**ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ**

**ԻՆՖՈՐՄԱՏԻԿԱՅԻ ԵՎ ԿԻՐԱՌԱԿԱՆ ՄԱԹԵՄԱՏԻԿԱՅԻ ՖԱԿՈՒԼՏԵՏ**

**Ծրագրավորման և Ինֆորմացիոն տեխնոլոգիաների ամբիոն**

**ՀԱՇՎՈՂԱԿԱՆ ՄԵՔԵՆԱՆԵՐԻ, ՀԱՄԱԼԻՐՆԵՐԻ, ՀԱՄԱԿԱՐԳԵՐԻ ԵՎ ՑԱՆՑԵՐԻ ՄԱԹԵՄԱՏԻԿԱԿԱՆ ԵՎ ԾՐԱԳՐԱՅԻՆ ԱՊԱՀՈՎՈՒՄ**

**ՈՍԿԱՆՅԱՆ ՎԱՀԱԳՆ ԳևՈՐԳԻ**

**ՄԱԳԻՍՏՐՈՍԱԿԱՆ ԹԵԶ**

**ՄԵՔԵՆԱՅԱԿԱՆ** **ՈՒՍՈՒՑՄԱՆ ՄԵԹՈԴՆԵՐԻ ԿԻՐԱՌՈՒՄԸ ՌԵԶՈԼՅՈՒՏԻՎ ԱՐՏԱԾՄԱՆ ՄԵՋ**

***«Տեղեկատվական տեխնոլոգիաներ» մասնագիտությամբ***

***Ինֆորմատիկայի և կիրառական մաթեմատիկայի մագիստրոսի որակավորման աստիճանի հայցման համար***

**ԵՐԵՎԱՆ 2025**

***Ուսանող`****\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*ստորագրություն*

***Ոսկանյան Վահագն***

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ազգանուն, անուն

***Գիտական ղեկավար՝\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

*ստորագրություն*

***ֆ․մ.գ.թ. , դոցենտ, Հովհաննես Բոլիբեկյան***

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

գիտ. աստիճան, կոչում, ազգանուն, անուն

***«Թույլատրել պաշտպանության»***

***Ամբիոնի վարիչ`\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

ստորագրություն

***ֆ․մ.գ.թ. , դոցենտ, Սարգսյան Ս․***

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

գիտ. աստիճան, կոչում, ազգանուն, անուն

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025թ.

# 

# ՀԱՄԱՌՈՏԱԳԻՐ

ՄԵՔԵՆԱՅԱԿԱՆ ՈՒՍՈՒՑՄԱՆ ՄԵԹՈԴՆԵՐԻ ԿԻՐԱՌՈՒՄԸ ՌԵԶՈԼՅՈԻՏԻՎ ԱՐՏԱԾՄԱՆ ՄԵՋ

ПРИМЕНЕНИЕ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ В РЕЗОЛЮТИВНОМ ВЫВОДЕ

THE APPLICATION OF MACHINE LEARNING METHODS IN RESOLUTION INFERENCE

Այս աշխատանքը հետազոտում է առաջին կարգի տրամաբանության մեջ ռեզոլյուցիայի մեթոդի արդյունավետության բարձրացման խնդիրը՝ կենտրոնանալով յուրաքանչյուր քայլում ռեզոլյուցիայի համար լիտերալների օպտիմալ զույգերի ընտրության վրա: Ռեզոլյուցիան, որպես ավտոմատ ապացուցման հիմնական գործիք, հաճախ բախվում է հաշվողական բարդության խնդիրների՝ պայմանավորված լիտերալների ոչ արդյունավետ ընտրությամբ, ինչը հանգեցնում է որոնման տարածության էքսպոնենցիալ աճի:

Ուսումնասիրությունը նպատակ ունի մշակել նոր մոտեցում, որը կօգտագործի մեքենայական ուսուցման մոդել, որը կսովորի օպտիմալ լուծված խնդիրներից և կկանխատեսի ռեզոլյուցիայի ժամանակ լիտերալների ամենաարդյունավետ զույգի ընտրությունը: Այն կնվազեցնի որոնման տարածությունը, կբարելավի ապացուցման արագությունը և ապացույց գտնելու հնարավորությունը։

Փորձարկումները ցույց են տալիս, որ առաջարկվող մոտեցումը նվազեցնում է ապացուցման քայլերի քանակը և կրճատում ապացուցման ժամանակը՝ զգալիորեն բարելավելով ավտոմատ թեորեմ ապացուցող համակարգերի արդյունավետությունը: Աշխատանքի արդյունքները կարող են կիրառվել ֆորմալ վերիֆիկացիայի, ծրագրային ապահովման ստուգման և արհեստական բանականության տրամաբանական համակարգերում:

[ՆԵՐԱԾՈՒԹՅՈՒՆ 5](#_Toc198813535)

[Գլուխ 1 6](#_Toc198813536)

[1.1 Հիմնական հասկացություններ 6](#_Toc198813537)

[1.2 Դևիսի և Փաթնեմի մեթոդը 7](#_Toc198813538)

[1.3 Ռեզոլյուցիայի մեթոդը տրամաբանական արտահայտություններում 8](#_Toc198813539)

[1.4 Տեղադրություն և ունիֆիկացիա 10](#_Toc198813540)

[1.5 Ունիֆիկացիայի ալգորիթմ 12](#_Toc198813541)

[1.6 Ռեզոլյուցիայի մեթոդը առաջին կարգի պրեդիկատային համակարգերում 14](#_Toc198813542)

[Գլուխ 2 16](#_Toc198813543)

[2.1 TPTP գրադարանի նկարագրություն 16](#_Toc198813544)

[2.2 Vampire ATP համակարգի նկարագրություն 17](#_Toc198813545)

[2.3 GNN մոդելի նկարագրություն 18](#_Toc198813546)

[2.4 TPTP գրադարանի աքսիոմների օգտագործում 22](#_Toc198813547)

[2.5 Սինթետիկ տվյալների բազայի գեներացման մեթոդաբանություն 23](#_Toc198813548)

[Գլուխ 3 26](#_Toc198813549)

[3.1 Լիտերալների ունիֆիկացիայի և ռեզոլյուցիայի օժանդակ մոդուլ 26](#_Toc198813550)

[3.2 Սինթետիկ աքսիոմների գեներացում 26](#_Toc198813551)

[3.3 Սինթետիկ խնդիրների գեներացում 27](#_Toc198813552)

[3.4 Խնդիրների լուծում և ապացույցների մշակում 30](#_Toc198813553)

[3.5 Մեքենայական ուսուցման մոդելի ուսուցում 32](#_Toc198813554)

[3.6 Մոդելի թեստավորում 35](#_Toc198813555)

[ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ 37](#_Toc198813556)

[ՕԳՏԱԳՈՐԾՎԱԾ ԳՐԱԿԱՆՈՒԹՅԱՆ ՑԱՆԿ 38](#_Toc198813557)

# ՆԵՐԱԾՈՒԹՅՈՒՆ

Ռեզոլյուցիայի մեթոդը հանդիսանում է առաջին կարգի տրամաբանության մեջ ավտոմատ ապացուցման հիմնական գործիքներից մեկը, սակայն դրա արդյունավետությունը էականորեն կախված է լիտերալների ընտրության ռազմավարությունից։ Ուսումնասիրության արդիականությունը պայմանավորված է տրամաբանական խնդիրների ավտոմատ լուծման համակարգերի կատարելագործման անհրաժեշտությամբ, հատկապես բարդ խնդիրների համար, որտեղ որոնման տարածությունն էքսպոնենցիալ է աճում։

Աշխատանքի նպատակն է մշակել ռեզոլյուցիայի մեթոդում լիտերալների օպտիմալ ընտրության մոտեցում՝ մեքենայական ուսուցման տեխնոլոգիաների կիրառմամբ։ Հիմնական խնդիրներն են՝ լիտերալների ընտրության համար կարևոր հատկանիշների բացահայտումը, մեքենայական ուսուցման համար որակյալ տվյալների հավաքագրման մեթոդաբանության մշակումը, և մոդելի ինտեգրումը գործող ավտոմատ ապացուցման համակարգերում։

Ուսումնասիրության օբյեկտը առաջին կարգի տրամաբանության մեջ ռեզոլյուցիայի մեթոդն է, իսկ առարկան՝ ռեզոլյուցիայի ընթացքում լիտերալների ընտրության ավտոմատացումը մեքենայական ուսուցման միջոցով։ Հետազոտության վարկածն այն է, որ մեքենայական ուսուցման մոդելների կիրառումը կարող է զգալիորեն կրճատել որոնման տարածությունը և ապացուցման ժամանակը՝ ուսումնասիրելով հաջողված ապացույցների օրինաչափությունները։

# Գլուխ 1

## 1.1 Հիմնական հասկացություններ

Դիտարկենք առաջին կարգի բանաձևեր, որոնք կառուցված են փոփոխականներից, ֆունկցիոնալ նշաններից, պրեդիկատային նշաններից և տրամաբանական կապերերից: Կաշխատենք հետևյալ տրամաբանական նշանների հետ՝ ⊤ (ճշմարիտ), ⊥ (կեղծ), ¬ (ժխտում), ∨ (դիզյունկցիա), ∧ (կոնյունկցիա), → (հետևում է), ↔ (համարժեք է) ։

**Սահմանում՝** *թերմերը* սահմանվում են ռեկուրսիվ կերպով հետևյալ ձևով՝

1. Հաստատունը թերմ է։
2. Փոփոխականը թերմ է։
3. Եթե -ը -տեղանի ֆունկցիոնալ սիմվոլ է և -ը թերմեր են, ապա -ը թերմ է։
4. Վերը նշված կանոնների կիրառմամբ ստացված թերմերից բացի այլ թերմեր չկան։

**Սահմանում՝** եթե -ն -տեղանի պրեդիկատային սիմվոլ է և -ը թերմեր են, ապա -ը *ատոմ* է։

**Սահմանում՝** *լիտերալը* հանդիսանում է ատոմ կամ ատոմի ժխտում։

**Սահմանում՝** ճիշտ կառուցված բանաձևերը, կամ *առաջին կարգի տրամաբանության բանաձևերը*, ռեկուրսիվ կերպով սահմանվում են հետևյալ կերպ՝

1. Ատոմը բանաձև է:
2. Եթե -ը և -ն բանաձևեր են, ապա , , , և նույնպես բանաձևեր են:
3. Եթե -ը բանաձև է, իսկ -ը ազատ փոփոխական է -ում, ապա և նույնպես բանաձևեր են:
4. Բանաձևերը ստեղծվում են միայն 1, 2 և 3 կանոնների կիրառման վերջավոր քանակությամբ:

**Սահմանում՝**  բանաձևը գտնվում է *կոնյունկտիվ նորմալ ձևում (ԿՆՁ)* այն և միայն այն դեպքում, երբ -ը ունի տեսք, որտեղ յուրաքանչյուր հանդիսանում է լիտերալների դիզյունկցիա։

Ցանկացած բանաձև հնարավոր է փոխակերպել կոնյունկտիվ նորմալ ձևի։

**Թեորեմ 1.1[2]՝** ենթադրենք տրված են բանաձևերը և բանաձևը։ Այդ դեպքում բանաձևը հանդիսանում է -ի *տրամաբանական հետևանք* այն և միայն այն դեպքում, երբ -ն անհամատեղելի է։

