul lor în funcție de p .

O demonstrație formală a clauzei φ din mulțimea de clauze $\varphi_1, \varphi_2, \dots, \varphi_n$ este o listă de clauze $\phi_1, \phi_2, \dots, \phi_m$ astfel încât, pentru orice $1 \leq i \leq m$:

- sau $\phi_i \in \{\varphi_1, \dots, \varphi_n\}$,
- sau ϕ_i este rezolventul a două clauze ϕ_j, ϕ_k care apare mai “devreme” în demonstrația formală: $1 \leq j, k < i$.

Mai mult, ϕ_m trebuie să fie aceeași formulă cu φ . Clauzele $\varphi_1, \dots, \varphi_n$ se numesc premisele demonstrației formale și φ se numește concluzia demonstrației formale.

Dacă formula în FNC $\varphi_1 \wedge \dots \wedge \varphi_n$ este nesatisfiabilă, atunci există o derivare prin rezoluție a \square (clauza vidă, o clauză ce conține 0 literali) pornind de la clauzele $\varphi_1, \varphi_2, \dots, \varphi_n$ [15].

Logica de ordinul I

Pentru a testa satisfiabilitatea unei formule aflate în FNSC, putem folosi sistemul deductiv descris în secțiunea aceasta.

O formulă φ aflată în FNSC este nesatisfiabilă dacă se poate obține prin rezoluție de bază pornind de la clauzele din φ .

O substituție σ este unificator al termenilor t_1 și t_2 dacă $\sigma^\#(t_1) = \sigma^\#(t_2)$.

Doi termeni sunt unificabili dacă au cel puțin un unificator.

Rezoluția pentru logica de ordinul I este un sistem deductiv alcătuit din două reguli de inferență : REZOLUȚIE BINARĂ și FACTORIZARE POZITIVĂ [10] (Figura 3.9).

Procedura de aplicare a rezoluției de ordinul I este similară cu cea de la logica propozițională.

$$\text{REZOLUȚIE BINARĂ} \frac{P(t_1, \dots, t_n) \vee C_1 \quad \neg P(t'_1, \dots, t'_n) \vee C_2}{\sigma^b(C_1 \vee C_2)} \quad \begin{array}{l} V_1 \cap V_2 = \emptyset \\ \sigma = mgu(\{t_1 \doteq t'_1, \dots, t_n \doteq t'_n\}) \end{array}$$

unde $V_1 = \text{vars}(P(t_1, \dots, t_n) \vee C_1)$ și $V_2 = \text{vars}(\neg P(t'_1, \dots, t'_n) \vee C_2)$.

$$\text{FACTORIZARE POZITIVĂ} \frac{P(t_1, \dots, t_n) \vee P(t'_1, \dots, t'_n) \vee C}{\sigma^b(P(t_1, \dots, t_n) \vee C)} \quad \sigma = mgu(\{t_1 \doteq t'_1, \dots, t_n \doteq t'_n\})$$

Figura 3.9: Regulile de inferență pentru rezoluție în cadrul logicii de ordinul I

3.6.2 Implementare

În cadrul aplicației, rezoluția pentru logica propozițională este implementată cu ajutorul clasei `Resolution`. Aceasta conține o demonstrație prin rezoluție sub forma unei liste de perechi ce conțin o clauză și explicația acesteia. Printre altele, conține metode pentru a verifica dacă rezoluția poate fi aplicată asupra unor parametri dați (index-ii celor două clauze și literalul) și pentru aplicarea propriu-zisă a rezoluției. Au fost implementate clase dedicate literarilor și clauzelor pentru a eficientiza modalitatea de reprezentare și de prelucrare.

Pentru generarea automată a unei demonstrații prin rezoluție este folosită clasa *ResolutionProof*. Algoritmul de determinare al acestei demonstrații (Algoritmul 5) este unul de tip A^* ³. O stare este văzută ca o demonstrație (completă sau incompletă) iar o tranziție este văzută ca fiind aplicarea rezoluției de două dintre clauzele din aceasta. Funcția euristică folosită determină utilitatea unei stări obținute prin numărul de literali din clauza nou adăugată.

Algorithm 5: Resolution Proof Generation

Result: The proof if it exists or null if it doesn't exist

```

for clause1 in the current proof do
    for clause2 in the current proof do
        if the clauses are not the same then
            find all the possible literals that can be used to apply resolution;
            for literal in possibleLiteralsList do
                apply resolution to clause1 and clause2 with literal;
                add the result to the possible transitions list;
            end
        end
    end
end
sort the possible transition list;
for transition in possible transitions do
    copy the current demonstration and add the clause from the current
    transition;
    if the proof is complete then
        return the proof;
    else
        try again with the new proof;
    end
if no complete proof was found then
    return null;

```

Verificarea unei demonstrații aflate în format text se face pe două nivele. Evaluarea demonstrației se face linie cu linie. Dacă o linie este corectă din punct de vedere sintactic se va face evaluarea semantică : se încearcă justificarea clauzei ca rezultat al

³https://en.wikipedia.org/wiki/A*_search_algorithm

rezoluției aplicat clauzelor menționate în explicație cu literalul precizat (dacă este premisă). Dacă a fost parcursă toată demonstrația și nu au fost semnalate probleme atunci aceasta este considerată validă.

Implementarea rezoluției pentru logica de ordinul I este similară din multe puncte de vedere cu cea de la logica propozițională. Pe lângă modul de reprezentare diferit specific logicii de ordinul I, un element de noutate este reprezentat de către algoritmul [17] pentru găsirea celui mai general unificator pentru literali (Algoritmul 6).

Acest algoritm este o componentă de bază din cadrul celor două reguli pentru Rezoluția de ordinul I. Este folosit pentru a testa aplicabilitatea regulilor, iar rezultatul furnizat în cazuri pozitive este folosit pentru aplicarea propriu-zisă a regulilor.

3.7 Quiz

Logic E-Learning Assistant oferă utilizatorului șansa de a-și testa cunoștințele dobândite. Testele sunt structurate pe nivele și trebuie rezolvate secvențial. Acest lucru este realizat prin intermediul unor teste din deducție naturală, forme normale și rezoluție. Interfața acestor teste este similară cu cea de la redactarea demonstrațiilor sau a transformărilor, fiind însă omisă opțiunea de a fi generate automat rezultatele. Atunci când a fost obținut rezultatul dorit se poate face trecerea la următorul subiect, utilizatorul nu va fi lăsat să avanseze la următorul nivel dacă nu l-a rezolvat pe cel curent.

Informațiile relevante pentru această componentă sunt memorate într-o bază de date locală SQLite ce poate fi găsită în fișierul cu resurse al aplicației. În cadrul acesteia se regăsesc următoarele tabele :

- Current Level : memorează nivelul la care a ajuns utilizatorul la fiecare capitol
 1. Chapter : numele capitolului vizat - tip de date : TEXT
 2. Level : nivelul curent al capitolului respectiv - tip de date : INTEGER
 3. PropLogic : apartenența exercițiului la logica propozițională - tip de date : INTEGER (doar cu valorile 1 și 0)
- NaturalDeductionGoals : memorează secvențele care trebuie demonstrate prin deducție naturală
 1. Goal : secvența - tip de date : TEXT

2. PropLogic : apartenența exercițiului la logica propozițională - tip de date : INTEGER (doar cu valorile 1 și 0)
 3. Level : nivelul corespunzător exercițiului - tip de date : INTEGER
- NaturalDeductionHypothesis : memorează ipotezele (una pe rând) de la care se pleacă în demonstrația prin deducție naturală în cadrul unui exercițiu
 1. Formula : formula respectivă - tip de date : TEXT
 2. PropLogic : apartenența exercițiului la logica propozițională - tip de date : INTEGER (doar cu valorile 1 și 0)
 3. Level : nivelul corespunzător exercițiului - tip de date : INTEGER
 - NormalForms : memorează formulele care trebuie aduse într-o formă normală pentru logica propozițională
 1. Formula : formula respectivă - tip de date : TEXT
 2. PropLogic : apartenența exercițiului la logica propozițională - tip de date : INTEGER (doar cu valorile 1 și 0)
 3. ToFNC : dacă formula trebuie transformată în FNC - tip de date : INTEGER (doar cu valorile 1 și 0)
 4. Level : nivelul corespunzător exercițiului - tip de date : INTEGER
 - Resolution : memorează formulele pe care va fi aplicată rezoluția
 1. Formula : formula respectivă - tip de date : TEXT
 2. PropLogic : apartenența exercițiului la logica propozițională - tip de date : INTEGER (doar cu valorile 1 și 0)
 3. Level : nivelul corespunzător exercițiului - tip de date : INTEGER
 - PrenexNormalForm : memorează formulele care trebuie aduse în formă normală Prenex
 1. Formula : formula respectivă - tip de date : TEXT
 2. Level : nivelul corespunzător exercițiului - tip de date : INTEGER

Comunicarea cu baza de date se realizează prin intermediul claselor din pachetul Db-Management :

- `DbConnection` : conține o metodă statică ce returnează o conexiune la baza de date
- `CurrentQuizLevel` : conține metode pentru citirea și modificarea nivelului curent pentru fiecare capitol
- `NaturalDeductionChapter` : responsabilă cu interogarea sau modificarea înregistrărilor din tabelul `NaturalDeduction`
- `NormalFormsChapter` : responsabilă cu interogarea sau modificarea înregistrărilor din tabelul `NormalForms`
- `PrenexNormalFormChapter` : responsabilă cu interogarea sau modificarea înregistrărilor din tabelul `PrenexNormalForm`
- `ResolutionChapter` : responsabilă cu interogarea sau modificarea înregistrărilor din tabelul `Resolution`

