

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 ԳԱՈՐԳԻ

ՄԱԳԻՍՏՐՈՍԱԿԱՆ ԹԵԶ

**ՄԵՔԵՆԱՅԱԿԱՆ ՈՒՍՈՒՑՄԱՆ ՄԵԹՈԴՆԵՐԻ ԿԻՐԱՌՈՒՄԸ
ՌԵԶՈԼՅՈՒՏԻՎ ԱՐՏԱԾՄԱՆ ՄԵԶ**

«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

«Թույլատրելի պաշտպանության»

Ամբիոնի վարիչ՝ _____
ստորագրություն

Ֆ. Մ. Գ. Թ. , դոցենտ, Սարգսյան Ս.

գիտ. աստիճան, կոչում, ազգանուն, անուն

« ____ » _____ 2025թ.

ՄԵՔԵՆԱՅԱԿԱՆ ՈՒՍՈՒՑՄԱՆ ՄԵԹՈԴՆԵՐԻ ԿԻՐԱՌՈՒՄԸ
ՌԵԶՈԼՅՈՒՏԻՎ 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
Գլուխ 1	6
1.1 Հիմնական հասկացություններ	6
1.2 Դեփոզիտ և Փաթենթի մեթոդը	7
1.3 Ռեզոլյուցիայի մեթոդը տրամաբանական արտահայտություններում	8
1.4 Տեղադրություն և ունիֆիկացիա.....	10
1.5 Ունիֆիկացիայի ալգորիթմ	12
1.6 Ռեզոլյուցիայի մեթոդը առաջին կարգի պրեդիկատային համակարգերում .	14
Գլուխ 2	16
2.1 TPTP գրադարանի նկարագրություն	16
2.2 Vampire ATP համակարգի նկարագրություն	17
2.3 GNN մոդելի նկարագրություն.....	18
2.4 TPTP գրադարանի աքսիոմների օգտագործում	22
2.5 Սինթետիկ տվյալների բազայի գեներացման մեթոդաբանություն	23
Գլուխ 3	26
3.1 Լիտերալների ունիֆիկացիայի և ռեզոլյուցիայի օժանդակ մոդուլ.....	26
3.2 Սինթետիկ աքսիոմների գեներացում	26
3.3 Սինթետիկ խնդիրների գեներացում	27
3.4 Խնդիրների լուծում և ապացույցների մշակում	30
3.5 Մեքենայական ուսուցման մոդելի ուսուցում	32
3.6 Մոդելի թեստավորում	35
ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ	37
ՕԳՏԱԳՈՐԾՎԱԾ ԳՐԱԿԱՆՈՒԹՅԱՆ ՑԱՆԿ.....	38

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 կապերներից: Կաշխատենք հետևյալ տրամաբանական նշանների հետ՝ \top (ճշմարիտ), \perp (կեղծ), \neg (Ժխտում), \vee (դիզյունակցիա), \wedge (կոնյունկցիա), \rightarrow (հետևում է), \leftrightarrow (համարժեք է) :

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

1. Հաստատունը թերմ է:
2. Փոփոխականը թերմ է:
3. Եթե f -ը n -տեղանի ֆունկցիոնալ սիմվոլ է և t_1, t_2, \dots, t_n -ը թերմեր են, ապա $f(t_1, t_2, \dots, t_n)$ -ը թերմ է:
4. Վերը նշված կանոնների կիրառմամբ ստացված թերմերից բացի այլ թերմեր չկան:

Սահմանում՝ Եթե P -ն n -տեղանի պրեդիկատային սիմվոլ է և t_1, t_2, \dots, t_n -ը թերմեր են, ապա $P(t_1, t_2, \dots, t_n)$ -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 F -ը և G -ն բանաձևեր են, ապա $\neg F$, $(F \vee G)$, $(F \wedge G)$, $(F \rightarrow G)$ և $(F \leftrightarrow G)$ նույնպես բանաձևեր են:
3. Եթե F -ը բանաձև է, իսկ x -ը ազատ փոփոխական է F -ում, ապա $(\forall x)F$ և $(\exists x)F$ նույնպես բանաձևեր են:
4. Բանաձևերը ստեղծվում են միայն 1, 2 և 3 կանոնների կիրառման վերջավոր քանակությամբ:

Սահմանում՝ F բանաձևը գտնվում է *կոնյունկտիվ նորմալ ձևում* (ԿՆԶ) այն և միայն այն դեպքում, երբ F -ը ունի $F_1 \wedge F_2 \wedge \dots \wedge F_n$ տեսք, որտեղ յուրաքանչյուր F_1, F_2, \dots, F_n հանդիսանում է լիտերալների դիզյունակցիա:

Ցանկացած բանաձև հնարավոր է փոխակերպել կոնյունկտիվ նորմալ ձևի:

Թեորեմ 1.1^[2]՝ Ենթադրենք տրված են F_1, F_2, \dots, F_n բանաձևերը և G բանաձևը: Այդ դեպքում բանաձևը հանդիսանում է F_1, F_2, \dots, F_n -ի տրամաբանական հետևանք այն և միայն այն դեպքում, երբ $F_1 \wedge F_2 \wedge \dots \wedge F_n \wedge \neg G$ -ն անհամատեղելի է:

1.2 Դևիսի և Փաթենմի մեթոդը

Ենթադրենք՝ S -ը դիզյունկոնյունկտիվ բազմություն է: Մեթոդը, ըստ էության, բաղկացած է հետևյալ չորս կանոններից՝

1. *Նույնաբանության կանոն*՝ S -ից ջնջում ենք բոլոր նույնաբանական հիմնական դիզյունկոնյունկտները: Մնացած S' բազմությունը անհամատեղելի է, այն և միայն այն դեպքում, եթե S -ը անհամատեղելի է:
2. *Մեկ լիտերալ դիզյունկոնյունկտի կանոն*՝ եթե S -ում գոյություն ունի մեկ լիտերալ պարունակող հիմնական դիզյունկոնյունկտ L , ապա S' -ը ստացվում է S -ից՝ ջնջելով այն հիմնական դիզյունկոնյունկտները, որոնք պարունակում են L : Եթե S' -ը դատարկ է, ապա S -ը համատեղելի է: Հակառակ դեպքում, կառուցում ենք S'' -ը՝ S' -ից ջնջելով $\neg L$ -ի մուտքերը: S'' -ը անհամատեղելի է, այն և միայն այն դեպքում, եթե S -ը նույնպես անհամատեղելի է: Նշենք, որ եթե $\neg L$ -ը մեկ լիտերալ հիմնական դիզյունկոնյունկտ է, ապա այն ջնջելիս կվերաժվի \square -ի:
3. *Մաքուր լիտերալների կանոն*՝ S -ի հիմնական դիզյունկոնյունկտում գտնվող L լիտերալը կոչվում է *մաքուր* S -ում, այն և միայն այն դեպքում, եթե $\neg L$ -ը չի հանդիպում S -ի որևէ հիմնական դիզյունկոնյունկտում: Եթե L -ը մաքուր լիտերալ է, ապա ջնջում ենք բոլոր հիմնական դիզյունկոնյունկտները, որոնք պարունակում են L : Մնացած S' բազմությունը անհամատեղելի է, այն և միայն այն դեպքում, եթե S -ը անհամատեղելի է:
4. *Տրոհման կանոն*՝ եթե S բազմությունը կարելի է ներկայացնել հետևյալ տեսքով՝ $(A_1 \vee L) \wedge \dots \wedge (A_m \vee L) \wedge (B_1 \vee \neg L) \wedge \dots \wedge (B_n \vee \neg L) \wedge R$, որտեղ A_i, B_i -ին և R -ը ազատ են L -ից և $\neg L$ -ից, ապա ստանում ենք երկու բազմություն՝ $S_1 = A_1 \wedge \dots \wedge A_m \wedge R$ և $S_2 = B_1 \wedge \dots \wedge B_n \wedge R$, S -ը անհամատեղելի է, այն և միայն այն դեպքում, երբ $(S_1 \vee S_2)$ -ը անհամատեղելի է, այսինքն՝ և S_1 -ը, և S_2 -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 $S = (P \vee Q \vee \neg R) \wedge (P \vee \neg Q) \wedge \neg P \wedge R \wedge U$ -ը անհամատեղելի է:

$$(1) (P \vee Q \vee \neg R) \wedge (P \vee \neg Q) \wedge \neg P \wedge R \wedge U,$$

$$(2) (Q \vee \neg R) \wedge (\neg Q) \wedge R \wedge U \quad \text{կանոն 2. } \neg P,$$

$$(3) \neg R \wedge R \wedge U \quad \text{կանոն 2. } \neg Q,$$

$$(4) \Box \wedge U \quad \text{կանոն 2. } \neg R$$

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

Օրինակ՝ ցույց տանք, որ $S = (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (Q \vee \neg R) \wedge (\neg Q \vee \neg R)$ -ը համատեղելի է:

$$(1) (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (Q \vee \neg R) \wedge (\neg Q \vee \neg R),$$

$$(2) (\neg Q \wedge (Q \vee \neg R) \wedge (\neg Q \vee \neg R))$$

$$\vee (Q \wedge (Q \vee \neg R) \wedge (\neg Q \vee \neg R)) \quad \text{կանոն 4. } P$$

$$(3) \neg R \vee \neg R \quad \text{կանոն 2. } \neg Q \text{ և } Q$$

$$(4) \blacksquare \vee \blacksquare \quad \text{կանոն 2. } \neg R$$

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

Օրինակ՝ ցույց տանք, որ $S = (P \vee Q) \wedge (P \vee \neg Q) \wedge (R \vee Q) \wedge (R \vee \neg Q)$ -ը համատեղելի է:

$$(1) (P \vee Q) \wedge (P \vee \neg Q) \wedge (R \vee Q) \wedge (R \vee \neg Q),$$

$$(2) (R \vee Q) \wedge (R \vee \neg Q) \quad \text{կանոն 3. } P$$

$$(3) \blacksquare \quad \text{կանոն 3. } R$$

Այսպիսով S -ը համատեղելի է:

1.3 Ռեզոլյուցիայի մեթոդը տրամաբանական արտահայտություններում

Ռեզոլյուցիայի մեթոդը, ըստ էության, Դևիսի և Փաթնեմի մեկ լիտերալ դիզյունկտների կանոնի ընդհանրացումն է:

Օրինակ դիտարկենք հետևալ դիզյունկտները՝

$$C_1: P,$$

$$C_2: \neg P \vee Q$$

Օգտագործելով մեկ լիտերալ դիզյունկտների կանոնը, C_1 -ից և C_2 -ից մենք կարող ենք ստանալ նոր դիզյունկտ

$$C_3: Q$$

Մեկ լիտերալ դիզյունկտների կանոնը մեզ անհրաժեշտ է, որպեսզի նախ որոշենք, արդյոք կա լիտերալների հակադիր զույգ (օրինակ՝ P) C_1 -ում և (օրինակ՝ $\neg P$) C_2 -ում, ապա ջնջենք այդ զույգը C_1 -ից և C_2 -ից, որպեսզի ստանանք նոր դիզյունկտ C_3 , որը Q -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՝ C_1 և C_2 , եթե գոյություն ունի L_1 լիտերալ C_1 -ում, որը հակադիր է L_2 լիտերալին C_2 -ում, ապա ջնջելով L_1 -ը C_1 -ից և L_2 -ը C_2 -ից, մենք կառուցում ենք մնացած դիզյունկտների դիզյունկցիան: Ստացված դիզյունկտը կոչվում է C_1 -ի և C_2 -ի **ռեզոլվենտ**:

Օրինակ դիտարկենք հետևյալ դիզյունկտները՝

$$C_1: P \vee R,$$

$$C_2: \neg P \vee Q$$

C_1 -ը պարունակում է P լիտերալ, որը հակադիր է C_2 -ում գտնվող $\neg P$ լիտերալին: Ուստի, ջնջելով P -ն C_1 -ից և $\neg P$ -ն C_2 -ից, մենք կառուցում ենք մնացած դիզյունկտների դիզյունկցիան՝ R և Q , ստացված ռեզոլվենտը կլինի $R \vee Q$:

Ռեզոլվենտի կարևոր հատկությունն այն է, որ ցանկացած ռեզոլվենտ, որը ստացվում է երկու դիզյունկտներից՝ C_1 և C_2 , C_1 -ի և C_2 -ի տրամաբանական հետևանքն է: Այս հատկությունը հաստատվում է հետևյալ թեորեմով՝

Թեորեմ 1.2^[1] Եթե տրված են երկու դիզյունկտներ՝ C_1 և C_2 , ապա C_1 -ի և C_2 -ի ռեզոլվենտը C -ն C_1 -ի և C_2 -ի տրամաբանական հետևանքն է:

Սահմանում Ենթադրենք՝ S -ը դիզյունկտների բազմություն է: S -ից C -ի ռեզոլյուցիոն արտաձուլը դիզյունկտների վերջավոր հաջորդականություն է՝ C_1, C_2, \dots, C_k որտեղ յուրաքանչյուր C_i -ն կամ պատկանում է S -ին, կամ նախորդ դիզյունկտների ռեզոլվենտն է, և $C_k = C$: S -ից \square (դատարկ դիզյունկտ) արտաձուլը կոչվում է S -ի հերքում (կամ S -ի անհամատեղելիության ապացույց):

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

Օրինակ դիտարկենք բազմություն՝

$$S \begin{cases} \neg P \vee Q & (1) \\ \neg Q & (2) \\ P & (3) \end{cases}$$

(1)-ից և (2)-ից կարող ենք ստանալ ռեզոլվենտ՝ $\neg P$ (4): (4)-ից և (3)-ից կարող ենք ստանալ ռեզոլվենտ՝ \square : Զանի, որ \square -ն ստացվում է S -ից ռեզոլյուցիայի կանոնի կիրառմամբ, ապա համաձայն թեորեմ 1.0-ի, \square -ը S -ի տրամաբանական հետևանքն է: Ուստի, S -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 դիզյունկոնտներում: Երբ դիզյունկոնտերը չեն պարունակում փոփոխականներ, ապա դա շատ պարզ է: Սակայն, երբ դիզյունկոնտերը պարունակում են փոփոխականներ, ապա խնդիրը բարդանում է: Օրինակի համար դիտարկենք հետևյալ դիզյունկոնտերը՝

$$C_1: P(x) \vee Q(x),$$

$$C_2: \neg P(f(x)) \vee R(x)$$

Չկա որևէ լիտերալ C_1 -ում, որը հակադիր լինի C_2 -ի որևէ լիտերալի: Սակայն, եթե մենք C_1 -ում x -ը փոխարինենք $f(a)$ -ով, իսկ C_2 -ում x -ը փոխարինենք a -ով, ապա կստանանք՝

$$C'_1: P(f(a)) \vee Q(f(a)),$$

$$C'_2: \neg P(f(a)) \vee R(a),$$

Գիտենք, որ C'_1 -ը և C'_2 -ը համապատասխանաբար C_1 -ի և C_2 -ի հիմնական օրինակներն են, իսկ $P(f(a))$ -ն և $\neg P(f(a))$ -ն հակադիր են միմյանց: Ուստի, C'_1 -ից և C'_2 -ից մենք կարող ենք ստանալ ռեզոլվենտ՝