## 1.2 Դևիսի և Փաթնեմի մեթոդը

Ենթադրենք՝ -ը դիզյունկտների բազմություն է։ Մեթոդը, ըստ էության, բաղկացած է հետևյալ չորս կանոններից`

1. *Նույնաբանության կանոն՝* -ից ջնջում ենք բոլոր նույնաբանական հիմնական դիզյունկտները։ Մնացած բազմությունը անհամատեղելի է, այն և միայն այն դեպքում, եթե -ը անհամատեղելի է:
2. *Մեկ լիտերալ դիզյունկտների կանոն՝* եթե -ում գոյություն ունի մեկ լիտերալ պարունակող հիմնական դիզյունկտ , ապա -ը ստացվում է -ից՝ ջնջելով այն հիմնական դիզյունկտները, որոնք պարունակում են : Եթե -ը դատարկ է, ապա -ը համատեղելի է: Հակառակ դեպքում, կառուցում ենք -ը՝ -ից ջնջելով -ի մուտքերը: -ը անհամատեղելի է, այն և միայն այն դեպքում, եթե -ը նույնպես անհամատեղելի է: Նշենք, որ եթե -ը մեկ լիտերալ հիմնական դիզյունկտ է, ապա այն ջնջելիս կվերածվի -ի։
3. *Մաքուր լիտերալների կանոն`* -ի հիմնական դիզյունկտում գտնվող լիտերալը կոչվում է *մաքուր*-ում, այն և միայն այն դեպում, եթե -ը չի հանդիպում -ի որևէ հիմնական դիզյունկտում: Եթե -ը մաքուր լիտերալ է, ապա ջնջում ենք բոլոր հիմնական դիզյունկտները, որոնք պարունակում են ։ Մնացած բազմությունը անհամատեղելի է, այն և միայն այն դեպքում, եթե-ը անհամատեղելի է:
4. *Տրոհման կանոն`* եթե  բազմությունը կարելի է ներկայացնել հետևյալ տեսքով՝ , որտեղ -ի​ն և -ը ազատ են -ից և -ից, ապա ստանում ենք երկու բազմություն՝  *և* , -ը անհամատեղելի է, այն և միայն այն դեպքում, երբ *-*ը անհամատեղելի է, այսինքն՝ և -ը, և ​-ը անհամատեղելի են:

Վերոհիշյալ կանոնները շատ կարևոր են: Հաջորդիվ կտեսնենք, որ այս կանոններն ունեն ավելի լայն կիրառություն: Բերենք օրինակներ՝ այս կանոնների օգտագործումը ցույց տալու համար:

Օրինակ՝ ցույց տանք, որ -ը անհամատեղելի է:

Քանի, որ վերջնական բանաձևը պարունակում է դատարկ դիզյունկտ , ապա -ը անհամատեղելի է:

Օրինակ՝ ցույց տանք, որ -ը համատեղելի է:

Քանի որ բաժանման երկու բազմություններն էլ համատեղելի են, ապա -ը նույնպես համատեղելի է:

Օրինակ՝ ցույց տանք, որ -ը համատեղելի է:

Այսպիսով -ը համատեղելի է։

## 1.3 Ռեզոլյուցիայի մեթոդը տրամաբանական արտահայտություններում

Ռեզոլյուցիայի մեթոդը, ըստ էության, [Դևիսի և Փաթնեմի](#_1.1_Դևիսի_և) մեկ լիտերալ դիզյունկտների կանոնի ընդհանրացումն է:

Օրինակ դիտարկենք հետևալ դիզունկտները՝

Օգտագործելով մեկ լիտերալ դիզունկտների կանոնը, -ից և -ից մենք կարող ենք ստանալ նոր դիզյունկտ

Մեկ լիտերալ դիզյունկտների կանոնը մեզ անհրաժեշտ է, որպեսզի նախ որոշենք, արդյոք կա լիտերալների հակադիր զույգ (օրինակ՝ ) -ում և (օրինակ՝  ) -ում, ապա ջնջենք այդ զույգը -ից և -ից, որպեսզի ստանանք նոր դիզյունկտ , որը -ն է:

Վերոհիշյալ կանոնը ընդհանրացնելով և այն կիրառելով դիզյունկտների ցանկացած զույգի նկատմամբ (ոչ պարտադիր միայն մեկ լիտերալ պարունակող), մենք ստանում ենք հետևյալ կանոնը, որը կանվանենք ***ռեզոլյուցիայի կանոն***:

Ցանկացած երկու դիզյունկտների համար՝ և , եթե գոյություն ունի ​լիտերալ -ում, որը հակադիր է ​ լիտերալին -ում, ապա ջնջելով ​-ը -ից և ​-ը -ից, մենք կառուցում ենք մնացած դիզյունկտների դիզյունկցիան: Ստացված դիզյունկտը կոչվում է -ի և -ի *ռեզոլվենտ*:

Օրինակ դիտարկենք հետևյալ դիզյունկտները՝

-ը պարունակում է լիտերալ, որը հակադիր է -ում գտնվող լիտերալին: Ուստի, ջնջելով -ն -ից և -ն -ից, մենք կառուցում ենք մնացած դիզյունկտների դիզյունկցիան`, ստացված ռեզոլվենտը կլինի ։

Ռեզոլվենտի կարևոր հատկությունն այն է, որ ցանկացած ռեզոլվենտ, որը ստացվում է երկու դիզյունկտներից՝  և , -ի և -ի տրամաբանական հետևանքն է: Այս հատկությունը հաստատվում է հետևյալ թեորեմով`

**Թեորեմ 1.2[1]`** Եթե տրված են երկու դիզյունկտներ՝ և , ապա -ի և -ի ռեզոլվենտը -ն -ի և -ի տրամաբանական հետևանքն է:

**Սահմանում`** Ենթադրենք՝ -ը դիզյունկտների բազմություն է: -ից -ի ռեզոլյուցիոն արտածումը դիզյունկտների վերջավոր հաջորդականություն է՝ որտեղ յուրաքանչյուր ​-ն կամ պատկանում է -ին, կամ նախորդ դիզյունկտների ռեզոլվենտն է, և : -ից (դատարկ դիզյունկտ) արտածումը կոչվում է -ի հերքում (կամ -ի անհամատեղելիության ապացույց):

Մենք ասում ենք, որ դիզյունկտը կարող է արտածվել կամ ստացվել -ից, եթե գոյություն ունի -ի արտածում -ից:

Օրինակ դիտարկենք բազմություն՝

(1)-ից և (2)-ից կարող ենք ստանալ ռեզոլվենտ՝ : (4)-ից և (3)-ից կարող ենք ստանալ ռեզոլվենտ՝ ։ Քանի, որ -ն ստացվում է -ից ռեզոլյուցիայի կանոնի կիրառմամբ, ապա համաձայն թեորեմ 1.0-ի, -ը -ի տրամաբանական հետևանքն է: Ուստի, -ը անհամատեղելի է։

## 1.4 Տեղադրություն և ունիֆիկացիա

Մենք դիտարկեցինք *ռեզոլյուցիայի մեթոդը* տրամաբանական արտահայտությունների համար: Հաջորդիվ մենք այն կտարածենք *առաջին կարգի տրամաբանության* վրա: Նշել ենք, որ ռեզոլյուցիայի կանոնի կիրառման հիմնական պահը հակադիր լիտերալների գտնելն է երկու դիզյունկտներում: Երբ դիզյունկտները չեն պարունակում փոփոխականներ, ապա դա շատ պարզ է: Սակայն, երբ դիզյունկտները պարունակում են փոփոխականներ, ապա խնդիրը բարդանում է: Օրինակի համար դիտարկենք հետևյալ դիզյունկտները՝

Չկա որևէ լիտերալ ​-ում, որը հակադիր լինի -ի որևէ լիտերալի: Սակայն, եթե մենք ​-ում x-ը փոխարինենք -ով, իսկ ​-ում x-ը փոխարինենք -ով, ապա կստանանք՝

Գիտենք, որ ​-ը և ​-ը համապատասխանաբար ​-ի և ​-ի հիմնական օրինակներն են, իսկ -ն և -ն հակադիր են միմյանց: Ուստի, -ից և ​-ից մենք կարող ենք ստանալ ռեզոլվենտ՝

Ընդհանուր դեպքում, եթե ​-ում x-ը փոխարինենք -ով, ապա կստանանք՝

Կրկին ​-ը ​-ի օրինակ է: Միևնույն ժամանակ, ​-ում -ը հակադիր է ​-ում -ին: Ուստի, մենք կարող ենք ստանալ ռեզոլվենտ ​-ից և ​-ից:

-ը ​-ի օրինակ է: Փոփոխականները ​-ում և -ում համապատասխան թերմերով փոխարինելով, ինչպես նշված է վերևում, մենք կարող ենք ստեղծել նոր դիզյունկտներ ​-ից և ​-ից: Բացի այդ, ​-ը *ամենաընդհանուր դիզյունկտն* է այն իմաստով, որ վերը նշված գործընթացով ստացված բոլոր այլ դիզյունկտները -ի օրինակներ են: ​-ը նույնպես կանվանենք -ի և ​-ի ռեզոլվենտ:

**Սահմանում`**  *տեղադրությունը* վերջավոր բազմություն է՝ , որտեղ՝ յուրաքանչյուր -ն փոփոխական է, յուրաքանչյուր -ն թերմ է, որը տարբերվում է -ից, բոլոր ​-երը տարբեր են: Եթե -ը հիմնական թերմեր են (այսինքն՝ չեն պարունակում փոփոխականներ), ապա տեղադրությունը կոչվում է հիմնական տեղադրություն: Տեղադրությունը, որը չի պարունակում որևէ տարր, կոչվում է դատարկ տեղադրություն և նշանակվում է -ով: Տեղադրությունը գրելու համար մենք կօգտագործենք հունարեն տառեր (օրինակ՝ ):

Օրինակ հետևալ երկու բազմությունները հանդիսանում են տեղադրություն՝

**Սահմանում`** ենթադրենք -ը տեղադրություն է, և -ն արտահայտություն է: Այդ դեպքում -ն արտահայտություն է, որը ստացվում է -ից՝ -ում *​-*ի բոլոր հանդիպումները միաժամանակ փոխարինելով -ով: -ն կոչվում է -ի *օրինակ*: (Նշենք, որ օրինակի այս սահմանումը համատեղելի է գլուխ 4-ում տրված դիզյունկտի հիմնական օրինակի սահմանման հետ:)

Օրինակ՝ ենթադրենք և: Այդ դեպքում։

**Սահմանում`** ենթադրենք և երկու տեղադրություններ են: Այդ դեպքում -ի և -ի *կոմպոզիցիան* (նշանակում ենք ) այն տեղադրությունն է, որը ստացվում է հետևալ բազմությունից՝

Ջնջելով բոլոր այն -երը որոնց համար , և բոլոր -երը որոնց համար (այսինքն՝  ​-ն արդեն առկա է -ում)։