Algorithm 6: Unify(ϕ_1, ϕ_2)

Result: An unifier for the two literals or FAILURE

```
if  $\phi_1$  or  $\phi_2$  is a variable or a constant then
|
|   if  $\phi_1$  and  $\phi_1$  are identical then
|   |   return NULL;
|   else if  $\phi_1$  is a variable then
|   |   if  $\phi_1$  occurs in  $\phi_2$  then
|   |   |   return FAILURE;
|   |   else
|   |   |   return  $[\phi_1 \rightarrow \phi_2]$ 
|   else if  $\phi_2$  is a variable then
|   |   if  $\phi_2$  occurs in  $\phi_1$  then
|   |   |   return FAILURE;
|   |   else
|   |   |   return  $[\phi_2 \rightarrow \phi_1]$ 
|   else
|   |   return FAILURE;
|
|   if the initial predicate symbol in  $\phi_1$  and  $\phi_2$  are not the same then
|   |   return FAILURE;
|
|   if  $\phi_1$  and  $\phi_2$  have a different number of arguments then
|   |   return FAILURE;
|
Set Substitution set(SUBST) to NIL;
for  $i=1$  to the number of elements in  $\phi_1$  do
|   Call Unify function with the  $i$ th element of  $\phi_1$  and  $i$ th element of  $\phi_2$ , and put
|   the result into S;
|   if  $S=FAILURE$  then
|   |   return FAILURE;
|   if  $S \neq NULL$  then
|   |   Apply S to the remainder of both L1 and L2;
|   |   SUBST= APPEND(S, SUBST);
end
Return SUBST;
```

Capitolul 4

Evaluarea soluției

Pentru testarea funcționalităților aplicației au fost utilizate unele formule de test și au fost înregistrate rezultatele. Pentru evaluarea componentelor ce includ demonstrații au fost utilizate exemplele găsite în cursurile de Logică pentru Informatică ¹ din cadrul Facultății de Informatică de la Universitatea Alexandru Ioan Cuza din Iași.

4.1 Logica Propozițională

4.1.1 Analiza Formulei

Pentru testarea componentei de analiză a formulei pentru logica propozițională a fost selectată formula $b \leftrightarrow v \wedge x$ și au fost obținute următoarele rezultate :

1. Check Syntax : Valid formula

2. Subformulas : $\text{Subf}(b \leftrightarrow v \wedge x)$

$$= \{ b \leftrightarrow v \wedge x \} \cup \text{Subf}(b) \cup \text{Subf}(v \wedge x)$$

$$= \{ b \leftrightarrow v \wedge x \} \cup \{ b \} \cup \text{Subf}(v \wedge x)$$

$$= \{ b \leftrightarrow v \wedge x \} \cup \{ b \} \cup \{ v \wedge x \} \cup \text{Subf}(v) \cup \text{Subf}(x)$$

$$= \{ b \leftrightarrow v \wedge x \} \cup \{ b \} \cup \{ v \wedge x \} \cup \{ v \} \cup \text{Subf}(x)$$

$$= \{ b \leftrightarrow v \wedge x \} \cup \{ b \} \cup \{ v \wedge x \} \cup \{ v \} \cup \{ x \}$$

3. Variables : $\text{Vars}(b \leftrightarrow v \wedge x)$

$$= \text{Vars}(b) \cup \text{Vars}(v \wedge x)$$

¹<https://profs.info.uaic.ro/logica/logica-2019-2020>

$$\begin{aligned}
&= \{ b \} \cup \text{Vars}(v \wedge x) \\
&= \{ b \} \cup \text{Vars}(v) \cup \text{Vars}(x) \\
&= \{ b \} \cup \{ v \} \cup \text{Vars}(x) \\
&= \{ b \} \cup \{ v \} \cup \{ x \}
\end{aligned}$$

4. Remove Implications : $(\neg b \vee v \wedge x) \wedge (\neg(v \wedge x) \vee b)$
5. Complement : $b \wedge (\neg v \vee \neg x) \vee v \wedge x \wedge \neg b$
6. Tree Height : 3
7. Tree Size : 5
8. Satisfiability : $b \leftrightarrow v \wedge x$ is satisfiable
9. Contradiction : $b \leftrightarrow v \wedge x$ is NOT a contradiction (is satisfiable)
10. Tautology : $b \leftrightarrow v \wedge x$ is NOT a tautology

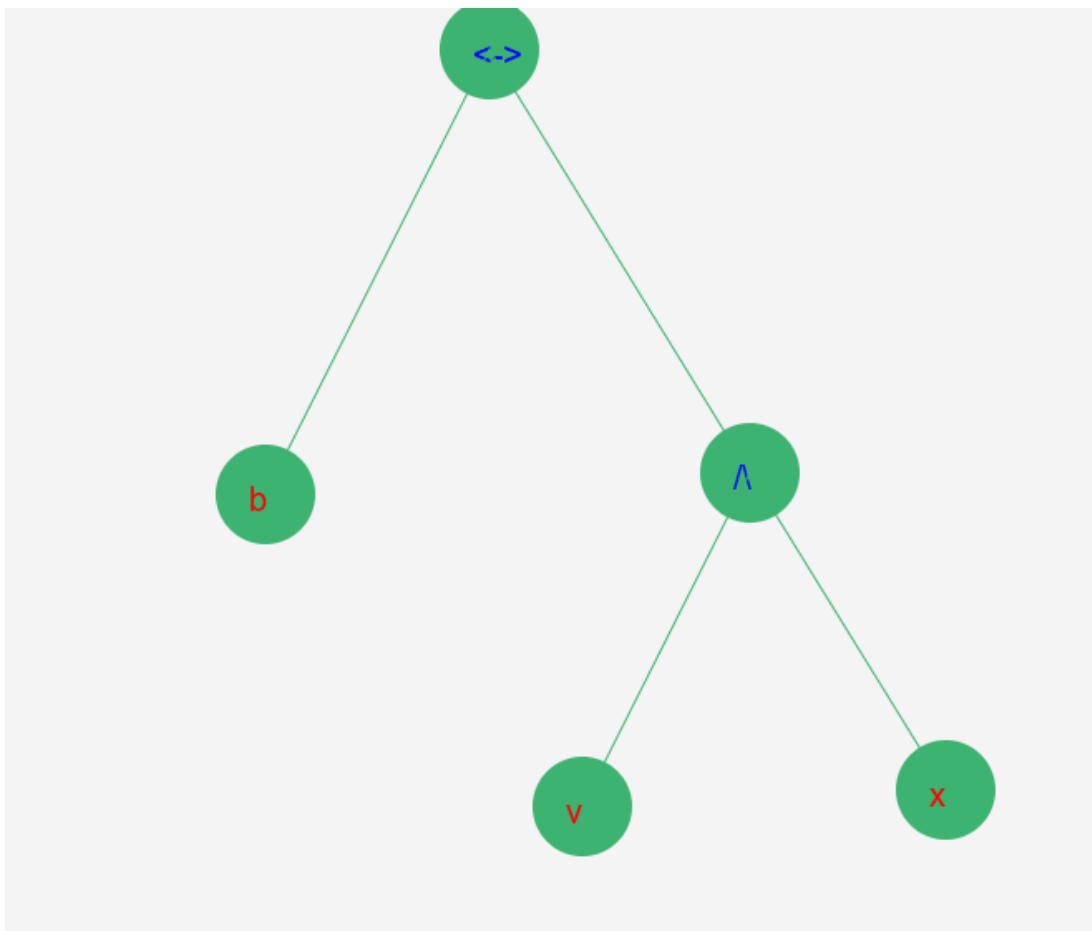


Figura 4.1: Arborele generat pentru formula $b \leftrightarrow v \wedge x$

b	v	x	$b \leftrightarrow v \wedge x$
true	true	true	true
true	true	false	false
true	false	true	false
true	false	false	false
false	true	true	false
false	true	false	true
false	false	true	true
false	false	false	true

Figura 4.2: Tabelul de adevăr generat pentru formula $b \leftrightarrow v \wedge x$

4.1.2 Deducție Naturală

Pentru verificarea componentei de deducție naturală a fost selectată următoarea demonstrație[12] :

1. $\{\neg(p \vee \neg p), p\} \vdash \neg(p \vee \neg p);$ (IPOTEZĂ)
2. $\{\neg(p \vee \neg p), p\} \vdash p;$ (IPOTEZĂ)
3. $\{\neg(p \vee \neg p), p\} \vdash (p \vee \neg p);$ ($\vee i_1, 2$)
4. $\{\neg(p \vee \neg p), p\} \vdash \perp;$ ($\neg e, 1, 3$)
5. $\{\neg(p \vee \neg p)\} \vdash \neg p;$ ($\neg i, 4$)
6. $\{\neg(p \vee \neg p)\} \vdash (p \vee \neg p);$ ($\vee i_2, 5$)
7. $\{\neg(p \vee \neg p)\} \vdash \neg(p \vee \neg p);$ (IPOTEZĂ)
8. $\{\neg(p \vee \neg p)\} \vdash \perp;$ ($\neg e, 7, 6$)
9. $\{\} \vdash \neg\neg(p \vee \neg p);$ ($\neg i, 8$)
10. $\{\} \vdash (p \vee \neg p).$ ($\neg\neg e, 9$)

Figura 4.3: Demonstrația selectată

1. { $!(p \vee !p), p$ } $\vdash !(p \vee !p)$	(IPOTEZA)
2. { $!(p \vee !p), p$ } $\vdash p$	(IPOTEZA)
3. { $!(p \vee !p), p$ } $\vdash p \vee !p$	($\vee i1, 2, !p$)
4. { $!(p \vee !p), p$ } $\vdash \bot$	($!e, 3, 1$)
5. { $!(p \vee !p)$ } $\vdash !p$	($!i, 4, p$)
6. { $!(p \vee !p)$ } $\vdash p \vee !p$	($\vee i2, 5, p$)
7. { $!(p \vee !p)$ } $\vdash !(p \vee !p)$	(IPOTEZA, { $!(p \vee !p)$ } $\vdash !(p \vee !p)$)
8. { $!(p \vee !p)$ } $\vdash \bot$	($!e, 6, 7$)
9. { } $\vdash !! (p \vee !p)$	($!i, 8, !(p \vee !p)$)
10. { } $\vdash p \vee !p$	($!!e, 9$)

Figura 4.4: Demonstrația obținută în cadrul aplicației

Exemplul a fost reprodus cu succes în cadrul aplicației.