$$C'_3: Q(f(a)) \vee R(a)$$

Ընդհանուր դեպքում, եթե C_1 -ում x -ը փոխարինենք $f(x)$ -ով, ապա կստանանք՝

$$C_1^*: P(f(x)) \vee Q(f(x))$$

Կրկին C_1^* -ը C_2 -ի օրինակ է: Միևնույն ժամանակ, C_1 -ում $P(f(x))$ -ը հակադիր է C_2 -ում $\neg P(f(x))$ -ին: Ուստի, մենք կարող ենք ստանալ ռեզոլվենտ C_1^* -ից և C_2 -ից:

$$C_3: Q(f(x)) \vee R(x)$$

C_3' -ը C_3 -ի օրինակ է: Փոփոխականները C_1 -ում և C_2 -ում համապատասխան թերմերով փոխարինելով, ինչպես նշված է վերևում, մենք կարող ենք ստեղծել նոր դիզյունկոնյունտներ C_1 -ից և C_2 -ից: Բացի այդ, C_3 -ը *ամենաընդհանուր դիզյունկոնյունտ* է այն իմաստով, որ վերը նշված գործընթացով ստացված բոլոր այլ դիզյունկոնյունտները C_3 -ի օրինակներ են: C_3 -ը նույնպես կանվանենք C_1 -ի և C_2 -ի ռեզոլվենտ:

Սահմանում *տեղադրությունը* վերջավոր բազմություն է՝ $\{t_1/v_1, \dots, t_n/v_n\}$, որտեղ՝ յուրաքանչյուր v_i -ն փոփոխական է, յուրաքանչյուր t_i -ն թերմ է, որը տարբերվում է v_i -ից, բոլոր v_i -երը տարբեր են: Եթե t_1, t_2, \dots, t_n -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 ε -ով: Տեղադրությունը գրելու համար մենք կօգտագործենք հունարեն տառեր (օրինակ՝ θ, σ):

Օրինակ հետևալ երկու բազմությունները հանդիսանում են տեղադրություն՝

$$\{f(z)/x, y/z\}, \{a/x, g(y)/y, f(g(b))/z\}$$

Սահմանում Ենթադրենք $\theta = \{t_1/v_1, \dots, t_n/v_n\}$ -ը տեղադրություն է, և E -ն արտահայտություն է: Այդ դեպքում $E\theta$ -ն արտահայտություն է, որը ստացվում է E -ից՝ E -ում $v_i (1 \leq i \leq n)$ -ի բոլոր հանդիպումները միաժամանակ փոխարինելով t_i -ով: $E\theta$ -ն կոչվում է E -ի *օրինակ*: (Նշենք, որ օրինակի այս սահմանումը համատեղելի է գլուխ 4-ում տրված դիզյունկոնյունտի հիմնական օրինակի սահմանման հետ:)

Օրինակ՝ ենթադրենք $\theta = \{a/x, f(b)/y, c/z\}$ և $P(x, y, z)$: Այդ դեպքում $E\theta = P(a, f(b), c)$:

Սահմանում Ենթադրենք $\theta = \{t_1/x_1, \dots, t_n/x_n\}$ և $\lambda = \{u_1/y_1, \dots, u_m/y_m\}$ երկու տեղադրություններ են: Այդ դեպքում θ -ի և λ -ի *կոմպոզիցիան* (նշանակվում ենք $\theta \circ \lambda$) այն տեղադրությունն է, որը ստացվում է հետևալ բազմությունից՝

$$\{t_1\lambda/x_1, \dots, t_n\lambda/x_n, u_1/y_1, \dots, u_m/y_m\}$$

Չնշելով բոլոր այն $t_j\lambda/x_j$ -երը որոնց համար $t_j\lambda = x_j$, և բոլոր u_i/y_i -երը որոնց համար $y_i \in \{x_1, \dots, x_n\}$ (այսինքն՝ y_i -ն արդեն առկա է θ -ում):