Օրինակ ՝ ենթադրենք

Այդ դեպքում՝ ։ Սակայն, քանի որ (այսինքն ), ապա պետք է հեռացնել բազմությունից։ Բացի այդ, քանի որ -ը և ​-ը առկա են -ում, ապա ​-ը և ​-ը (այսինքն՝  -ը և -ը) նույնպես պետք է հեռացվեն: Այսպիսով, մենք ստանում ենք՝

**Սահմանում`** տեղադրություն -ն կոչվում է ունիֆիկատոր բազմության համար, այն և միայն այն դեպքում, երբ ։ Ասում են, որ բազմությունը *ունիֆիկացվող* է, եթե բազմության համար գոյություն ունի ունիֆիկատոր:

**Սահմանում`** ունիֆիկատոր -ն բազմության համար կոչվում է ամենաընդհանուր ունիֆիկատոր, այն և միայն այն դեպքում, երբ ցանկացած այլ ունիֆիկատոր -ի համար գոյություն ունի տեղադրություն , այնպես որ՝ ։

Օրինակ՝ բազմությունը ունիֆիկացվող է քանի, որ հանդիսանում է ունիֆիկատոր նրա համար։

## 1.5 Ունիֆիկացիայի ալգորիթմ

Այս պարբերությունում կներկայացնենք ունիֆիկացման ալգորիթմ, որը թույլ է տալիս գտնել ամենաընդհանուր ունիֆիկատորը վերջավոր ունիֆիկացվող բազմության համար: Եթե բազմությունը չի ունիֆիկացվում, ալգորիթմը կհայտնաբերի նաև այդ փաստը:

**Սահմանում`** ոչ դատարկ արտահայտությունների բազմության -ի անհամապատասխանությունների բազմությունը ստացվում է գտնելով առաջին (ձախից) դիրքը, որտեղ -ի բոլոր արտահայտությունները չունեն նույն սիմվոլը, այնուհետև յուրաքանչյուր արտահայտությունից դուրս գրելով այն ենթաարտահայտությունը, որը սկսվում է այդ դիրքում գտնվող սիմվոլից։ Այս ենթաարտահայտությունների բազմությունը կոչվում է -ի *անհամապատասխանությունների բազմություն*:

Օրինակ՝ եթե -ն հետևյալ բազմությունն է՝ , ապա առաջին դիրքը, որտեղ -ի բոլոր արտահայտությունները չունեն նույն սիմվոլը, հինգերորդ դիրքն է, քանի որ բոլոր արտահայտությունները ունեն նույն առաջին չորս սիմվոլները՝ : Այսպիսով, անհամապատասխանությունների բազմությունը բաղկացած է համապատասխան ենթաարտահայտություններից, որոնք սկսվում են հինգերորդ դիրքից, և դա հետևյալ բազմությունն է՝ ։

*Ունիֆիկացման ալգորիթմ`*

*Քայլ* 1. Բազմություններ ։

*Քայլ* 2. Եթե -ն միալիտերալ դիզյունկտ է, ապա -ն -ի *ամենաընդհանուր ունիֆիկատորն* է: Հակառակ դեպքում, գտնել -ի անհամապատասխանությունների բազմությունը​:

*Քայլ* 3. Եթե -ում գոյություն ունեն և տարրեր, այնպիսին որ -ն փոփոխական է, որն չի հայտնվում -ում, ապա անցնել քայլ 4-ին: Հակառակ դեպքում, ավարտել՝ -ն չի ունիֆիկացվում:

*Քայլ* 4. Սահմանել և : (Նշենք, որ )։

*Քայլ* 5. -ին վերագրել  արժեքը և անցնել քայլ 2-ին:

Օրինակ՝ գտնել ամենաընդհանուր ունիֆիկատորը

1. և ։ Քանի, որ ​-ն միալիտերալ դիզյունկտ չէ, ուստի ​-ն -ի ամենաընդհանուր ունիֆիկատոր չէ:
2. Անհամապատասխանությունների բազմությունը՝ : -ում գոյություն ունի փոփոխական : որը չի հանդիպում *-*ում։
3. Սահմանենք՝

,

1. ​-ը միալիտերալ դիզյունկտ չէ, քանի որ գտնվել է անհամապատասխանությունների բազմություն -ի համար։
2. -ից կգտնենք և *։*
3. Սահմանենք՝

,

1. -ը միալիտերալ դիզյունկտ չէ, քանի որ գտնվել է անհամապատասխանությունների բազմություն -ի համար։ : -ից կգտնենք և :
2. Սահմանենք՝

,

1. Քանի, որ​-ը միալիտերալ դիզյունկտ է, ապա -ն -ի ամենաընդհանուր ունիֆիկատորն է:

**Թեորեմ 1.3[1]`** (Ունիֆիկացման թեորեմ)՝ եթե-ն վերջավոր ոչ դատարկ ունիֆիկացվող արտահայտությունների բազմություն է, ապա ունիֆիկացման ալգորիթմը միշտ կավարտվի քայլ 2-ում, և վերջին -ն կլինի -ի ամենաընդհանուր ունիֆիկատորը:

## 1.6 Ռեզոլյուցիայի մեթոդը առաջին կարգի պրեդիկատային համակարգերում

Նախորդ պարբերությունում ներկայացված ունիֆիկացման ալգորիթմի շնորհիվ մենք կարող ենք այժմ դիտարկել առաջին կարգի տրամաբանության համար ռեզոլյուցիայի մեթոդը:

**Սահմանում`** եթե դիզյունկտ -ի երկու կամ ավելի լիտերալներ (նույն նշանով) ունեն ամենաընդհանուր ունիֆիկատոր , ապա -ն կոչվում է -ի *սոսնձում*: Եթե -ն միալիտերալ դիզյունկտ է, ապա սոսնձումը կոչվում է *միալիտերալ սոսնձում*:

Օրինակ՝ ենթադրենք : Այդ դեպքում առաջին և երկրորդ լիտերալները (ընդգծված) ունեն ամենաընդհանուր ունիֆիկատոր : Ուստի, -ը -ի սոսնձումն է:

**Սահմանում`** եթե ​-ը և ​-ը երկու դիզյունկտներ են (կոչվում են դիզյունկտ-նախադրյալներ), որոնք չունեն ընդհանուր փոփոխականներ: Թող ​-ը և ​-ը լինեն երկու լիտերալներ համապատասխանաբար ​-ում և ​-ում: Եթե -ը և ​-ն ունեն ամենաընդհանուր ունիֆիկատոր , ապա դիզյունկտը՝

կոչվում է -ի և -ի (երկուական) ռեզոլվենտ: ​-ը և ​-ը կոչվում են *հեռացվող լիտերալներ*:

Օրինակ՝ ենթադրենք , : Քանի որ -ը ներառված է և՛ ​-ում, և՛ ​-ում, մենք փոխարինում ենք ​-ում -ը -ով՝ ։ Ընտրում ենք՝ , : Քանի որ :, ապա ​-ը և ​-ն ունեն ամենաընդհանուր ունիֆիկատոր: Այսպիսով՝

Ուստի, *-*ն -ի և ​-ի երկուական ռեզոլվենտն է։ -ը և -ն հեռացվող լիտերալներն են:

**Թեորեմ 1.4[2]** (ռեզոլյուցիա մեթոդի լրիվություն)՝ բազմությունը անհամատեղելի է այն և միայն այն դեպքում, երբ գոյություն ունի դատարկ դիզյունկտի դուրս բերում -ից։

# Գլուխ 2

## 2.1 TPTP գրադարանի նկարագրություն

**TPTP** ( *Thousands of Problems for Theorem Provers* — «Հազարավոր խնդիրներ թեորեմներ ապացուցող ծրագրերի համար») միավորում է ավելի քան 10.000 փորձնական խնդիր՝ նախատեսված ավտոմատ ապացուցման (ATP) համակարգերի կատարողականությունը չափելու, համեմատելու և զարգացնելու նպատակով։ Գրադարանը ստեղծվել է 1993-ին, պահպանվում է Մայամիի համալսարանում և նոր թողարկումներով ընդլայնվում է մինչ օրս։

TPTP-ն ընդգրկում է տրամաբանական խնդիրներ տարբեր ոլորտներից (մաթեմատիկական ապացույցներ, ծրագրա­վորման լեզուների վերլուծություն, արհեստական բանականություն, խմբերով տեսություն և այլն) և տարբեր բարդության մակարդակների՝ ապահովելով համաչափ «դյուրին → չափավոր → դժվար» սանդղակ։ Յուրաքանչյուր ֆայլ պարունակում է՝

* **մետատվյալներ** - եզակի կարճ անուն, թեմատիկ դաս, դժվարության գնահատական, աղբյուր,
* **խնդրի ձևակերպում** ընտրված TPTP ֆորմատներից մեկով,
* **աքսիոմների** և **հետազոտվող վարկածի** բաժանում։

TPTP լեզուն Prolog-ի սինթաքսի ընդլայնում է։ Նախադասությունները գրվում են *annotated formula* տեսքով՝ fof(<անուն>, <դեր>, <բանաձև>), cnf(<անուն>, <դեր>, <դիզյունկտ>), tff / thf և այլն։

* **CNF** (clause normal form) - կոնյունկտիվ նորմալ ձև,
* **FOF** (first-order form) - չտիպավորված առաջին կարգ,
* **TFF** (typed first-order form) - տիպավորված առաջին կարգ,
* **TXF** (extended first-order form) - ընդլայնված առաջին կարգ՝ ներկառուցված թվաբանական տիպերով և ֆունկցիաներով։

Յուրաքանչյուր տողի *role* դաշտը (axiom, conjecture, lemma, definition…) հնարավորություն է տալիս թեստային հավաքածուն ճշգրիտ բաժանել աքսիոմների և ապացուցելի թեզի, ինչը պարտադիր է ռեզոլյուցիոն ATP-ների համար։

Մենք ծրագրի տվյալների բազա ստեղծելու համար օգտվելու ենք այս գրադարանի աքսիոմների հավաքածուից և հիմնականում CNF,FOF ֆորմատներից։

## 2.2 Vampire ATP համակարգի նկարագրություն

**Vampire-ը** (Vampire Automated Theorem Prover) ավտոմատ թեորեմներ ապացուցող (ATP) առաջատար համակարգերից մեկն է, որը մշակվել է Մանչեստերի համալսարանում։ Այն հատկապես հայտնի է իր բարձր արդյունավետությամբ առաջին կարգի տրամաբանության (first-order logic, FOL) և ավելի բարդ ֆորմալ համակարգերի համար։ Vampire-ն օգտագործում է ռեզոլյուցիայի (resolution), սուպերպոզիցիայի (superposition) և այլ ժամանակակից մեթոդներ՝ թեորեմների ապացուցման համար, ինչը հնարավորություն է տալիս աշխատել ինչպես մաթեմատիկական, այնպես էլ հաշվողական տրամաբանության բարդ խնդիրների հետ։

Այն աշխատում է TPTP-ի բոլոր հիմնական ֆորմատներով, այսինք խնդիրը հնարավոր է ծրագիր մուտքագրել այդ ֆորմատով։ Ծրագիրը հենվում է ժամանակակից ապացուցման ռազմավարությունների վրա, որոնք հատուկ կիրառելի են TPTP-ում ներկայացված խնդիրների համար։ Այն բազմիցս ճանաչվել է ամենաարդյունավետ ATP համակարգերից մեկը CASC (Conference on Automated Deduction) մրցույթներում։