Aceeași demonstrație a fost apoi transpusă în format text și verificată. Rezultatul a fost pozitiv, iar orice modificări aduse acesteia ar fi generat erori de natură sintactică sau semantică. De exemplu, prin înlocuirea formulei $\neg(p \vee \neg p)$ din partea dreaptă a secvenței de la linia 7 cu $\neg\neg(p \vee \neg p)$ a rezultat o eroare ce semnalează faptul că partea din dreapta a secvenței nu este inclusă în partea din stânga.

{ $!(p \vee !p), p$ } $\vdash !(p \vee !p)$	(IPOTEZA)
{ $!(p \vee !p), p$ } $\vdash p$	(IPOTEZA)
{ $!(p \vee !p), p$ } $\vdash p \vee !p$	($\vee i1, 2$)
{ $!(p \vee !p), p$ } $\vdash \bot$	($!e, 3, 1$)
{ $!(p \vee !p)$ } $\vdash !p$	($!i, 4$)
{ $!(p \vee !p)$ } $\vdash p \vee !p$	($\vee i2, 5$)
{ $!(p \vee !p)$ } $\vdash !(p \vee !p)$	(IPOTEZA)
{ $!(p \vee !p)$ } $\vdash \bot$	($!e, 6, 7$)
{ } $\vdash !! (p \vee !p)$	($!i, 8$)
{ } $\vdash p \vee !p$	($!!e, 9$)

Figura 4.5: Demonstrația în format text

4.1.3 Forme normale

Pentru redactarea unei transformări în FNC a fost aleasă ca exemplu formula:

$$(p \rightarrow q) \rightarrow (!s \rightarrow \neg r)$$

$$(p \vee s \vee !r) \wedge (!q \vee s \vee !r)$$

Figura 4.6: Formula rezultată în urma transformărilor aplicate

A fost omisă redactarea unei demonstrații în FND deoarece procesul este foarte asemănător. Aceeași formulă a fost folosită pentru a genera automat transformări în FNC și în FND.

```
Removed implications : ( p-> q) ->(!s -> !r) ==> !( !p \vee q ) \vee ( !!s \vee !r )
Rule 5 applied : !( !p \vee q ) \vee ( !!s \vee !r ) ==> !( !p \vee q ) \vee !!s \vee !r
Resulting formula : !( !p \vee q ) \vee !!s \vee !r
Rule 7 applied : !( !p \vee q ) ==> !!p /\ !q
Resulting formula : !!p /\ !q \vee !!s \vee !r
Rule 9 applied : !!s ==> s
Resulting formula : !!p /\ !q \vee s \vee !r
Rule 4 applied : !!p /\ !q \vee s ==> ( !!p \vee s ) /\ ( !q \vee s )
Resulting formula : ( !!p \vee s ) /\ ( !q \vee s ) \vee !r
Rule 9 applied : !!p ==> p
Resulting formula : ( p \vee s ) /\ ( !q \vee s ) \vee !r
Rule 4 applied : ( p \vee s ) /\ ( !q \vee s ) \vee !r ==> ( p \vee s \vee !r ) /\ ( !q \vee s \vee !r )
Resulting formula : ( p \vee s \vee !r ) /\ ( !q \vee s \vee !r )
Transformed formula : ( p \vee s \vee !r ) /\ ( !q \vee s \vee !r )
|
```

Figura 4.7: Transformarea în FNC generată pentru formula $(p \rightarrow q) \rightarrow (!s \rightarrow \neg r)$

```
Removed implications : ( p-> q) ->(!s -> !r) ==> !( !p \vee q ) \vee ( !!s \vee !r )
Rule 5 applied : !( !p \vee q ) \vee ( !!s \vee !r ) ==> !( !p \vee q ) \vee !!s \vee !r
Resulting formula : !( !p \vee q ) \vee !!s \vee !r
Rule 7 applied : !( !p \vee q ) ==> !!p /\ !q
Resulting formula : !!p /\ !q \vee !!s \vee !r
Rule 9 applied : !!s ==> s
Resulting formula : !!p /\ !q \vee s \vee !r
Rule 9 applied : !!p ==> p
Resulting formula : p /\ !q \vee s \vee !r
Transformed formula : ( p /\ !q ) \vee s \vee !r
```

Figura 4.8: Transformarea în FND generată pentru formula $(p \rightarrow q) \rightarrow (!s \rightarrow \neg r)$

Poate fi observat faptul că formula finală din transformarea generată automat corespunde cu cea obținută prin redactarea manuală a unei transformări. De asemenea, toate formulele obținute, prin redactarea manuală a demonstrației sau prin generarea automată a acestora corespund formei normale vizate.

4.1.4 Rezoluție

Pentru verificarea componentei de rezoluție a fost selectată următoarea demonstrație [15]:

1. $p \vee \neg q$;	(premisă)
2. q ;	(premisă)
3. $\neg p$;	(premisă)
4. p ;	(rezoluție, 2, 1, $a = q$)
5. \square .	(rezoluție, 4, 3, $a = p$)

Figura 4.9: Demonstrația selectată

Aceeași demonstrație a fost reprodusă în cardul aplicației :

1. { $p, !q$ }	(premisa)
2. { q }	(premisa)
3. { $!p$ }	(premisa)
4. { p }	(1 , 2 , q)
5. { }	(3 , 4 , p)

Figura 4.10: Demonstrația redactată în cadrul aplicației

Demonstrația generată automat este identică cu cea redactată.

Aceeași demonstrație a fost transpusă în format text, unii indexi din explicații au fost inversați deoarece convenția pentru rezoluție din cardul aplicației cere ca index-ul clauzei ce conține literalul pozitiv să fie primul.

{ $p, !q$ }	(premisa)
{ q }	(premisa)
{ $!p$ }	(premisa)
{ p }	(2 , 1 , q)
{ }	(4 , 3 , p)

Figura 4.11: Demonstrația în format text

Rezultatul evaluării pentru demonstrație este pozitiv. O modificare ce ar afecta corectitudinea demonstrației ar rezulta în furnizarea unui mesaj de eroare ce afișează numărul liniei și comunică faptul că rezoluția nu poate fi aplicată cu parametrii dați.

4.2 Logica de ordinul I

4.2.1 Analiza Formulei

Pentru testarea componentei de analiză a formulei pentru logica propozițională am luat ca exemplu formula $\exists x.P(F(x), y) \wedge G(y, H(x))$. Fișierele ce conțin implementările pentru simbolurile predicative și cele funcționale, asignarea și domeniile variabilelor au următorul conținut:

- Asignări :

$$y=10$$

- Domenii :

$$x=[1,2,3,4,5,6,7,8,9,10]$$

$$y=[1,2,3,4,5,6,7,8,9,10]$$

- Implementări :

1. Simboluri predicative :

$$P(x,y) = x==y$$

$$G(x,y) = x<y$$

2. Simboluri funcționale :

$$F(x) = x+5$$

$$H(x) = x * x$$

Au fost obținute următoarele rezultate :

1. Check Syntax : Valid formula

2. Subformulas : Subf($\exists x.P(F(x), y) \wedge G(y, H(x))$)

$$= \{ \exists x.P(F(x), y) \wedge G(y, H(x)) \} \cup \text{Subf}(P(F(x), y) \wedge G(y, H(x)))$$

$$= \{ \exists x.P(F(x), y) \wedge G(y, H(x)) \} \cup \{ P(F(x), y) \wedge G(y, H(x)) \} \cup \text{Subf}(P(F(x), y)) \cup \text{Subf}(G(y, H(x)))$$

$$= \{ \exists x.P(F(x), y) \wedge G(y, H(x)) \} \cup \{ P(F(x), y) \wedge G(y, H(x)) \} \cup \{ P(F(x), y) \} \cup \text{Subf}(G(y, H(x)))$$

$$= \{ \exists x.P(F(x), y) \wedge G(y, H(x)) \} \cup \{ P(F(x), y) \wedge G(y, H(x)) \} \cup \{ P(F(x), y) \} \cup \{ G(y, H(x)) \}$$

3. Variables : $\text{Vars}(\exists x.P(F(x), y) \wedge G(y, H(x)))$
 $= \text{Vars}(P(F(x), y) \wedge G(y, H(x)))$
 $= \text{Vars}(P(F(x), y)) \cup \text{Vars}(G(y, H(x)))$
 $= \text{Vars}(F(x)) \cup \text{Vars}(y) \cup \text{Vars}(G(y, H(x)))$
 $= \text{Vars}(x) \cup \text{Vars}(y) \cup \text{Vars}(G(y, H(x)))$
 $= \{x\} \cup \text{Vars}(y) \cup \text{Vars}(G(y, H(x)))$
 $= \{x\} \cup \{y\} \cup \text{Vars}(G(y, H(x)))$
 $= \{x\} \cup \{y\} \cup \text{Vars}(y) \cup \text{Vars}(H(x))$
 $= \{x\} \cup \{y\} \cup \{y\} \cup \text{Vars}(H(x))$
 $= \{x\} \cup \{y\} \cup \{y\} \cup \text{Vars}(x)$
 $= \{x\} \cup \{y\} \cup \{y\} \cup \{x\}$
4. Remove Implications : $\exists x.P(F(x), y) \wedge G(y, H(x))$
5. Tree Height : 5
6. Tree Size : 10
7. Evaluation : True
8. Satisfiability : True
9. Tautology : False

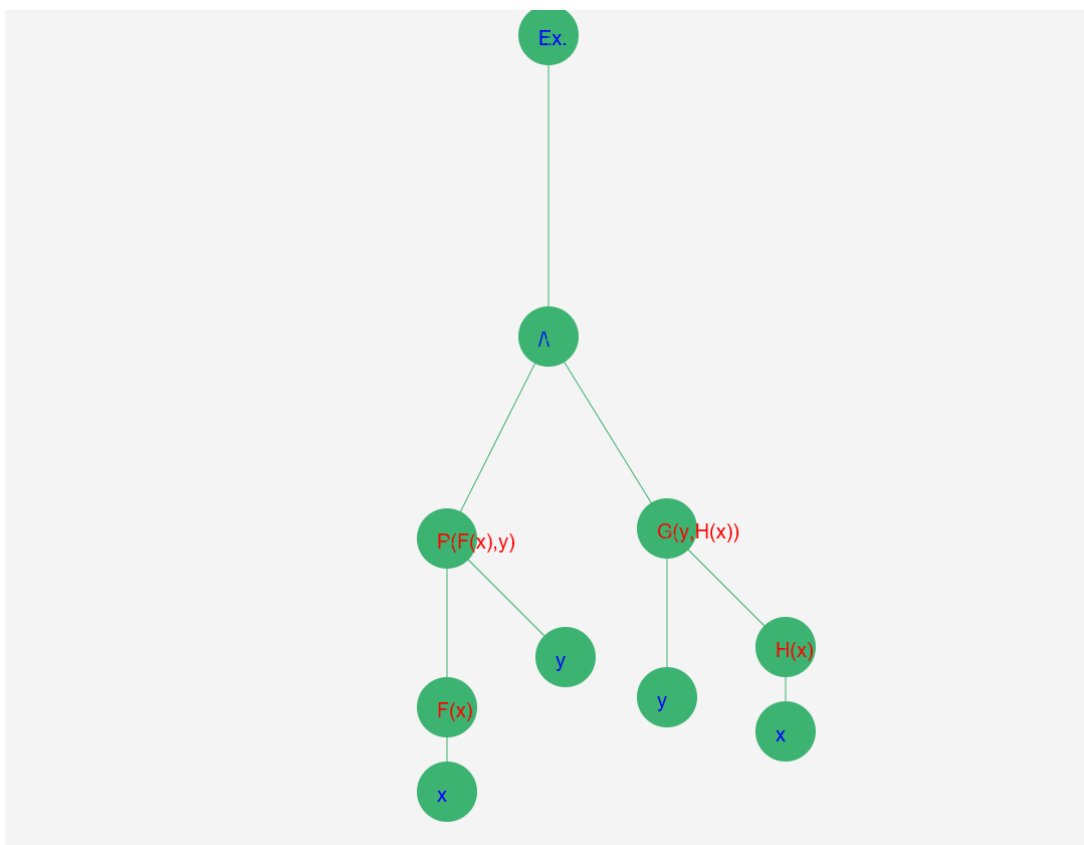


Figura 4.12: Arborele generat pentru formula $\exists x.P(F(x),y) \wedge G(y,H(x))$

Pentru generarea substituției s-a încercat găsirea unei substituții care să transforme formula $P(x) \wedge H(a, J(y, a))$ în formula $P(b) \wedge H(q, J(B(y), q))$. Rezultatul furnizat a fost $[x \rightarrow b, y \rightarrow B(y), a \rightarrow q]$. Dacă modificăm prima formulă în $P(F(x)) \wedge H(a, J(y, a))$ nu va fi găsită o substituție deoarece corespondentul funcției $F(x)$ în cealaltă formulă este o variabilă. Orice modificare asupra formulelor ce le afectează arborele până la simbolurile predicative (inclusiv) va duce de asemenea la un rezultat negativ.

4.2.2 Deducție Naturală

Pentru verificarea componentei de deducție naturală a fost selectată următoarea demonstrație [7] :

1. $\{\forall x.(P(x) \rightarrow Q(x)), \exists x.P(x)\} \vdash \exists x.P(x)$ (IPOTEZĂ)
2. $\{\forall x.(P(x) \rightarrow Q(x)), \exists x.P(x), P(x_0)\} \vdash P(x_0)$ (IPOTEZĂ)
3. $\{\forall x.(P(x) \rightarrow Q(x)), \exists x.P(x), P(x_0)\} \vdash \forall x.(P(x) \rightarrow Q(x))$ (IPOTEZĂ)
4. $\{\forall x.(P(x) \rightarrow Q(x)), \exists x.P(x), P(x_0)\} \vdash (P(x_0) \rightarrow Q(x_0))$ ($\forall e, 3, x_0$)
5. $\{\forall x.(P(x) \rightarrow Q(x)), \exists x.P(x), P(x_0)\} \vdash Q(x_0)$ ($\rightarrow e, 4, 2$)
6. $\{\forall x.(P(x) \rightarrow Q(x)), \exists x.P(x), P(x_0)\} \vdash \exists x.Q(x)$ ($\exists i, 5$)
7. $\{\forall x.(P(x) \rightarrow Q(x)), \exists x.P(x)\} \vdash \exists x.Q(x)$ ($\exists e, 1, 6$)

Figura 4.13: Demonstrația selectată

```

1.{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- Vx.P(x) -> Q(x)      (IPOTEZA)
2.{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- Ex.P(x)              (IPOTEZA)
3.{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- P(xi)                (IPOTEZA)
4.{ Vx.(P(x) ->Q(x)), Ex.P(x) } |- Ex.P(x)                    ( IPOTEZA, { Vx.(P(x) ->Q(x)), Ex.P(x) } |- Ex.P(x) )
5.{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- P(xi) -> Q(xi)      ( Ve, 1, xi )
6.{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- Q(xi)                ( ->e, 5, 3 )
7.{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- Ex.Q(x)              ( Ei, 6, xi, x )
8.{ Vx.P(x) -> Q(x), Ex.P(x) } |- Ex.Q(x)                     ( Ee, 4, 7 )

```

Figura 4.14: Demonstrația obținută în cadrul aplicației

Exemplul a fost reprodus cu succes în cadrul aplicației. Diferențele sunt minore și nu afectează în vreun fel decursul sau corectitudinea demonstrației.

Aceeași demonstrație a fost apoi transpusă în format text și verificată. Rezultatul a fost pozitiv, iar orice modificări aduse acesteia ce i-ar fi afectat corectitudinea ar fi generat erori de natură sintactică sau de natură semantică. De exemplu, dacă la justificarea ultimei secvențe am fi pus index-ul 2 în loc de 4 ar fi fost comunicat un mesaj de eroare deoarece nu este respectată condiția aplicării regulii.

```

{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- Vx.P(x) -> Q(x)      (IPOTEZA)
{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- Ex.P(x)              (IPOTEZA)
{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- P(xi)                (IPOTEZA)
{ Vx.(P(x) ->Q(x)), Ex.P(x) } |- Ex.P(x)                    ( IPOTEZA)
{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- P(xi) -> Q(xi)      ( Ve, 1 )
{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- Q(xi)                ( ->e, 5, 3 )
{ Vx.P(x) -> Q(x), Ex.P(x), P(xi) } |- Ex.Q(x)              ( Ei, 6 )
{ Vx.P(x) -> Q(x), Ex.P(x) } |- Ex.Q(x)                     ( Ee, 4, 7 )

```

Figura 4.15: Demonstrația în format text

4.3 Forme Normale

Observație : Numele variabilelor create pentru lema redenumiri și numele simbolurilor funcționale și predicative noi adăugate pentru lema de skolemizare au fost generate aleator în cadrul aplicației respectând condițiile de corectitudine.

Pentru redactarea unei transformări în formă normală Prenex a fost aleasă formula $(\forall x. \neg(P(x, x) \wedge \neg \exists y. P(x, y)) \wedge P(x, x))$ [https://profs.info.uaic.ro/logica/logica-2019-2020/curs4_fol.pdf].

Vun.(Edh.!(P(un,un) ∧ !P(un,dh)) ∧ P(x,x))

Figura 4.16: Formula rezultată în urma transformărilor aplicate

Aceeași formulă a fost folosită și pentru generarea automată a transformărilor în Formă Normală Prenex, Formă Normală Skolem și Formă Normală Skolem Clauzală.