Մենք կօգտագործենք այս ծրագիրը գեներացված տվյալների բազայի խնդիրները լուծելու համար և խնդիրները կոնյունկտիվ նորմալ ձևի բերելու համար։

**Vampire‑ը Docker միջավայրում**

Վերոնշյալ առավելությունները ամբողջությամբ պահպանելու և փորձերի վերարտադրելիությունն ապահովելու նպատակով մենք Vampire ATP‑ն գործարկում ենք Docker կոնտեյների մեջ։

Vampire-ի կոդը ներբեռնել ենք GitHub-ի պաշտոնական պահոցից, իսկ կազմավորումը կատարել՝ **CMake** կառուցման համակարգի միջոցով։ CMake-ը գործում է որպես «մետա-բիլդ» փուլային գործիք, որն ապահովում է՝

* կոդի փաթեթի ավտոմատ սկանավորում և աղբյուրների փոխկախվածությունների հայտնաբերում,
* պլատֆորմից անկախ կառավարման ֆայլերի ստեղծում (Unix-ում՝ Makefile, Windows-ում՝ Visual Studio solution և այլն),
* տարբեր կազմավորման պրոֆիլների (Debug/Release, CPU ֆլագների ընտրություն, ստատիկ/դինամիկ լինկեր) կառավարում մեկ կոնֆիգուրացիայից։

Այսպիսով, մեծ մասշտաբի նախագծերի (ինչպիսին է Vampire-ը) համար CMake-ը հեշտացնում է բազմապլատֆորմային կոդի արագ և վերահսկելի կոմպիլյացիան, միասնական կառուցման գործընթացը պարզեցված է մինչև մեկ հրամանի (cmake .. && make -j) գործարկմամբ։

Docker‑ը բաց կոդով կոնտեյներացման հարթակ է, որը թույլ է տալիս փաթեթավորել հավելվածն իր բոլոր կախվածություններով մեկ ճկուն image‑ի մեջ։

Առավելությունները՝

* *Վերարտադրելիություն* - նույն Docker image-ը գործարկվում է նույն կերպ ցանկացած օպերացիոն համակարգում (Linux/macOS/Windows): Չի պահանջում կախվածությունների ձեռքով տեղադրում կամ ժամանակատար կարգավորումներ:
* *Մեկուսացում* - Vampire-ի տարբեր տարբերակները կարող են աշխատել զուգահեռ՝ առանց փոխադարձ բախումների: Այլ ATP-ների հետ համատեղելիություն՝ առանց գրադարանների վերսիաների խնդիրների:
* *Ինտեգրում աշխատային գործընթացում* - Docker կոնտեյներները թույլ են տալիս ճշգրիտ սահմանել CPU, RAM և այլ ռեսուրսների սահմանաչափեր՝ համատեղելի GNN ուսուցման pipelines-ի և CI/CD գործիքների հետ: Ապահովում է ավտոմատացված և մասշտաբավորվող լուծում, որը հարմար է մեծածավալ փորձերի համար:

Docker-ացված Vampire-ը վերածվում է «սև արկղի» լուծման համակարգի, որը պահանջում է միայն TPTP/CNF ֆայլերի տեղադրության ուղիների նշում։ Ամբողջ գործընթացը՝ սկսած ռեսուրսների կառավարումից մինչև աշխատանքային արդյունքների պահպանում, կարգավորվում է մեկ մեկնարկային հրամանով։

Այսպիսով, Vampire-ի Docker-ացված տարբերակը թույլ է տալիս արագ անցնել «սինթետիկ խնդիրների» ավտոմատ ստեղծումից անմիջապես ATP-ով լուծմանը, պահպանելով փորձերի ճշգրիտ վերահսկելիությունը և տասնյակ հազարավոր օրինակների պարամետրերի միասնականությունը։

## 2.3 GNN մոդելի նկարագրություն

Գրաֆային նեյրոնային Ցանցերը (GNN-ները) նեյրոնային մոդելներ են, որոնք աշխատում են գրաֆային կառուցվածքով տվյալների վրա՝ օգտագործելով օբյեկտների (հանգույցների) միջև կապերը (կողերը) ուսուցման համար: Վերջին տասնամյակում առաջացել են բազմաթիվ GNN ճարտարապետություններ՝ տարբեր նախագծային լուծումներով, հաղորդագրությունների փոխանցման մեխանիզմներով և ուսուցման մեթոդներով: Չկա մեկ «լավագույն» ճարտարապետություն, օպտիմալ ընտրությունը կախված է գրաֆի բնութագրերից, առաջադրանքի պահանջներից և հաշվարկային սահմանափակումներից: Հիմնական օրինակներն են Գրաֆային Կոնվուլյացիոն Ցանցերը (GCN), Գրաֆային Ուշադրության Ցանցերը (GAT), GraphSAGE-ը և Գրաֆային Իզոմորֆիզմի Ցանցերը (GIN) և այլն:

**Հաղորդագրությունների Փոխանցումը GNN-ներում, ընդհանուր ֆրեյմորք**

Ժամանակակից GNN-ների մեծ մասը կարելի է նկարագրել հաղորդագրությունների փոխանցման ֆրեյմորքով։ GNN-ի յուրաքանչյուր շերտում հանգույցները տեղեկատվություն են փոխանակում իրենց հարևանների հետ և թարմացնում իրենց սեփական ներկայացումները (հատկանիշային վեկտորները): Սա սովորաբար ներառում է երեք հիմնական փուլ՝

1. *Հաղորդագրության հաշվարկ (Message computation)* - յուրաքանչյուր հանգույց «հավաքում է» հաղորդագրություններ իր հարևաններից՝ սովորաբար օգտագործելով հարևանի հատկանիշային վեկտորները (հնարավոր է՝ վերափոխված ձևով)։ Ֆորմալ առումով, յուրաքանչյուր կողի համար (𝑗 → 𝑖), հաղորդագրություն  հաշվարկվում է հարևան 𝑗-ի հատկանիշներից (և հնարավոր է նաև 𝑖-ի կամ կողի բնութագրերից)։ Օրինակ, պարզ հաղորդագրություն կարող է լինել ուղղակի (հարևանի ընթացիկ ներկայացումը), կամ ավելի բարդ ֆունկցիա՝ , որը ներառում է նաև կողի բնութագրերը -ին:
2. *Ագրեգացում (Aggregation)* - հանգույց 𝑖-ի բոլոր հարևաններից ստացված հաղորդագրությունները միավորվում են մեկ համակցված հաղորդագրության մեջ: Ագրեգացումը պետք է լինի փոխակերպումներին անփոփոխ (կարգից անկախ)։ Տարածված տարբերակներն են՝ գումարում, միջինացում, առավելագույն արժեքի ընտրություն։ Օրինակ՝ կարելի է գումարել բոլոր հարևանների հաղորդագրությունները , կամ հաշվել դրանց միջինը: Արդյունքում ստացվում է հանգույց 𝑖-ի համար միավորված հարևանության տեղեկատվություն:
3. *Թարմացում (Update)* - ագրեգացված հաղորդագրությունն այնուհետև օգտագործվում է հանգույցի սեփական ներկայացումը (embedding) թարմացնելու համար։ Սովորաբար դա արվում է՝ միավորելով այն հանգույցի ընթացիկ ներկայացման հետ նեյրոնային ցանցի միջոցով (թարմացման ֆունկցիա)։ Օրինակ, թարմացումը կարող է լինել պարզ պերսեպտրոն՝ կամ ավելի բարդ ֆունկցիա, ինչպիսին է gated recurrent unit-ը (GRU): Որտեղ, σ-ն ոչ գծային ակտիվացիայի ֆունկցիա է (օր. ReLU, sigmoid), ‖-ը նշանակում է վեկտորների կոնկատենացիա (միավորում), 𝑊-ը ուսուցվող կշիռների մատրիցն է։ Որոշ ձևակերպումներում թարմացումը գրվում է հետևյալ կերպ՝ ։ Այստեղ *COMBINE*-ը սովորաբար նեյրոնային ցանց է, որը միավորում է հանգույցի նախկին վիճակն ու հարևաններից ստացված ագրեգացված տեղեկատվությունը։

Մի քանի հաղորդագրությունների փոխանցման շերտեր (layers) կիրառելով՝ GNN-ը թույլ է տալիս, որ յուրաքանչյուր հանգույց աստիճանաբար ներառի տեղեկատվություն գրաֆի ավելի հեռավոր հատվածներից (հանգույցի k հարևանները k շերտերից հետո): Այս ֆրեյմորքը ունիվերսալ է, կոնկրետ ճարտարապետությունները հիմնականում տարբերվում են՝ ինչպես են հաշվարկվում և ագրեգացվում հաղորդագրությունները, ինչպես է կատարվում թարմացումը։ Ստորև մենք կքննարկենք հիմնական GNN տարբերակները՝ ըստ այս չափանիշների, ինչպես նաև դրանց ուսուցման մեթոդներն ու գործնական դիտարկումները:

**Գրաֆային Կոնվոլյուցիոն Ցանցեր (GCN)**

GCN-ները կիրառում են շերտեր, որոնք յուրաքանչյուր հանգույցի հատկանիշները միջինացնում են իր 1-հոպ հարևանների հատկանիշների հետ (ներառյալ self-loop-երը)՝ կիրառելով ուսուցվող գծային փոխակերպում: Ֆորմալ բանաձև` , որտեղ՝ -ին և -ն աստիճանների մատրիցն է (degree matrix): Այս գործողությունը գրաֆի վրա կիրառում է սպեկտրալ ցածրահաճախային ֆիլտր, որը հարթեցնում է ազդանշանները՝ հարևան հանգույցների հատկանիշները համադրելով:

Ուժեղ կողմերը՝

* *Պարզություն և արագություն* - յուրաքանչյուր շերտ հիմնված է հազվադեպ (sparse) մատրից-վեկտոր բազմապատկման վրա, 2-3 շերտերը սովորաբար բավարար են համասեռ (homophilous) տվյալների համար, ինչպիսիք են՝ մեջբերումների գրաֆները (citation networks), սոցիալական ցանցերը։
* *Հուսալի բազային մոդել* - չնայած իր հասակին (առաջարկվել է 2016թ.), GCN-ը հաճախ գերազանցում է ավելի բարդ մոդելներին չափավոր չափի հանգույցների դասակարգման (node classification) խնդիրներում, երբ գրաֆը ունի հստակ համասեռություն։

Սահմանափակումները`

* *Միատեսակ հարևանների կշռում*- բոլոր հարևանները հավասարապես են ազդում հանգույցի վրա (բացի աստիճանի նորմալացումից):
* *Բացակայում են ներկառուցված կողերի հատկանիշներ կամ կապի տեսակներ* - հետերոգեն գրաֆների համար անհրաժեշտ է լրացուցիչ ինժիներինգ:
* *Հիպերհարթեցում (over-smoothing)* - շատ շերտեր (>4) հանգեցնում են հանգույցների ներկայացումների միանմանության (ձգտում են դեպի նույն ենթատարածությունը):