Poate fi observat faptul că atât formula obținută prin redactarea transformării, cât și cele obținute prin transformările automate se află în forma normală corespunzătoare.

```
Rule 1 applied : ( Vx.!( P(x,x) /\ !( Ey.P(x,y) ) ) ) /\ P(x,x) ==> Vf1.!( P(f1,f1) /\ !( Ey.P(f1,y) ) ) /\ P(x,x)
Resulting formula : Vf1.!( P(f1,f1) /\ !( Ey.P(f1,y) ) ) /\ P(x,x)
Rule 6 applied : !( Ey.P(x,y) ) ==> Vy.!(P(x,y)
Resulting formula : Vf1.!( P(f1,f1) /\ !( Ey.P(f1,y) ) ) /\ P(x,x)
Rule 6 applied : !( Ey.P(f1,y) ) ==> Vy.!(P(f1,y)
Resulting formula : Vf1.!( P(f1,f1) /\ ( Vy.!(P(f1,y) ) ) /\ P(x,x)
Rule 1 applied : P(f1,f1) /\ ( Vy.!(P(f1,y) ) ) ==> Vnb.P(f1,f1) /\ !P(f1,nb)
Resulting formula : Vf1.!( Vnb.P(f1,f1) /\ !P(f1,nb) ) /\ P(x,x)
Rule 5 applied : !( Vnb.P(f1,f1) /\ !P(f1,nb) ) ==> Enb.!( P(f1,f1) /\ !P(f1,nb) )
Resulting formula : Vf1.( Enb.!( P(f1,f1) /\ !P(f1,nb) ) ) /\ P(x,x)
Rule 3 applied : ( Enb.!( P(f1,f1) /\ !P(f1,nb) ) ) /\ P(x,x) ==> Era.!( P(f1,f1) /\ !P(f1,ra) ) /\ P(x,x)
Resulting formula : Vf1.( Era.!( P(f1,f1) /\ !P(f1,ra) ) /\ P(x,x) )
Transformed formula : Vf1.( Era.!( P(f1,f1) /\ !P(f1,ra) ) /\ P(x,x) )
```

Figura 4.17: Transformarea în Formă Normală Prenex generată pentru formula $(\forall x. \neg(P(x, x) \wedge \neg \exists y. P(x, y)) \wedge P(x, x))$

```

Rule 1 applied : ( Vx.!( P(x,x) /\ !( Ey.P(x,y) ) ) ) /\ P(x,x) ==> Vu.!( P(u,u) /\ !( Ey.P(u,y) ) ) /\ P(x,x)
Resulting formula : Vu.!( P(u,u) /\ !( Ey.P(u,y) ) ) /\ P(x,x)
Rule 6 applied : !( Ey.P(x,y) ) ==> Vy.!(P(x,y)
Resulting formula : Vu.!( P(u,u) /\ !( Ey.P(u,y) ) ) /\ P(x,x)
Rule 6 applied : !( Ey.P(u,y) ) ==> Vy.!(P(u,y)
Resulting formula : Vu.!( P(u,u) /\ ( Vy.!(P(u,y) ) ) /\ P(x,x)
Rule 1 applied : P(u,u) /\ ( Vy.!(P(u,y) ) ) ==> Vj.P(u,u) /\ !P(u,j)
Resulting formula : Vu.!( Vj.P(u,u) /\ !P(u,j) ) /\ P(x,x)
Rule 5 applied : !( Vj.P(u,u) /\ !P(u,j) ) ==> Ej.!( P(u,u) /\ !P(u,j) )
Resulting formula : Vu.( Ej.!( P(u,u) /\ !P(u,j) ) ) /\ P(x,x)
Rule 3 applied : ( Ej.!( P(u,u) /\ !P(u,j) ) ) /\ P(x,x) ==> Ei.!( P(u,u) /\ !P(u,i) ) /\ P(x,x)
Resulting formula : Vu.( Ei.!( P(u,u) /\ !P(u,i) ) /\ P(x,x) )
Transformed formula : Vu.( Ei.!( P(u,u) /\ !P(u,i) ) /\ P(x,x) )
Existential closure : Vu.( Ei.!( P(u,u) /\ !P(u,i) ) /\ P(x,x) ) => Ex.Vu.( Ei.!( P(u,u) /\ !P(u,i) ) /\ P(x,x) )
Removed existential quantifier : Ex.
Removed existential quantifier : Ei.
Replaced variable : x => T()
Replaced variable : i => FAW(u)
Transformed formula : Vu.!( P(u,u) /\ !P(u,FAW(u)) ) /\ P(T(),T())

```

Figura 4.18: Transformarea în Formă Normală Skolem generată pentru formula

$$(\forall x. \neg(P(x, x) \wedge \neg \exists y. P(x, y)) \wedge P(x, x))$$

```

Rule 1 applied : ( Vx.!( P(x,x) /\ !( Ey.P(x,y) ) ) ) /\ P(x,x) ==> Vjy.!( P(jy,jy) /\ !( Ey.P(jy,y) ) ) /\ P(x,x)
Resulting formula : Vjy.!( P(jy,jy) /\ !( Ey.P(jy,y) ) ) /\ P(x,x)
Rule 6 applied : !( Ey.P(x,y) ) ==> Vy.!(P(x,y)
Resulting formula : Vjy.!( P(jy,jy) /\ !( Ey.P(jy,y) ) ) /\ P(x,x)
Rule 6 applied : !( Ey.P(jy,y) ) ==> Vy.!(P(jy,y)
Resulting formula : Vjy.!( P(jy,jy) /\ ( Vy.!(P(jy,y) ) ) /\ P(x,x)
Rule 1 applied : P(jy,jy) /\ ( Vy.!(P(jy,y) ) ) ==> Vbf.P(jy,jy) /\ !P(jy,bf)
Resulting formula : Vjy.!( Vbf.P(jy,jy) /\ !P(jy,bf) ) /\ P(x,x)
Rule 5 applied : !( Vbf.P(jy,jy) /\ !P(jy,bf) ) ==> Ebf.!( P(jy,jy) /\ !P(jy,bf) )
Resulting formula : Vjy.( Ebf.!( P(jy,jy) /\ !P(jy,bf) ) ) /\ P(x,x)
Rule 3 applied : ( Ebf.!( P(jy,jy) /\ !P(jy,bf) ) ) /\ P(x,x) ==> Ee.!( P(jy,jy) /\ !P(jy,e) ) /\ P(x,x)
Resulting formula : Vjy.( Ee.!( P(jy,jy) /\ !P(jy,e) ) /\ P(x,x) )
Transformed formula : Vjy.( Ee.!( P(jy,jy) /\ !P(jy,e) ) /\ P(x,x) )
Existential closure : Vjy.( Ee.!( P(jy,jy) /\ !P(jy,e) ) /\ P(x,x) ) => Ex.Vjy.( Ee.!( P(jy,jy) /\ !P(jy,e) ) /\ P(x,x) )
Removed existential quantifier : Ex.
Removed existential quantifier : Ee.
Replaced variable : x => UO()
Replaced variable : e => ZGU(jy)
Transformed formula : Vjy.!( P(jy,jy) /\ !P(jy,ZGU(jy)) ) /\ P(UO(),UO())
Quantifiers Removed : Vjy.!( P(jy,jy) /\ !P(jy,ZGU(jy)) ) /\ P(UO(),UO()) => !( P(jy,jy) /\ !P(jy,ZGU(jy)) ) /\ P(UO(),UO())
Converting to FNC...
Rule 6 applied : !( P(jy,jy) /\ !P(jy,ZGU(jy)) ) ==> !P(jy,jy) \/\ !P(jy,ZGU(jy))
Resulting formula : ( !P(jy,jy) \/\ !P(jy,ZGU(jy)) ) /\ P(UO(),UO())
Rule 7 applied : !!P(jy,ZGU(jy)) ==> P(jy,ZGU(jy))
Resulting formula : ( !P(jy,jy) \/\ P(jy,ZGU(jy)) ) /\ P(UO(),UO())
Transformed formula : ( !P(jy,jy) \/\ P(jy,ZGU(jy)) ) /\ P(UO(),UO())
Transformed Formula : Vjy.( !P(jy,jy) \/\ P(jy,ZGU(jy)) ) /\ P(UO(),UO())

```

Figura 4.19: Transformarea în Formă Normală Skolem generată pentru formula

$$(\forall x. \neg(P(x, x) \wedge \neg \exists y. P(x, y)) \wedge P(x, x))$$

4.4 Rezoluția de ordinul I

Pentru verificarea componentei de rezoluție a fost selectată formula :

$$\forall x.\forall b.(P(b) \wedge (\neg P(H(x)) \vee Q(F(x))) \wedge (\neg Q(F(G(A()))))).$$

```
1.{ P(b) }      ( premisa )
2.{ !P(H(x)) , Q(F(x)) }      ( premisa )
3.{ !Q(F(G(A())) )      ( premisa )
4.{ Q(F(x)) }      ( 1 , 2 , P )
5.{ }      ( 3 , 4 , Q )
```

Figura 4.20: Demonstrația redactată în cadrul aplicației

Demonstrația generată automat este identică cu cea redactată.

Aceeași demonstrație a fost transpusă în format text, unii indexi din explicații au fost inversați deoarece convenția pentru rezoluție din cardul aplicației cere ca index-ul clauzei ce conține literalul pozitiv să fie primul.

```
{ P(b) }      ( premisa )
{ !P(H(x)) , Q(F(x)) }      ( premisa )
{ !Q(F(G(A())) )      ( premisa )
{ Q(F(x)) }      ( 1 , 2 , P )
{ }      ( 4 , 3 , Q )
```

Figura 4.21: Demonstrația în format text

Rezultatul evaluării pentru demonstrație este pozitiv. O modificare ce ar afecta corectitudinea demonstrației ar rezulta în furnizarea unui mesaj de eroare ce afișează numărul liniei și comunică faptul că rezoluția nu poate fi aplicată cu parametrii dați.

4.5 Obeservații pe baza rezultatelor

În secțiunile precedente am putut observa corectitudinea rezultatelor furnizate de către aplicație. Acestea sunt reproduceri fidele ale rezultatelor teoretice. Output-ul a fost conform așteptărilor și pentru input-urile valide și pentru cele invalide.

Componentele ce reprezintă redactări de demonstrații sau transformări sunt similare cu scrierea acestora în format text în limbajul specific domeniului. Generările de demonstrații sau transformări produc rezultate valide, iar procesul prin care au fost generate acestea este vizibil și corect din punct de vedere teoretic.

Partea de verificare a unor rezultate deja prezente realizează o evaluare riguroasă și semnalează clar eventualele erori.

Cu toate că sunt totuși prezente unele restricții, cum ar fi ordinea parametrilor la rezoluție sau la regulile de inferență din cadrul deducției naturale, acestea sunt limitări ce pot fi ușor trecute cu vederea și nu afectează expresivitatea ce poate fi manifestată în utilizarea aplicației. Abaterea de la aceste convenții este semnalată clar și nu afectează în vreun fel funcționalitatea, acțiunea este negată și utilizatorul poate încerca o nouă variantă care să corespundă cerințelor.

Concluzii

Studiul logicii poate reprezenta o provocare din cauza lipsei de aplicații ce facilitează învățarea oferind o componentă practică pentru a o completa pe cea teoretică. Acest proces poate fi îngreunat de imposibilitatea lucrului individual într-un mediu care să ofere verificarea rezultatelor obținute sau să faciliteze obținerea acestora pas cu pas.

Sunt soluții deja existente demne de luat în seamă, însă majoritatea au măcar un neajuns. Acesta poate fi lipsa interacțiunii cu utilizatorul, neexplicarea pașilor urmați, limbajul diferit de cel utilizat în logică sau dificultatea utilizării. Chiar dacă există variante care să nu prezinte aceste neajunsuri, ele nu oferă un pachet de funcții complet, pot fi utilizate cu succes doar pentru anumite capitole. Pe lângă acest fapt, componenta de verificare a cunoștințelor acumulate sau a altor rezultate obținute este absentă.

Logic E-Learning Assistant este o unealtă destinată învățării logicii. Facilitează studiul prin oferirea unui cadru plăcut pentru experimentarea cu noțiunile fundamentale, însă oferă suport și pentru capitole mai complexe. Limbajul utilizat în cadrul aplicației este apropiat de cel folosit în domeniu, lucru ce face folosirea ei foarte intuitivă. Oferă posibilitatea de a produce rezultate, dar și de a le verifica pe cele deja existente. Pe lângă experimentele realizate după cum dorește utilizatorul, aplicația vine și cu un set de exerciții pentru a-și testa cunoștințele dobândite.

Pachetul de funcționalități este unul complet, Logic E-Learning Assistant oferă suport pentru toate capitolele fundamentale din cadrul Logicii Propoziționale și a Logicii de Ordinul I. Este ușor de folosit și furnizează explicații pentru rezultatele generate automat, dar previne și apariția erorilor în cele redactate manual. Funcționalitățile sunt separate clar în funcție de capitolul de care aparțin, dar și în funcție de utilitatea lor, dacă sunt componente de redactare sau componente de verificare. Utilizatorul este ajutat și încurajat să învețe și să experimenteze sau să își testeze cunoștințele.

Este pusă baza pentru implementarea unor extensii cu adevărat complexe. O

primă idee ar putea fi generarea automată de demonstrații prin deducție naturală, aceasta la randul ei ar putea reprezenta începutul pentru o componentă dedicată planning-ului. O altă extensie utilă ar fi adăugarea unei secțiuni pentru problemele NP, aceasta se poate folosi de sistemul care lucrează cu formule logice deja implementat și de posibilitatea reducerii problemelor respective la probleme din domeniul logicii. O variantă de îmbogățire a aplicației ar putea fi și adăugarea de procesare a limbajului natural, transformarea propozițiilor în formule logice și evaluarea acestora în funcție de alte afirmații.

Bibliografie

- [1] Bruno Barras **and others**. "The Coq proof assistant reference manual: Version 6.1". **in:** (1997).
- [2] *Forme normale*. URL: <https://profs.info.uaic.ro/~logica/logica-2019-2020/curs6.pdf>.
- [3] Matthew B Hoy. "Wolfphram— Alpha: a brief introduction". **in:** *Medical reference services quarterly* 29.1 (2010), **pages** 67–74.
- [4] Stephen C Johnson **and others**. *Yacc: Yet another compiler-compiler*. **volume** 32. Bell Laboratories Murray Hill, NJ, 1975.
- [5] Viswanathan Kodaganallur. "Incorporating language processing into java applications: A javacc tutorial". **in:** *IEEE software* 21.4 (2004), **pages** 70–77.
- [6] *Logică pentru Informatică - Curs 1*. URL: <https://profs.info.uaic.ro/~logica/logica-2019-2020/curs1.pdf>.
- [7] *Logică pentru Informatică - Săptămâna 11 Deducția Naturală*. **december** 2019. URL: https://profs.info.uaic.ro/~logica/logica-2019-2020/curs3_fol.pdf.
- [8] *Logică pentru Informatică - Săptămâna 12 Forme normale ale formulelor de ordinul I*. **december** 2019. URL: https://profs.info.uaic.ro/~logica/logica-2019-2020/curs4_fol.pdf.
- [9] *Logică pentru Informatică - Săptămâna 13 Forme normale ale formulelor de ordinul I - Partea a II-a*. **january** 2020. URL: https://profs.info.uaic.ro/~logica/logica-2019-2020/curs5_fol.pdf.
- [10] *Logică pentru Informatică - Săptămâna 14 Rezoluția pentru LP1*. URL: https://profs.info.uaic.ro/~logica/logica-2019-2020/curs6_fol.pdf.
- [11] *Logică pentru Informatică - Săptămâna 2 Sintaxa Logicii Propoziționale*. URL: <https://profs.info.uaic.ro/~logica/logica-2019-2020/curs2.pdf>.

- [12] *Logică pentru Informatică - Săptămâna 3 Semantica Logicii Propoziționale*. URL: <https://profs.info.uaic.ro/~logica/logica-2019-2020/curs3.pdf>.
- [13] *Logică pentru Informatică - Săptămâna 9 Sintaxa logicii de ordinul I*. **december** 2019. URL: https://profs.info.uaic.ro/~logica/logica-2019-2020/curs1_fol.pdf.
- [14] Tobias Nipkow, Lawrence C Paulson **and** Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*. **volume** 2283. Springer Science & Business Media, 2002.
- [15] *Rezoluție în Logica Propozițională*. URL: <https://profs.info.uaic.ro/~logica/logica-2019-2020/curs7.pdf>.
- [16] Jussi Rintanen. *Propositional Logic and Its Applications in Artificial Intelligence*. URL: <https://mycourses.aalto.fi/pluginfile.php/273655/course/section/60890/notes-logic.pdf>.
- [17] *Unification in First-order logic*. URL: <https://www.javatpoint.com/ai-unification-in-first-order-logic>.
- [18] Johan Vos **and others**. "Using Scene Builder to Create a User Interface". **in:** *Pro JavaFX 9*. Springer, 2018, **pages** 129–191.

Appendix

Tehnologii utilizate

Interfața grafică a fost realizată cu ajutorul lui **Scene Builder** [18]. Acesta este un tool pentru layout ce îi permite utilizatorului să construiască partea vizuală a interfeței fără a scrie cod. Acest proces este de tipul drag and drop, proprietățile elementelor pot fi modificate și se pot folosi fișiere CSS. Pentru fiecare fereastră este generat un FXML care poate fi folosit în cadrul proiectului².

JavaCC [5] este un tool utilizat pentru generarea de parsere și a fost folosit în cadrul aplicației pentru partea de verificare sintactică. Acesta generează parsere top-down, spre deosebire de alte variante asemănătoare cu YACC [4] care generează bottom-up. JavaCC generează parsere de tipul LL(1) în general, acest lucru poate fi însă schimbat după nevoi. Elementele generate de către acesta sunt în întregime în Java, lucru ce previne dependențele sau erorile de portare.

Clase pentru regulile de inferență

Pentru regulile de inferență au fost definite următoarele clase ce implementează `InferenceRule` sau `InferenceRuleFOL`:

- `CreateConjunction` și `CreateConjunctionFOL` - $\wedge i$
- `ExtractFromConjunction1` și `ExtractFromConjunction1FOL` - $\wedge e_1$
- `ExtractFromConjunction2` și `ExtractFromConjunction2FOL` - $\wedge e_2$
- `ExtractFromImplication` și `ExtractFromImplicationFOL` - $\rightarrow e$
- `CreateImplication` și `CreateImplicationFOL` - $\rightarrow i$
- `CreateDisjunction1` și `CreateDisjunction1FOL` - $\vee i1$

²<https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html>

- CreateDisjunction2 și CreateDisjunction2FOL - $\vee i2$
- RemoveDisjunction și RemoveDisjunctionFOL - $\vee e$
- CreateBottom și CreateBottomFOL - $\neg e$
- CreateNegationFromBottom și CreateNegationFromBottomFOL - $\neg i$
- CreateProvenFromBottom și CreateProvenFromBottomFOL - $\perp e$
- Hypothesis și HypothesisFOL - IPOTEAZĂ
- Extension și ExtensionFOL - EXTINDERE
- RemoveDoubleNegation și RemoveDoubleNegationFOL - $\neg\neg e$
- RemoveUniversalCuantifier (doar pentru logica de ordinul I, implementează InferenceRuleFOL) - $\forall e$
- RemoveExistentialCuantifier (doar pentru logica de ordinul I, implementează InferenceRuleFOL) - $\exists e$
- CreateUniversalCuantifier (doar pentru logica de ordinul I, implementează InferenceRuleFOL) - $\forall i$
- CreateExistentialCuantifier (doar pentru logica de ordinul I, implementează InferenceRuleFOL) - $\exists i$