GCN մոդելը հիմնականում օգտագործվում է, երբ գրաֆը փոքրից միջին չափի է, կողերի հատկանիշները հետաքրքիր չեն, մեկնաբանելիությունը առաջնային նշանակություն չունի:

**GraphSAGE (Նմուշառում և Ագրեգացում)**

GraphSAGE-ը լուծում է GCN-ի կողմից բաց թողնված երկու գործնական խնդիր՝ մասշտաբայնությունն ու ինդուկտիվ ընդհանրացումը: Յուրաքանչյուր շերտում այն ընտրում է հարևանների ֆիքսված քանակի ենթաբազմություն յուրաքանչյուր թիրախային հանգույցի համար և կիրառում է դասավորությունից անկախ ագրեգատոր (միջին, առավելագույն ագրեգացում կամ LSTM)՝ այդ հատկանիշների վրա: Հանգույցի սեփական ներկայացումը (embedding) այնուհետև միավորվում է ագրեգացված վեկտորի հետ և անցնում ուսուցվող փոխակերպման միջով:

Սահմանափակելով հարևանության չափը՝ մինի-խմբաքանակի հիշողությունն ու հաշվարկները մնում են սահմանափակված, նույնիսկ միլիոնավոր հանգույցներ ունեցող գրաֆերի դեպքում։ Ստոխաստիկությունը նաև հանդես է գալիս որպես կանոնակարգում։

Ուժեղ կողմերը՝

* *Վեբ-մասշտաբի ուսուցում* - օգտագործվում է արդյունաբերական համակարգերում, ինչպիսին է Pinterest-ի PinSAGE-ը՝ միլիարդավոր կողերով գրաֆերի վրա ուսուցման համար:
* *Ինդուկտիվություն* - սովորած պարամետրերը ֆունկցիաներ են, ոչ թե հանգույց-հատուկ վեկտորներ. դրանք ընդհանրացնում են անտեսանելի հանգույցների կամ նույնիսկ նոր գրաֆերի համար կանխատեսման փուլում:

Սահմանափակումները`

* *Տեղեկատվության հնարավոր կորուստ* - կարևոր հարևաններ կարող են բաց թողնվել, եթե նմուշի չափը չափազանց փոքր է:
* *Հիպերպարամետրերի ճշտում* - պետք է սահմանել՝ fan-out (հարևանների քանակը) յուրաքանչյուր շերտի համար, ագրեգատորի տեսակը, խորությունը (շերտերի քանակը), բացասական նմուշառման (negative-sampling) ռազմավարությունը:
* *Կողերի հատկանիշները դեռ պահանջում են հատուկ հաղորդագրության ֆունկցիաներ:*

GraphSAGE-ը սովորաբար առաջին ընտրությունն է՝ հսկայական կամ զարգացող գրաֆերի համար և այն իրավիճակների համար, որտեղ անհրաժեշտ են cold-start հանգույցների ներկայացումներ (embeddings):

**Գրաֆային Ուշադրության Ցանցեր (GAT)**

GAT-ը փոխարինում է միատեսակ միջինացումը ինքնա-ուշադրության (self-attention) մեխանիզմով։ Յուրաքանչյուր կողի համար սահմանվում է ուսուցվող կշիռ՝

Հարևանների հաղորդագրությունները դառնում են կշռված գումարներ՝

Բազմագլուխ ուշադրությունը (multi-head) կայունացնում է ուսուցումը՝ միավորելով մի քանի անկախ գլուխերի արդյունքները (միջինացում կամ concatenation):

Ուժեղ կողմերը՝

* *Անիզոտրոպ ագրեգացում* - մոդելը կարող է անտեսել աղմկոտ հարևաններին կամ ընդգծել ազդեցիկներին։
* *Մեկնաբանելիություն* – սովորած կշիռները ցույց են տալիս, թե որ կողերն են ազդել կանխատեսման վրա:
* *Աշխատում է հերերոֆիլիայի պայմաններում* - քանի որ կշիռները կախված են հատկանիշների նմանությունից, GAT-ը չի պահանջում, որ կապված հանգույցները ունենան նույն պիտակը:

Սահմանափակումները՝

* *Ծախսեր* - ուշադրության գնահատականները հաշվարկվում են յուրաքանչյուր նմուշառված կողի համար։ Հիշողության և ուշացման պահանջները մեծանում են հարևանների քանակի և ուշադրության գլուխների թվի հետ:
* *Մասշտաբավորման հնարքներ են պահանջվում* - արտադրական համակարգերում հաճախ համատեղում են GAT-ը GraphSAGE-ի ոճի հարևանների նմուշառման հետ:
* *Կողերի հատկանիշներն ըստ դիզայնի բացակայում են* - գոյություն ունեն ընդլայնումներ, ինչպիսին է edge-conditioned attention մեխանիզմը:

GAT-ը ընտրվում է երբ, հարևանների կարևորությունը խիստ անհավասար է, կարող ենք թույլ տալ հավելյալ հաշվարկային ծախսեր (ուշադրության մեխանիզմը ավելի ծանր է, քան GCN/GraphSAGE-ը) կամ արդեն կիրառում եք հարևանների նմուշառում (օգտագործելով միայն կարևոր հարևաններ՝ հաշվարկները օպտիմիզացնելու համար):

## 2.4 TPTP գրադարանի աքսիոմների օգտագործում

Մեր մոդելի ուսուցման համար տվյալների բազայի ստեղծումը սկսվում է **TPTP** գրադարանի **Axioms** պանակից ստանդարտ աքսիոմատիկ ֆայլերի (.ax) ընտրությամբ: Այս ֆայլերը հանդիսանում են անփոփոխ գիտելիքի բազա, որոնք Vampire ATP համակարգի միջոցով վերափոխվում են մեքենայական մշակման համար օպտիմալ ձևաչափի:

Ընտրված աքսիոմների բազան պատահականորեն բաժանվում է երկու մասի՝

* Ուսուցման համար (80%)՝ մոդելի վերապատրաստման նպատակով,
* Ստուգման համար (20%)՝ մոդելի արդյունավետությունը գնահատելու համար:

Այս բաժանումն ապահովում է մոդելի կատարողականության օբյեկտիվ գնահատում:

Յուրաքանչյուր *.ax* ֆայլ փոխանցվում է Vampire-ի **Clausify** ռեժիմին՝ հետևյալ հրամանի օգնությամբ՝

vampire --mode clausify --input problem.ax --output problem\_ax\_claused.txt

Այս գործընթացում առաջին կարգի տրամաբանության (FOF/TFF) արտահայտությունները ավտոմատ կերպով փոխակերպվում են *Կոնյունկտիվ Նորմալ Ձևի* (CNF), որը հանդիսանում է մեր մոդելի հիմնական մուտքային ձևաչափը: Ստացված *problem\_ax\_claused.txt* ֆայլերը պահպանվում են *Axioms\_clausified* պանակում՝ որպես տվյալների բազայի անփոփոխ հիմք:

## 2.5 Սինթետիկ տվյալների բազայի գեներացման մեթոդաբանություն

Հետազոտական աշխատանքում կիրառվում է «Forward Proposer» ալգորիթմը սինթետիկ թեորեմների ստեղծման համար, հետևյալ մոտեցմամբ՝

**1. Նախնական տվյալների պատրաստում**

* Ընտրվում է TPTP գրադարանի 10 հիմնական տիրույթներից (դաշտերի տեսություն, երկրաչափություն, խմբերի տեսություն և այլն) մեկի որևէ աքսիոմների բազա պարունակող *.ax* ֆայլ։
* Բոլոր աքսիոմները փոխակերպվում են կոնյունկտիվ նորմալ ձևի (CNF)՝ օգտագործելով Vampire ATP համակարգը։

**2. Գծային ռեզոլյուցիայի կիրառում**

* Ենթադրենք -ը դիզյունկտների հաջորդականություն Է։
* Գործընթացը սկսվում է դիզյունկտից, որը պատահականորեն ընտրվում է աքսիոմների բազայից։
* Յուրաքանչյուր քայլի համար՝
  + Վերցվում է նախորդ դիզյունկտը։
  + Իրականացվում է ռեզոլյուցիա ցանկացած այլ դիզյունկտի հետ, որի հետ հնարավոր է այն իրականացնել։ Դիզյունկտը կարող է լինել կամ ակսիոմների ցանկից կամ արդեն ստացված շղթայի դիզյունկտների ցանկից։
  + Ստեղծվում է նոր դիզյունկտ, որը հանդիսանում է նախորդ երկու դիզյունկտների ռեզոլվենտը։
  + Եթե -ում առկա են կրկնվող կամ ունիֆիկացվող լիտերալներ, ապա թողնում ենք դրանցից միայն մեկը։

Քանի որ յուրաքանչյուր նոր դիզյունկտ պարտադիր մասնակցում է հաջորդ քայլում, ապացույցի ծառը դառնում է ուղիղ գիծ (այստեղից էլ «գծային» անվանումը): Այնուամենայնիվ, գծային ռեզոլյուցիան պահպանում է ամբողջականությունը, այսինքն՝ տեսականորեն այն կարող է հանգել ցանկացած դիզյունկտի, որին կարող է հանգել ռեզոլյուցիան:

**3. Դիզյունկտի չափի օպտիմալացում**

Միատեսակ նմուշառումը ռեզոլյուցիաների հանգեցնում է դիզյունկտի չափի արագ աճի:  Այդ պատճառով ամեն մի թույլատրելի եզրակացություն գնահատվում է ըստ ստացվող դիզյունկտի չափի (պրեդիկատների քանակով) և ընտրվում է *soft-max* բաշխմամբ՝

Որտեղ T-ն փոփոխական է, -ին եզրակացություն -ից ստացված դիզյունկտի պրեդիկատների քանակը: Որքան ցածր է T-ն, այնքան բարձր է նախապատվությունը կոմպակտ դրույթներին: Որքան բարձր է T-ն, այնքան ավելի մեծ է հետազոտության հնարավորությունը:

**4. Խնդրի ձևակերպում**

* N քայլերից հետո վերջնական դիզյունկտը դառնում է ապացուցման թեզ, որը Vampire ATP-ն պետք է լուծի ռեզոլուցիաներ անելով։
* Ստացվում է վավեր խնդիր՝ ։

**5. Պարամետրերի տեղադրում**

* Յուրաքանչյուր տիրույթի համար ընտրվում են օպտիմալ N և T արժեքներ՝
  + Գեներացվում է 1 միլիոն թեկնածու թեորեմ։
  + Չափվում է դժվարությունը Vampire ATP-ի միջոցով։
  + Մերժվում են պարամետրերը, եթե միջին դիզյունկտի չափը > 64 նիշ։
  + Պահպանվում են միայն այն տարբերակները, որոնք տալիս են ≥500,000 ունիկալ թեորեմ։
  + Ընտրվում է ամենադժվար տարբերակը սահմանված պայմաններում։