Clase pentru regulile de transformare în FNC sau FND

1. RemoveDoubleImplication și RemoveDoubleImplicationFOL

$$(\varphi_1 \leftrightarrow \varphi_2) \equiv ((\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1))$$
2. RemoveImplication și RemoveImplicationFOL

$$(\varphi_1 \rightarrow \varphi_2) \equiv (\neg\varphi_1 \vee \varphi_2)$$
3. ReplaceLeftDisjunction și ReplaceLeftDisjunctionFOL

$$(\varphi_1 \vee (\varphi_2 \wedge \varphi_3)) \equiv ((\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3)) \text{ (utilă doar pentru transformarea în FNC)}$$
4. ReplaceRightDisjunction și ReplaceRightDisjunctionFOL

$$((\varphi_2 \wedge \varphi_3) \vee \varphi_1) \equiv ((\varphi_2 \vee \varphi_1) \wedge (\varphi_3 \vee \varphi_1)) \text{ (utilă doar pentru transformarea în FNC)}$$

5. ReplaceLeftConjunction

$$(\varphi_1 \wedge (\varphi_2 \vee \varphi_3)) \equiv ((\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \varphi_3)) \text{ (utilă doar pentru transformarea în FND)}$$

6. ReplaceRightConjunction

$$((\varphi_2 \vee \varphi_3) \wedge \varphi_1) \equiv ((\varphi_2 \wedge \varphi_1) \vee (\varphi_3 \wedge \varphi_1)) \text{ (utilă doar pentru transformarea în FND)}$$

7. DisjunctionAssociativity și DisjunctionAssociativityFOL

$$(\varphi_1 \vee (\varphi_2 \vee \varphi_3)) \equiv ((\varphi_1 \vee \varphi_2) \vee \varphi_3)$$

8. ConjunctionAssociativity și ConjunctionAssociativityFOL

$$(\varphi_1 \wedge (\varphi_2 \wedge \varphi_3)) \equiv ((\varphi_1 \wedge \varphi_2) \wedge \varphi_3)$$

9. DeMorganDisjunction și DeMorganDisjunctionFOL

$$\neg(\varphi_1 \vee \varphi_2) \equiv (\neg\varphi_1 \wedge \neg\varphi_2)$$

10. DeMorganConjunction și DeMorganConjunctionFOL

$$\neg(\varphi_1 \wedge \varphi_2) \equiv (\neg\varphi_1 \vee \neg\varphi_2)$$

11. RemoveDoubleNegation și RemoveDoubleNegationFOL

$$\neg\neg\varphi \equiv \varphi$$

Clase pentru regulile de transformare în Formă Normală Prenex

1. UniversalCuantifierConjunction

$$(\forall x.\varphi_1) \wedge \varphi_2 \equiv \forall x.(\varphi_1 \wedge \varphi_2), \text{ dacă } x \notin \text{free}(\varphi_2)$$

2. UniversalCuantifierDisjunction

$$(\forall x.\varphi_1) \vee \varphi_2 \equiv \forall x.(\varphi_1 \vee \varphi_2), \text{ dacă } x \notin \text{free}(\varphi_2)$$

3. ExistentialCuantifierConjunction

$$(\exists x.\varphi_1) \wedge \varphi_2 \equiv \exists x.(\varphi_1 \wedge \varphi_2), \text{ dacă } x \notin \text{free}(\varphi_2)$$

4. ExistentialCuantifierDisjunction

$$(\exists x.\varphi_1) \vee \varphi_2 \equiv \exists x.(\varphi_1 \vee \varphi_2), \text{ dacă } x \notin \text{free}(\varphi_2)$$