**6. Ուսուցման կորպուսի ստեղծում**

* Ընտրված պարամետրերով գեներացնում եմ տասնյակ միլիոններով սինթետիկ խնդիրներ։
* Այս մոտեցումն ապահովում է՝
  + Վավերություն (բոլոր թեորեմները ապացուցելի են կառուցվածքով)
  + Կառավարելի ուսուցման ծրագիր (N-ը վերահսկում է ապացույցի խորությունը)
  + Տիրույթի լրիվ ծածկույթ (բոլոր սիմվոլները գալիս են աքսիոմներից)

Այս մոտեցումը հնարավորություն է տալիս ստեղծել լայնածավալ և բազմաբնույթ ուսուցման տվյալներ, որոնք զգալիորեն գերազանցում են TPTP-ի խնդիրների քանակն ու բազմազանությունը։

# Գլուխ 3

## 3.1 Լիտերալների ունիֆիկացիայի և ռեզոլյուցիայի օժանդակ մոդուլ

Խնդիրներ գեներացնելու ընթացքում գործարկվում է հատուկ օժանդակ մոդուլ, որը կատարում է Ռոբինսոնի ունիֆիկացիայի դասական ալգորիթմը և դրան հաջորդող ռեզոլյուցիայի գործողությունը։

Ալգորիթմի էական քայլերը՝

* *Թերմերի տարանջատում* - յուրաքանչյուր լիտերալ տրոհվում է ֆունկցիայի անվան, արգումենտների և (եթե կա) ժխտման նշանի վրա։
* *Փոփոխականների տեղադրություն* - փոփոխական-թերմ զույգերի համար հաշվարկվում է ամենաընդհանուր ունիֆիկատոր (MGU), որը կիրառում է «ցիկլի ստուգում» (occurs-check)՝ խուսափելով անվերջ ռեկուրսիայից, միաժամանակ պահպանելով և թարմացնելով արդեն հայտնաբերված փոխարինումները։
* *Ֆունկցիոնալ թերմերի ունիֆիկացիա* - ֆունկցիոնալ տերմերը համարվում են ունիֆիկացվող, եթե ունեն նույն ֆունկցիայի անունը և արգումենտների քանակը: Այս դեպքում ունիֆիկացիան ռեկուրսիվ կիրառվում է յուրաքանչյուր համապատասխան արգումենտի զույգի վրա։
* *Ռեզոլյուցիա* - եթե լիտերալների զույգը կոնտրար են (օր. P(a) և ¬P(a) կամ տիպավորված ունիֆիկացվող տարբերակ), կոնտրար լիտերալները հեռացվում են, իսկ մնացած լիտերալները միավորվում են նոր դիզյունկտի մեջ՝ կիրառելով հայտնաբերված ունիֆիկատորը։

Սույն մոդուլը pipeline-ում խաղում է երկու դեր՝

* *Լիտերալների զույգերի ֆիլտրում* - այն արագ որոշում է, արդյոք տվյալ երկու լիտերալները ունիֆիկացվում են թե ոչ, եթե այո, ապա վերադարձնում է ամենաընդհանուր ունիֆիկատորը։
* *Նոր դիզյունկտների կառուցում* - գեներացված շղթայի յուրաքանչյուր քայլում հենց այս մեխանիզմով է ստացվում հաջորդ դիզյունկտը (ռեզոլվենտը)։

## 3.2 Սինթետիկ աքսիոմների գեներացում

Դիզյունկտների բազան ստեղծվում է ոչ միայն TPTP գրադարանի առկա աքսիոմներից, այլ նաև սինթետիկ եղանակով գեներացված աքսիոմների միջոցով։ Սինթետիկ աքսիոմների ավտոմատ գեներացիան հնարավորություն է տալիս ստեղծել բազմազան և վերահսկելի բարդության թեստային տվյալներ որոնք չեն վերաբերվում կոնկրետ ոլորտի։

Սինթետիկ աքսիոմների ստեղծման համար մշակվել է հատուկ Python մոդուլ (generate\_axioms.py), որը թույլ է տալիս՝

* Գեներացնել տարբեր քանակի պրեդիկատային սիմվոլներ, փոփոխականներ և հաստատուններ։
* Կարգավորել դիզյունկտների նվազագույն և առավելագույն երկարությունը։
* Կարգավորել պրեդիկատների և ֆունկցիաների առավելագույն արգումենտների քանակը։

Գեներատորը կառուցում է ոչ տրիվիալ դիզյունկտներ, խուսափելով՝

* Հակասություններից նույն դիզյունկտի ներսում (օրինակ՝ p(X) | ¬p(X)):
* Միևնույն լիտերալի կրկնություններից:
* Բազմաթիվ հաստատունների առկայությունից նույն դիզյունկտում, ինչը կարող է հանգեցնել անարդյունավետ ռեզոլյուցիայի:

Գեներացված աքսիոմները պահպանվում են ստանդարտ TPTP .ax ֆայլերում՝ fof(...) ֆորմատով, օրինակ՝

fof(u1, axiom, (pred1(X0) | ~pred2(X0,X1))).

fof(u2, axiom, (pred3(X2,X1) | pred2(X0,func\_f(X1)) | ~pred4(X0))).

## 3.3 Սինթետիկ խնդիրների գեներացում

**ՔԱՅԼ 1 - աքսիոմների ներբեռնում:**

Ծրագիրը բացում է տրված TPTP ֆորմատի աքսիոմների ֆայլը (օրինակ՝ CAT001.ax\_claused.txt-ի նման) և փոխանցում է օժանդակ մոդուլի առաջին ֆունկցիային։ Այն կարդում է ամբողջ բովանդակությունը որպես տեքստ, հեռացնում է %-ով սկսվող մեկնաբանությունները, ապա յուրաքանչյուր cnf(...) տրամաբանական տողը բաժանում է անուն-դեր-լիտերալներ բաղադրիչների։ Արդյունքում ստացվում է Python-ի «դիզյունկտների ցանկ», որը պահվում է հիշողության մեջ՝ հետագա քայլերում հեշտությամբ մշակելու համար։

**ՔԱՅԼ 2 - դիզյունկտային շղթայի կառուցում։**

Հիմնական սցենարի ֆունկցիան ընտրում է պատահական աքսիոմ և սկսում է հաջորդականություն կառուցել՝

* Արագորեն ստանում է բոլոր հնարավոր ռեզոլուցիոն քայլերը -ի և մնացած աքսիոմների միջև։
* Հավանականությունների հիման վրա ընտրում է մեկ «առաջնահերթ» ելք՝ ստանալով դիզյունկտը։
* Նույն գործընթացը կրկնում է մինչև հասնում է նախորոշված N երկարությանը (օրինակ՝ 10 քայլ)

Այսպես ձևավորվում է շղթան, որտեղ յուրաքանչյուր հաջորդ դիզյունկտ հանդիսանում է աքսիոմների և նախորդ ստացված դիզյունկտների տրամաբանական հետևանքը։

**ՔԱՅԼ 3 - վարկածի ժխտում։**

Վերջնական ​դիզյունկտը ստանալուց հետո այն տրոհվում է առանձին լիտերալների։ Յուրաքանչյուր լիտերալ ժխտվում է (օր. p(X)→¬p(X)), և այդ նոր, մեկական դիզյունկտները ֆայլում գրանցվում են «negated\_conjecture» դերով։ Ստացված ֆայլում այս մասը կունենա հետևյալ տեսքը՝

cnf(cn\_neg1, negated\_conjecture, (¬p(X))).

Այս քայլը անհրաժեշտ է, որպեսզի ապացուցումը կատարվի հերքման (refutation) սխեմայով՝ աքսիոմներին ավելացնելով վարկածի ժխտումն ու ցուցադրելով, որ միասնական բազմությունը անբավարարելի է։

**ՔԱՅԼ 4 - խնդրի ձևավորում TPTP ֆորմատով։**

ԿՆՁ-ի բերված աքսիոմներն ու նոր ստեղծված ժխտված վարկածը միավորվում են և դասավորվում են cnf(...) կառուցվածքով։ Արդյունքում ստացվում է լիարժեք *.p* ֆորմատի խնդիր, որը կարելի է անմիջապես փոխանցել Vampire-ին։

**ՔԱՅԼ 5 - ռեզոլյուցիայի ենթակա զույգերի ցուցակ։**

Դիզյունկտների բազմությունն արդեն պատրաստ է, և ծրագիրը հերթով ստուգում է դրանցում եղած բոլոր լիտերալ‑զույգերը։ Ամեն մի զույգ, որի լիտերալները լրացնում են իրար (օրինակ՝ P և ¬P) և անհրաժեշտության դեպքում, ունիֆիկացվում են, ավելացվում են «resolvable\_pairs» ցանկում։ Յուրաքանչյուր նման գրառում պարունակում է՝

* clauseA\_index, literalA\_index - առաջին դիզյունկտում լիտերալի դիրքը,
* clauseB\_index, literalB\_index - երկրորդ դիզյունկտում լրացուցիչ լիտերալի դիրքը

Այս ցանկը հանդիսանում է բոլոր հնարավոր «քայլերի» ամբողջական նկարագրությունը, որոնք ապացուցիչը կարող է կատարել տվյալ վիճակում։ Հետագայում այս հավաքածուն ծառայում է որպես թեկնածու դասակարգում GNN-ի համար, որտեղ մոդելը պետք է սովորի տարբերակել «լավագույն» (best\_pair) և «այլ» հնարավորությունները։

**ՔԱՅԼ 6 - պիտակավորված JSONL գրառում։**

Ամբողջ գործընթացն ամփոփվում և պահպանվում է կոմպակտ JSON տողի տեսքով՝

{"clauses":[...], "resolvable\_pairs":[...], "best\_pair":{...}}

Նման գրառումները պահվում են Res\_Pairs/...jsonl ֆայլում, հետևելով «մեկ տող = մեկ խնդիր» սկզբունքին։ Այս մոտեցումն ապահովում է՝

* **Հստակ պիտակավորում** - «best\_pair»-ը հանդիսանում է դրական օրինակ մոդելի ուսուցման համար։ «resolvable\_pairs»-ի մյուս տարրերը ծառայում են որպես բացասական/չեզոք օրինակներ։
* **Արդյունավետ մշակում** - JSONL ֆորմատը թույլ է տալիս աճող ձևաչափով աշխատել։ Համատեղելի է PyTorch/TensorFlow data loader-ների հետ առանց ամբողջ ֆայլը հիշողություն բեռնելու։
* **Ընդլայնելիություն** - նոր խնդիրների ավելացումն իրականացվում է ֆայլի վերջում նոր տող ավելացնելով։ Պահպանվում է տվյալների ամբողջականությունն ու կառուցվածքը։

Քանի որ տվյալների գեներացման պահին դեռևս անհայտ է, թե որ լիտերալ‑զույգը կհանգեցնի արդյունավետ ռեզոլյուցիայի, «best\_pair» դաշտը սկզբում ստեղծվում է դատարկ ({}) / null արժեքով։

**ՔԱՅԼ 7 - TPTP պատճենի պահպանում։**

Նույն խնդիրը պահպանվում է նաև առանձին՝ TPTP ֆորմատով, որպեսզի ցանկացած պահի հնարավոր լինի վերահաստատել ապացուցման գործընթացը և տվյալների ամբողջականությունը։ Պահպանվում է Gen\_Problems/ պանակում որպես *.p* ֆայլ։

**ՔԱՅԼ 8 - բազմակի գեներացիա։**

Գլխավոր ցիկլը (for k in range(num\_examples)) համակարգված կերպով նորից անցնում է ՔԱՅԼ 1 → ՔԱՅԼ 7 ճանապարհը՝ յուրաքանչյուր կրկնության համար փոփոխելով՝

* *Շղթայի երկարություն (N)* - փոփոխական ապացուցման խորություն
* *Ջերմաստիճան (T)* - տարբերակում է պատահական ընտրության աստիճանը
* *Աքսիոմների ենթաբազմություն* - օգտագործում է տարբեր թեմատիկ խմբեր (SET, ALG, ...)
* *Սերմի արժեք (random.seed(k))* - ապահովում է եզակիություն և կրկնությունների բացառում

Վերջնական արդյունքը հանդիսանում է՝

* Հավասարակշռված խնդիրների հավաքածու՝ պարունակելով
  + 30% հեշտ օրինակներ
  + 50% միջին դժվարության օրինակներ
  + 20% բարդ օրինակներ
* Ուսուցման օպտիմալ պայմաններ՝ ապահովելով
  + Մոդելի կայուն ուսուցում
  + Չկողմնակալված կանխատեսումներ
  + Լայն թեմատիկ ծածկույթ

Այս իտերատիվ գործընթացը հնարավորություն է տալիս ստեղծել տարբեր դժվարության մակարդակի խնդիրներ, որոնք անհրաժեշտ են մեքենայական ուսուցման մոդելի համակողմանի զարգացման համար։

**ՔԱՅԼ 9 - արդյունքների օգտագործում։**

Գեներացիայի ավարտից հետո յուրաքանչյուր օրինակ ստացվում է երկու զուգահեռ ձևաչափով՝

* *TPTP/CNF փաթեթ* - CNF ֆորմատով աքսիոմներ և ժխտված վարկածներ՝ պատրաստ Vampire-ին փոխանցելու համար։
* *JSONL պիտակավորված ֆայլեր* - պատրոաստ GNN մոդելի supervised ուսուցման, fine‑tune կամ վերաորակավորման համար։

## 3.4 Խնդիրների լուծում և ապացույցների մշակում

Այս ենթաբաժինը ներկայացնում է pipeline-ի այն քայլը, որը ավտոմատ կերպով գործարկում է Vampire ATP-ն սինթետիկ խնդիրների հավաքածուի վրա և յուրաքանչյուր խնդրի համար ստացված ապացույցի պատասխանից դուրս է բերում ընտրված զույգը, որպես «best\_pair» լիտերալ-զույգ։ Վերջինս պիտակավորվում է տվյալների JSONL ֆայլում և հետագայում ծառայում է մոդելի ուսուցման համար։

**ՔԱՅԼ 1 - Խնդիրների լուծում։**

solve\_problems\_ATP.py սկրիպտը հնարավորություն է տալիս մեկ սեղմումով լուծել Gen\_Problems/ պանակի բոլոր .p ֆայլերը և արդյունքները պահպանել Output/ պանակում որպես <basename>\_solved.txt ֆայլեր։

Այն լուծում է խնդիրները օգտագործելով մեր ստեղծած Vampire ATP-ի Docker image-ը և գործարկում է այն, որպես Docker կոնտեյներ։ Սկրիպտի հիմնական ֆունկցիան է run\_docker\_solve\_command()-ը, որը՝

* Ստեղծում է output\_dir (եթե այն գոյություն չունի)։
* Կազմում է Docker հրամանը երկու bind‑mount թղթապանակներով (մուտքային և ելքային), որոնք վերցվում են հիմնական համակարգչից։
* Կոնտեյների ներսում for ցիկլով կանչում է՝ ./vampire --mode casc --proof\_extra full -t 100, որտեղ ընտրված --proof\_extra full ռեժիմը ապահովում է լրացուցիչ մետատվյալները ապացույցի մեջ որպեսզի հետագայում կարողանանք ապացույցից դուրս բերել Vampire ATP-ի կողմից ընտրված «best\_pair» լիտերալ-զույգերը։

**ՔԱՅԼ 2 - Ապացույցների մշակում։**

Այս փուլում գործարկվում է extract\_literals\_from\_solution.py ֆայլը, որը՝

* Սկանավորում է ապացույցը ներքևից վերև (Vampire ATP-ում լուծման քայլերի հերթականությունն այդպիսին է), գտնում առաջին ռեզոլուցիոն քայլը։ Այդ տողում նշված է ռեզոլվենտը և այն դիզյունկտների իդենտիֆիկատորները որոնք ընտրվել էին ռեզոլուցիայի համար։ Հաջորդող տողերում գրված են նաև այդ դիզյունկտների ամբողջական տեսքը։
* Ապացույցից ստացված դիզյունկտների իդենտիֆիկատորներով գտնում է այդ դիզյունկտները Res\_Pairs/ պանակում զետեղված այդ խնդրի .jsonl տիպի ֆայլում։ Եթե համընկնում չի գտնվում, ապա կատարվում է դանդաղ O(n) տեքստային համեմատություն, որը համեմատում է դիզյունկտների ամբողջական տեքստերը։
* Քանի, որ Vampire ATP-ում նշված չէ թե դիզյունկտների կոնկրետ, որ լիտերալներն են մասնակցել ռեզոլուցիայի համար ծրագիրը ունենալով սկզբնական դիզյունկտները և ստացված ռեզոլվնտը ստուգում է թե որ լիտերալներն են բացակայում և ստանում է թե դիզյունկտների որ լիտերալներն են ունիֆիկացվել։
* Գտնում է այդ լիտերալների ինդեքսները .jsonl ֆայլում։
* Ստուգում է արդյոք .jsonl ֆայլի «resolvable\_pairs» ցանկը պարունակի տվյալ զույգը, թե ոչ։ Եթե ոչ, ապա ավելացնում է։
* «best\_pair» նշում է դուրս բերված դիզյունկտների և լիտերալների ինդեքսները, որպես լավագույն ընտրություն։

Հաշվարկային բարդությունը O(N x L) է։ Այստեղ N-ը ապացույցում գրված դիզյունկտների թիվն է, իսկ L-ը մեկ դիզյունկտի միջին լիտերալների քանակը։ Միջին ապացույց (~200 դիզյունկտ) մշակվում է < 0.1 վարկյանում։

Արդյունքում .jsonl ֆայլերում հայտնվում է «best\_pair» դաշտը, որը մատնանշում է տվյալ խնդրում լավագույն ռեզոլյուցիոն զույգը։

## 3.5 Մեքենայական ուսուցման մոդելի ուսուցում

Այս բաժինը ներկայացնում է լիտերալների ընտրության խնդրի լուծման համար գրաֆային նեյրոնային ցանցի (GNN) մոդելի ուսուցման գործընթացը։ GNN մոդելը նախատեսված է ռեզոլյուցիայի ընթացքում լավագույն լիտերալ-զույգի ընտրության համար՝ հիմնվելով դիզյունկտների կառուցվածքային հատկանիշների վրա։

**ՔԱՅԼ 1 - Լիտերալների ներկայացում։**

Մեր մոդելում յուրաքանչյուր լիտերալ ներկայացվում է թվային հատկանիշների վեկտորի տեսքով, որը ներառում է՝

* Լիտերալի նշանը (դրական/բացասական) - 1 բիթ,
* Պրեդիկատի իդենտիֆիկատորը - ամբողջ թիվ (ինդեքս),
* Լիտեալների արգումենտները (max\_args=8) - յուրաքանչյուրը կոդավորված հետևյալ կերպ՝
  + 0՝ փոփոխական (օր.՝ X, Y),
  + 1՝ հաստատուն (օր.՝ a, b),
  + 2՝ ֆունկցիոնալ թերմ (օր.՝ f(x)),
  + -1՝ լրացնող արժեք (padding), եթե արգումենտների քանակը 8-ից պակաս է,
* Պրեդիկատի ներկառուցում (embedding) - պրեդիկատի ինդեքսը փոխարինվում է 16 չափանի վեկտորի, որը ուսուցման ընտացքում արտահայտում է տվյալ պրեդիկատի իմաստաբանական առանձնահատկությունները և կապը մյուս պրեդիկատների հետ։ ։

Այս եղանակով յուրաքանչյուր լիտերալ վերածվում է 25 չափանի վեկտորի, որը համալիր ձևով ներկայացնում է դրա կառուցվածքն ու իմաստը։

**ՔԱՅԼ 2 - Գրաֆի կառուցում։**

Խնդրի գրաֆային ներկայացման համար կիրառվում է հետևյալ մոտեցումը՝

* *Գագաթներ (Vertices)* - գրաֆի յուրաքանչյուր գագաթ համապատասխանում է մեկ լիտերալի։ Գագաթին վերաբերող տվյալները դա համապատասխան լիտերալի վեկտորային ներկայացումն է։
* *Կողեր (Edges)* - գրաֆի կողերը ստեղծվում են այն լիտերալների զույգերի միջև, որոնք կարող են ռեզոլյուցիայի ենթարկվել: Կողերն ունեն երկկողմանի բնույթ, որը հեշտացնում է հաղորդագրությունների փոխանցումը գրաֆում։
* *Պիտակներ (Labels)* - յուրաքանչյուր կող ունի երկուական պիտակ՝ 1 (լավագույն զույգ) կամ 0 (ոչ լավագույն զույգ)։ Այս պիտակները վերցվում են «best\_pair» դաշտից, որը լրացվել է Vampire ATP-ի լուծումներից։

Հավաքածուի յուրաքանչյուր նմուշ փոխակերպվում է վերոնշյալ գրաֆի, որը հետո օգտագործվում է GNN մոդելի ուսուցման համար։

**ՔԱՅԼ 3 - Մոդելի ճարտարապետություն։**

Մեր ռեզոլյուցիայի համար մշակվել է հատուկ GraphSAGE հիմքով GNN մոդել։ Այն ունի հետևյալ կառուցվածքը՝

* *Հաղորդագրությունների փոխանակում (Message Passing)* - երկու SAGEConv շերտ, որոնք լիտերալների հատկանիշները տարածում են գրաֆի կողերի միջոցով։ Յուրաքանչյուր գագաթ հավաքում է տեղեկատվություն իր հարևաններից՝ ստեղծելով ավելի հարուստ ներկայացում։
* *Կողերի դասակարգում (Edge Classification)* - կողերի դասակարգման MLP (Multi-Layer Perceptron), որը վերցնում է երկու հարևան գագաթների հատկանիշների կոնկատենացիան և կանխատեսում է, թե արդյոք տվյալ կողը պետք է ընտրվի որպես ռեզոլյուցիայի լավագույն թեկնածու։

Մոդելի ներքին չափերը ներառում են՝

* Գագաթի հատկանիշների չափը՝ 25 (լիտերալի ներկայացումը)։
* Թաքնված շերտի չափը՝ 64 (հարուստ ներկայացման համար)։
* Ելքային չափը՝ 2 (երկու դաս՝ լավագույն/ոչ լավագույն)։