5. UniversalCuantifierNegated

$$\neg\forall x.\varphi \equiv \exists x.\neg\varphi$$

6. ExistentialCuantifierNegated

$$\neg \exists x. \varphi \equiv \forall x. \neg \varphi$$

Exemple Interfața grafică

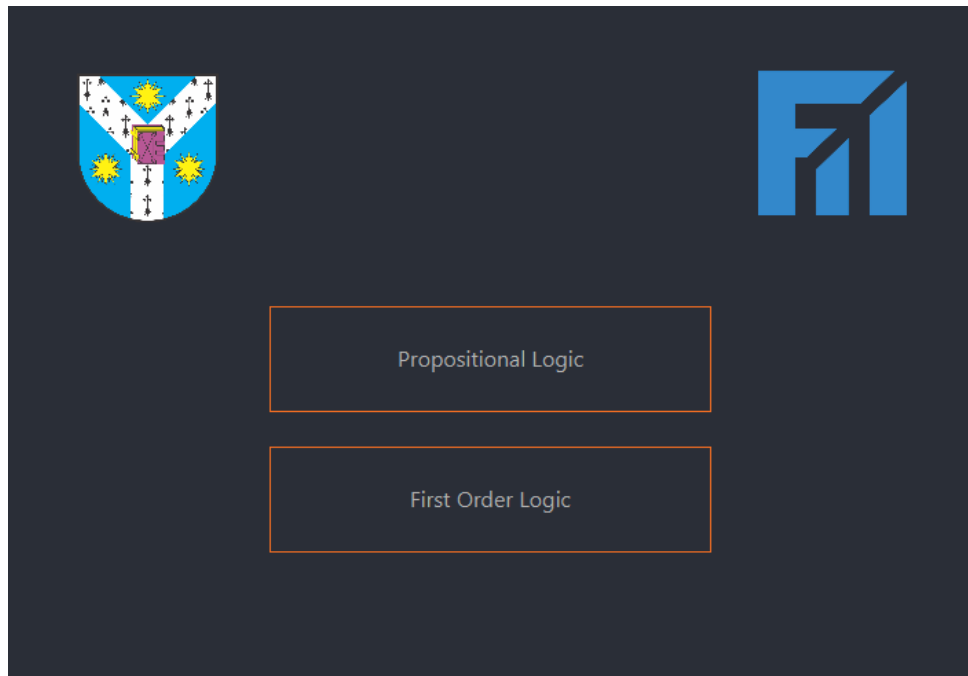


Figura 4.22: Meniul Principal

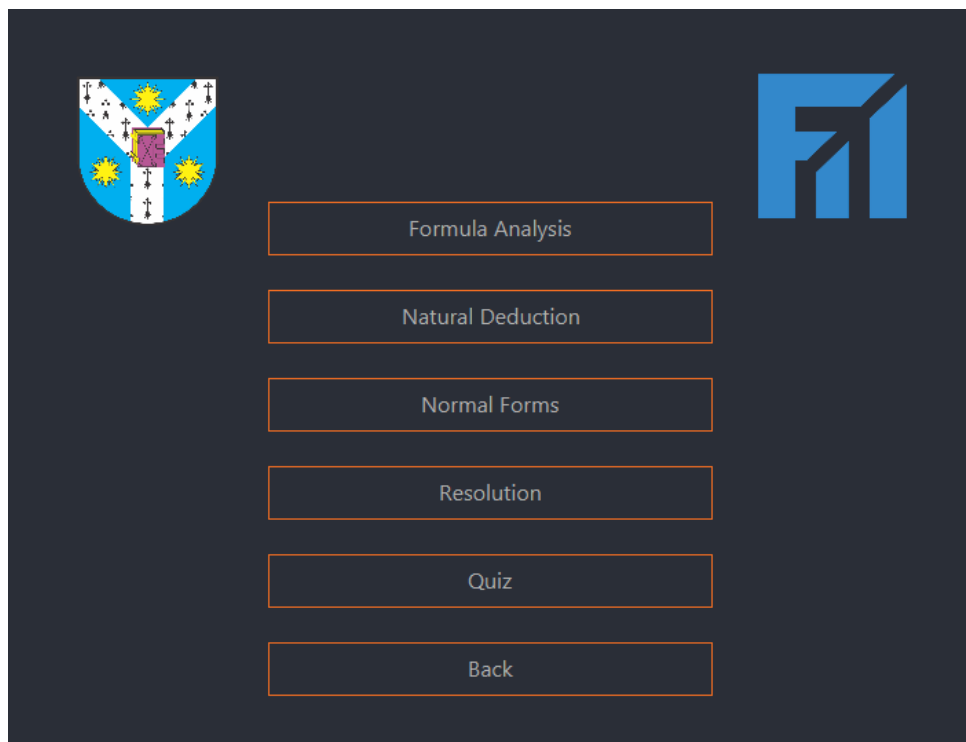


Figura 4.23: Selectarea capitolului

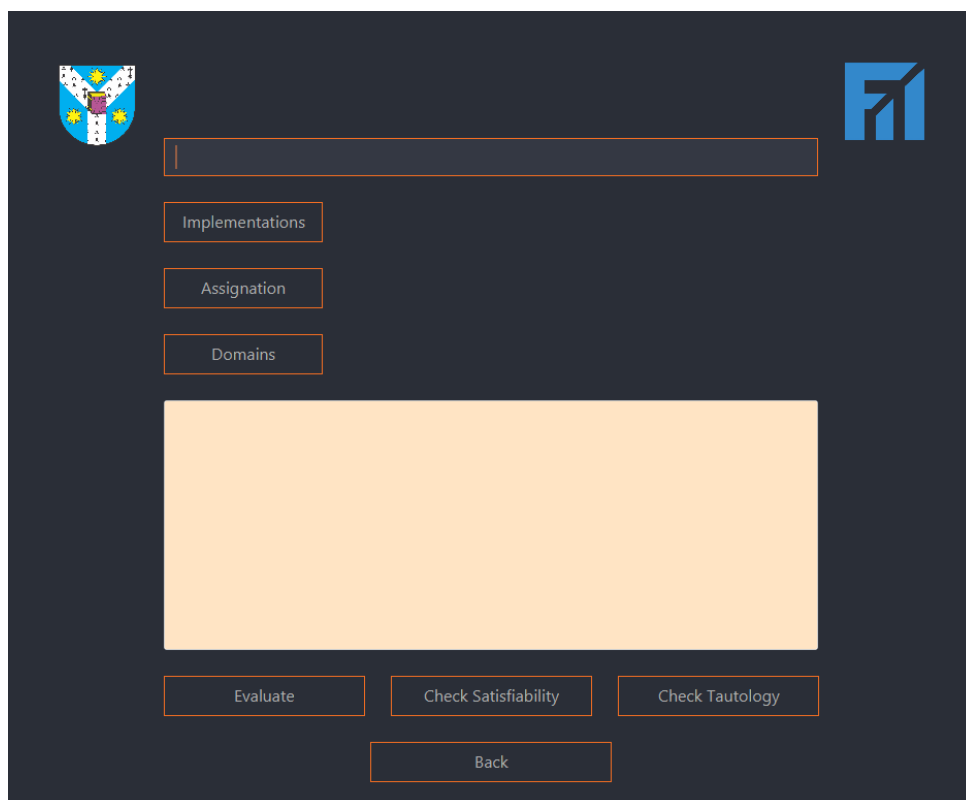


Figura 4.24: Evaluarea formulelor din LP1

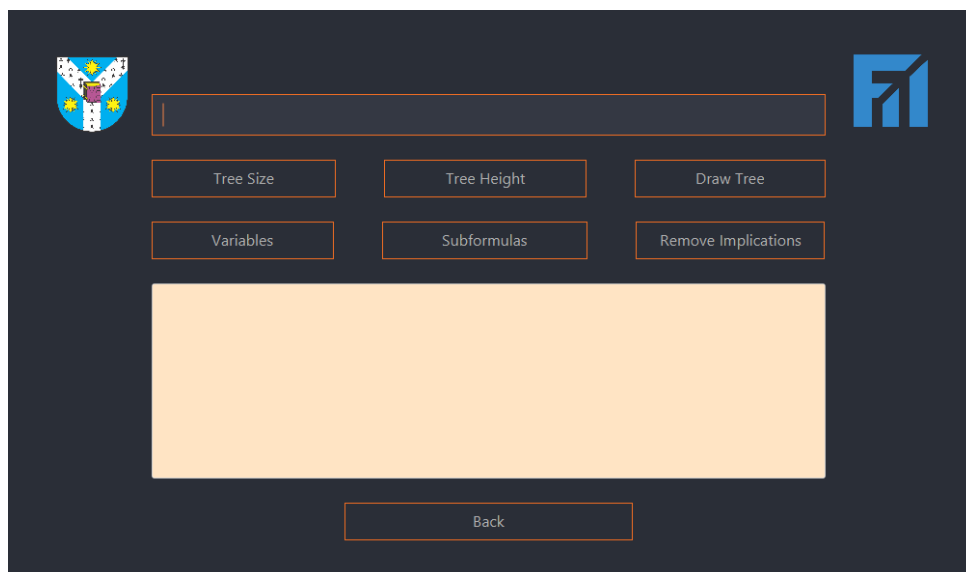


Figura 4.25: Analiza formulelor din LP1

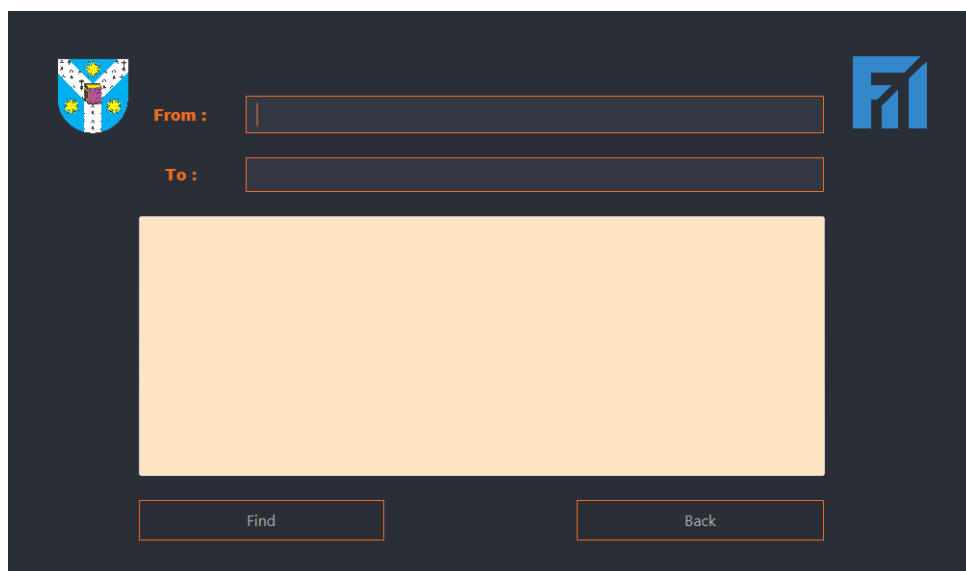


Figura 4.26: Găsirea unei substituții pentru a transforma o formulă din LP1 în altă formulă din LP1

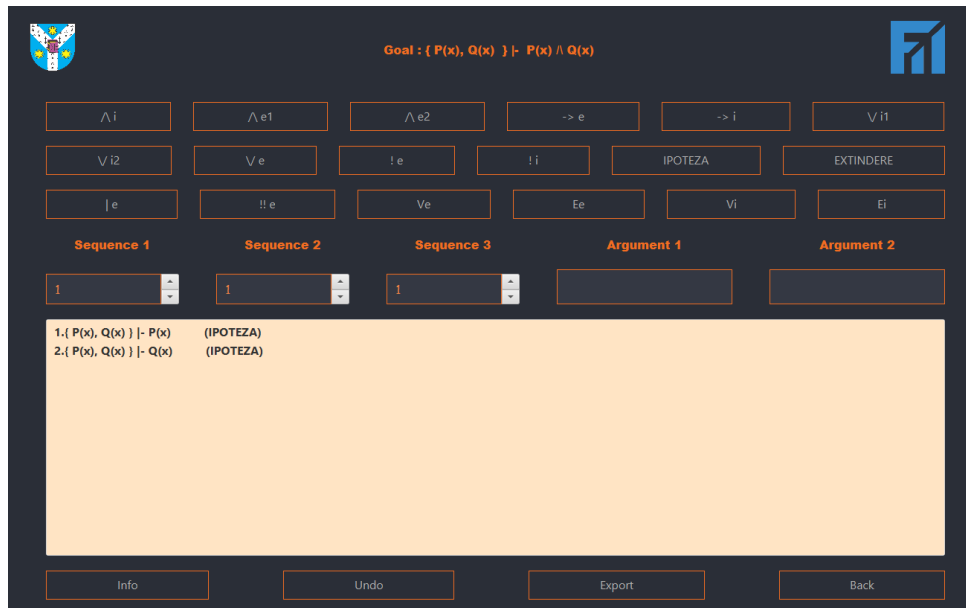


Figura 4.27: Redactarea unei demonstrații prin deducție naturală pentru LP1, similară cu cea pentru LP

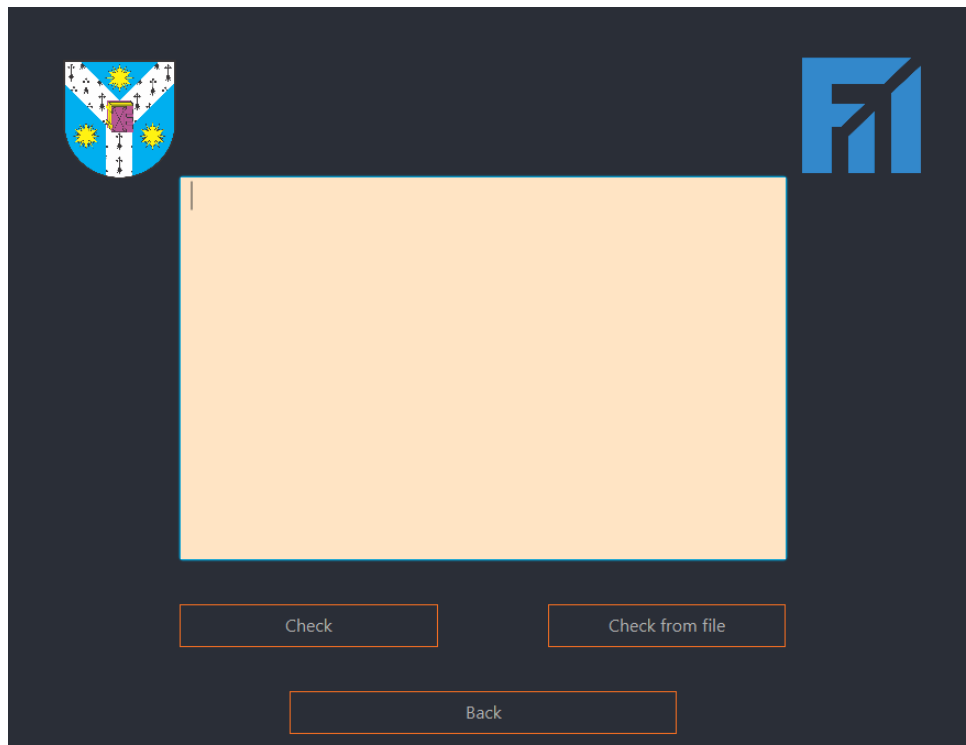




Figura 4.28: Evaluarea unei demonstrații prin deducție naturală sau rezoluție pentru LP sau LP1, aspectul interfeței este identic pentru toate 4

Clause 1

Clause 2

Literal

Apply

Positive Factorization

Generate Proof



Set Formula

Undo

Export

Back

Figura 4.29: Redactarea unei demonstrații prin rezoluție pentru LP1, similară cu cea pentru LP

From :

Result :

Transform into Prenex

Generate Prenex Transformation

Transform into Skolem

Generate Skolem Transformation

Transform into Skolem Clausal

Generate Skolem Clausal Transformation

Back

Figura 4.30: Generarea unei transformări în formă normală pentru LP1