**ՔԱՅԼ 4 - Ուսուցման գործընթաց։**

Ուսուցումն իրականացվում է հետևյալ քայլերով՝

* *Տվյալների բաժանում* - տվյալների հավաքածուն բաժանվում է ուսուցման (80%) և թեստավորման (20%) բազմությունների՝ մոդելի ընդհանրացումը գնահատելու համար։
* *Փնջերի ձևավորում (Batching)* – գրաֆները առանձնացվում են փնջերի (batch\_size=8)՝ զուգահեռ մշակման համար, ինչը զգալիորեն արագացնում է ուսուցումը։
* *Կշռված կորուստի ֆունկցիա* - քանի որ տվյալները անհավասարակշիռ են (դրական օրինակները շատ ավելի քիչ են, քան բացասականները), օգտագործվում է կշռված խաչաձև էնտրոպիայի ֆունկցիա՝ weight=[1.0, 3.0], որը ավելի մեծ կարևորություն է տալիս դրական օրինակներին։
* *Օպտիմիզացիա* - Adam օպտիմիզատորն օգտագործվում է մոդելի պարամետրերի թարմացման համար, հիմնականում 1e-3 կամ 1e-4 ուսուցման արագությամբ (learning rate)։

**ՔԱՅԼ 5 - Checkpoint-երի պահպանում և կատարելագործում։**

Ուսուցման գործընթացի կարևոր մասն է մոդելի checkpoint-երի պահպանումը, որը թույլ է տալիս՝

* Պահպանել լավագույն մոդելը ուսուցման ընթացքում։
* Շարունակել ուսուցումը նախկինում պահպանված վիճակից։
* Կատարելագործել այն՝ նոր տվյալների վրա հիմնվելով։

Այս մոտեցումը հատկապես արդյունավետ է տարբեր թեմատիկ ոլորտների ռեզոլյուցիոն խնդիրների համար։ Օրինակ, մենք կարող ենք նախապես ուսուցանել մոդելը ընդհանուր խնդիրների վրա, ապա կատարելագործել այն կոնկրետ տիրույթի (օր.՝ թվաբանական խնդիրներ, խմբերի տեսություն, երկրաչափություն, բազմությունների տեսություն և այլն) խնդիրների համար։

**ՔԱՅԼ 6 - Մետրիկաների մոնիտորինգ և վերլուծություն։**

Ուսուցման ընթացքում մենք հետևում ենք հետևյալ մետրիկաներին՝

* *Կորուստի արժեք (Loss)* - ցույց է տալիս, թե որքան հեռու է մոդելը օպտիմալ լուծումից։
* *Ուսուցման ճշգրտություն (Training Accuracy)* - մոդելի կատարողականը ուսուցման տվյալների վրա։
* *Թեստային ճշգրտություն (Test Accuracy)* - մոդելի կատարողականը թեստային տվյալների վրա։

Մոդելը սովորաբար ուսուցանվում է 10-30 էպոխաների ընթացքում, կախված տվյալների քանակից և բարդությունից։ Վերջնական մոդելն ունի մոտ 90-93% ճշգրտություն այն տվյալների բազմության վրա, որոնք առանձնացվել էին առանձնացվել էին տվյալների բազայից և չէին ընդգրկվել ուսուցման ժամանակ՝ ցույց տալով լավ ընդհանրացում նոր, չտեսնված խնդիրների համար։

**ՔԱՅԼ 7 - Բազմապլատֆորմային համատեղելիություն։**

Մոդելը մշակվել է այնպես, որ կարողանա աշխատել տարբեր հարթակներում՝

* CPU-ի վրա՝ սահմանափակ ռեսուրսներով միջավայրերում։
* GPU-ի վրա՝ արագացված ուսուցման համար (CUDA միջոցով)։

Սա ապահովում է, որ մեքենայական ուսուցման մոդելը կարող է օգտագործվել տարբեր համակարգիչների վրա, ներառյալ աշխատակայաններ և սերվերային պլատֆորմներ։

**ՔԱՅԼ 8 - Պրեդիկատների ավտոմատ հավաքագրում։**

Նկատի ունենալով, որ տարբեր խնդիրներ կարող են պարունակել տարբեր պրեդիկատներ, մոդելի ուսուցման համակարգը ներառում է ավտոմատ պրեդիկատների հավաքագրման մեխանիզմ։ Այն՝

* Տվյալների հավաքածուից դուրս է բերում բոլոր եզակի պրեդիկատները։
* Ստեղծում է պրեդիկատ-ինդեքս համապատասխանեցման բառարան։
* Համապատասխանեցնում է այս ինդեքսները լիտերալների ներկայացման մեջ։

Այս մոտեցումը թույլ է տալիս մոդելին հարմարվել տարբեր պրեդիկատների հավաքածուների և նոր, չտեսնված պրեդիկատների հետ՝ հնարավորինս պահպանելով մոդելի ընդհանրացման հատկությունը։

## 3.6 Մոդելի թեստավորում

Այս բաժինը ներկայացնում է մոդելի արդյունավետության գնահատումը։ Թեստավորման հիմնական նպատակն է պարզել, թե որքանով է մոդելով ուղղորդված լիտերալների ընտրությունը գերազանցում ստանդարտ (կույր) մոտեցումը ապացուցման գործընթացում։

Թեստավորումը հիմնված է երկու տարբեր մոտեցումների համեմատության վրա՝

* *Կույր ռեզոլյուցիա (Brute-force resolution)* - այս մոտեցումը փորձում է բոլոր հնարավոր լիտերալների զույգերը հերթականությամբ՝ առանց որևէ մեթոդի կիրառության կամ առաջնահերթության։
* *GNN-ուղղորդված ռեզոլյուցիա (GNN-guided resolution)* - այս մոտեցումը կիրառում է մեր ուսուցանված մոդելը յուրաքանչյուր քայլում լավագույն լիտերալների զույգը ընտրելու համար։

Երկու մոտեցումների համար կիրառվում են միևնույն սահմանափակումները՝

* Առավելագույն ռեզոլվենտի չափ՝ 8 լիտերալ։
* Ժամային սահմանափակում՝ 30 վայրկյան յուրաքանչյուր խնդրի համար։
* Միևնույն թեստային խնդիրների հավաքածու։

Անհրաժեշտության դեպքում մենք կարող ենք փոփոխել այս պարամետրերը: Այս կերպ ապահովվում է համեմատության արդարությունը և հիմնավորվածությունը։

Թեստավորման տվյալները պահվում են JSONL ֆորմատով, որի կառուցվածքը նման է մոդելի ուսուցման համար օգտագործված ֆորմատին, որտեղ առկա են միայն դիզունկտները։ Թեստավորումն իրականացվել է այն խնդիրների և աքսիոմների վրա, որոնք չեն օգտագործվել մոդելի ուսուցման ընթացքում։

Հավաքածուն ընտրվում է այնպես, որ լինի բավականաչափ բազմազան և ներառի տարբեր տիպի ռեզոլյուցիայի խնդիրներ՝

* Հեշտ խնդիրներ, որոնք երկու մոտեցումներն էլ կարող են լուծել արագ։
* Միջին բարդության խնդիրներ, որտեղ լիտերալների ճիշտ ընտրությունը կարող է էականորեն ազդել արդյունավետության վրա։
* Բարդ խնդիրներ, որոնք կարող են լուծվել միայն լիտերալների խելամիտ ընտրության դեպքում։

Թեստավորման համար մշակվել է compare\_solvers.py սկրիպտը, որն իրականացնում է հետևյալ գործողությունները՝

* Նախապես ուսուցանված GNN մոդելի բեռնում։
* Թեստային տվյալների հավաքածուի ֆայլերի հերթական մշակում։
* Յուրաքանչյուր խնդրի լուծում երկու մոտեցումներով։
* Արդյունքների գրանցում և վիճակագրության հավաքում։

Մոդելի թեստավորման ժամանակ կիրառվում են հետևյալ ցուցանիշները՝

* *Ապացուցման հաջողություն (Proof success)* - արդյոք ալգորիթմը կարողանում է գտնել ապացույցը սահմանված ժամանակի ընթացքում։
* *Ռեզոլյուցիայի քայլերի քանակ (Resolution steps)* - քանի ռեզոլյուցիայի քայլ է պահանջվում ապացույցը գտնելու համար։

Թեստավորման արդյունքում ստացվում են հետևյալ հիմնական բնութագրերը՝

* *Լուծված խնդիրների քանակ* – երկու մոտեցումներով լուծված խնդիրների քանակների համեմատում։
* *Միջին քայլերի քանակ* - երկու մոտեցումներով լուծված խնդիրների լուծման ընթացում կատարված քայլերի քանակների համեմատում։
* *Հարաբերական բարելավում* - հաշվարկվում է հարաբերական բարելավումը տոկոսային հարաբերությամբ։
* *Արագագործության բարելավում* - այն խնդիրների տոկոսը, որոնցում GNN-ուղղորդված մոտեցումը պահանջում է ավելի քիչ քայլեր, քան կույր մոտեցումը։

# ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ

Հետազոտական աշխատանքները ցույց տվեցին, որ մեքենայական ուսուցումը կիրառելի է ռեզոլյուտիվ արտածման մեջ։ Մոդելի համար անհրաժեշտ տվյալների բազան կարելի է գեներացնել սինթետիկ եղանակով, պահպանելով առաջին կարգի տրամաբանության պահանջները։ Խնդրի աքսիոների ցանկը կարելի է գեներացնել սինթետիկ եղանակով կամ վերցնել TPTP գրադարանի աքսիոների ցանկից, որոնցից յուրաքաչյուրը վերաբերում է կոնկրետ ոլորտի։ Առկա աքսիոների հիման վրա ստեղծել տարբեր բարդության խնդիրներ որոնք ապացուցելի են։ Ստեղծված խնդիրները լուծել արդեն առկա ավտոմատ թեորեմների ապացուցման ծրագրի միջոցով և ստանալ օպտիմալ լուծում։ Արդյունքում ունենալով միլիոնավոր օրինակներ ուսուցանել մոդել, որը խնդրի յուրաքաչյուր քայլում կառաջարկի ռեզոլյուցիա կատարելու համար անհրաժեշտ լիտերալների օպտիմալ զույգը։

Մոդելը կարող է էականորեն բարելավել ավտոմատ թեորեմներ ապացուցող համակարգերի արդյունավետությունը, ինչպես նաև կիրառվել ֆորմալ վերիֆիկացիայի, ծրագրային ապահովման ստուգման, արհեստական բանականության տրամաբանական համակարգերի և ռոբոտատեխնիկայի մի շարք խնդիրներում:

# ՕԳՏԱԳՈՐԾՎԱԾ ԳՐԱԿԱՆՈՒԹՅԱՆ ՑԱՆԿ

1. Ч.Чень, Р.Ли, Математическая логика и автоматическое доказательство теорем, 1983 г.
2. L. Bachmair, H. Ganzinger, ”Resolution Theorem Proving”. In Handbook of Automated Reasoning, A. Robinson and A. Voronkov (Eds.), vol. pp. 19-99, MII Press, 2001.
3. <https://arxiv.org/abs/2103.03798>
4. <https://vprover.github.io/>
5. <https://www.tptp.org/>
6. <https://theaisummer.com/gnn-architectures/>
7. <https://distill.pub/2021/gnn-intro/>