Օրինակ՝ ենթադրենք

$$\theta = \{t_1/x_1, t_2/x_2\} = \{f(y)/x, z/y\},$$

$$\lambda = \{u_1/y_1, u_2/y_2, u_3/y_3\} = \{a/x, b/y\},$$

Այդ դեպքում $\{t_1\lambda/x_1, t_2\lambda/x_2, u_1/y_1, u_2/y_2, u_3/y_3\} = \{f(b)/x, y/y, a/x, b/y, y/z\}$: Սակայն, քանի որ $t_2\lambda = x_2$ (այսինքն y/y), ապա պետք է հեռացնել բազմությունից: Բացի այդ, քանի որ y_1 -ը և y_2 -ը առկա են $\{x_1, x_2, x_3\}$ -ում, ապա u_1/y_1 -ը և u_2/y_2 -ը (այսինքն՝ a/x -ը և b/y -ը) նույնպես պետք է հեռացվեն: Այսպիսով, մենք ստանում ենք՝

$$\theta \circ \lambda = \{f(b)/x, y/z\}$$

Սահմանում տեղադրություն θ -ն կոչվում է ունիֆիկատոր $\{E_1, E_2, \dots, E_k\}$ բազմության համար, այն և միայն այն դեպքում, երբ $E_1\theta = E_2\theta = \dots = E_k\theta$: Ասում են, որ $\{E_1, E_2, \dots, E_k\}$ բազմությունը *ունիֆիկացվող* է, եթե բազմության համար գոյություն ունի ունիֆիկատոր:

Սահմանում ունիֆիկատոր σ -ն $\{E_1, E_2, \dots, E_k\}$ բազմության համար կոչվում է ամենաընդհանուր ունիֆիկատոր, այն և միայն այն դեպքում, երբ ցանկացած այլ ունիֆիկատոր θ -ի համար գոյություն ունի տեղադրություն λ , այնպես որ՝ $\theta = \sigma \circ \lambda$:

Օրինակ՝ $\{P(a, y), P(x, f(b))\}$ բազմությունը ունիֆիկացվող է քանի, որ $\theta = \{a/x, f(b)/y\}$ 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 W -ի անհամապատասխանությունների բազմությունը ստացվում է գտնելով առաջին (ձախից) դիրքը, որտեղ W -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 W -ի *անհամապատասխանությունների բազմություն*:

Օրինակ՝ եթե W -ն հետևյալ բազմությունն է՝ $\{P(x, f(y, z)), P(x, a), P(x, g(h(k(x))))\}$, ապա առաջին դիրքը, որտեղ W -ի բոլոր արտահայտությունները չունեն նույն սիմվոլը, հինգերորդ դիրքն է, քանի որ բոլոր արտահայտությունները ունեն նույն առաջին չորս սիմվոլները՝ $P(x, :$ Այսպիսով, անհամապատասխանությունների բազմությունը բաղկացած է համապատասխան

ենթաարտահայտություններից, որոնք սկսվում են հիևգերորդ դիրքից, և դա հետևյալ բազմությունն է՝ $\{P(f(y, z), a, g(h(k(x))))\}$:

Ունիֆիկացման ալգորիթմ

Քայլ 1. Բազմություններ $k = 0$, $W_k = W$, $\sigma_k = \varepsilon$:

Քայլ 2. Եթե W_k -ն միալիտերալ դիզյունկտ է, ապա σ_k -ն W -ի ամենաընդհանուր ունիֆիկատորն է: Հակառակ դեպքում, գտնել W_k -ի անհամապատասխանությունների D_k բազմությունը:

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

Քայլ 4. Սահմանել $\sigma_{k+1} = \sigma_k \{t_k/v_k\}$ և $W_{k+1} = W_k \{t_k/v_k\}$: (Նշենք, որ $W_{k+1} = W_{\sigma_{k+1}}$):

Քայլ 5. k -ին վերագրել $k + 1$ արժեքը և անցնել քայլ 2-ին:

Օրինակ՝ գտնել ամենաընդհանուր ունիֆիկատորը

$$W = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$$

1. $\sigma_0 = \varepsilon$ և $W_0 = W$: Քանի, որ W_0 -ն միալիտերալ դիզյունկտ չէ, ուստի σ_0 -ն W -ի ամենաընդհանուր ունիֆիկատոր չէ:
2. Անհամապատասխանությունների բազմությունը՝ $D_0 = \{a, z\}$: D_0 -ում գոյություն ունի փոփոխական $v_0 = z$: որը չի հանդիպում $t_0 = a$ -ում:
3. Սահմանենք՝

$$\sigma_1 = \sigma_0 \circ \{t_0/v_0\} = \varepsilon \circ \{a/z\} = \{a/z\},$$

$$W_1 = W_0 \{t_0/v_0\}$$

$$= \{P(a, x, f(g(y))), P(z, f(z), f(u))\} \{a/z\}$$

$$= \{P(a, x, f(g(y))), P(a, f(a), f(u))\}$$

4. W_1 -ը միալիտերալ դիզյունկտ չէ, քանի որ գտնվել է անհամապատասխանությունների բազմություն D_1 W_1 -ի համար: $D_1 = \{x, f(a)\}$
5. D_1 -ից կգտնենք $v_1 = x$ և $t_1 = f(a)$:
6. Սահմանենք՝

$$\sigma_2 = \sigma_1 \circ \{t_1/v_1\} = \{a/z\} \circ \{f(a)/x\} = \{a/z, f(a)/x\},$$

$$\begin{aligned}
W_2 &= W_1\{t_1/v_1\} \\
&= \{P(a, x, f(g(y))), P(a, f(a), f(u))\}\{f(a), x\} \\
&= \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\}
\end{aligned}$$

7. W_2 -ը միալիտերալ դիզյունկտ չէ, քանի որ գտնվել է անհամապատասխանությունների բազմություն D_2 W_2 -ի համար: $D_2 = \{g(y), u\}$: D_2 -ից կգտնենք $v_2 = u$ և $t_2 = g(y)$:

8. Սահմանենք՝

$$\begin{aligned}
\sigma_3 &= \sigma_2 \circ \{t_2/v_2\} = \{a/z, f(a)/x\} \circ \{g(y), u\} = \{a/z, f(a)/x, g(y)/u\}, \\
W_3 &= W_2\{t_2/v_2\} \\
&= \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\}\{g(y)/u\} \\
&= \{P(a, f(a), f(g(y))), P(a, f(a), f(g(y)))\} \\
&= \{P(a, f(a), f(g(y)))\}
\end{aligned}$$

9. Քանի, որ W_3 -ը միալիտերալ դիզյունկտ է, ապա $\sigma_3 = \{a/z, f(a)/x, g(y)/u\}$ -ն W -ի ամենաընդհանուր ունիֆիկատորն է:

Թեորեմ 1.3^[1] (Ունիֆիկացման թեորեմ)՝ Եթե W -ն վերջավոր ոչ դատարկ ունիֆիկացվող արտահայտությունների բազմություն է, ապա ունիֆիկացման ալգորիթմը միշտ կավարտվի քայլ 2-ում, և վերջին σ_k -ն կլինի W -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 C -ի երկու կամ ավելի լիտերալներ (նույն նշանով) ունեն ամենաընդհանուր ունիֆիկատոր σ , ապա $C\sigma$ -ն կոչվում է C -ի *սոսնձում*: Եթե $C\sigma$ -ն միալիտերալ դիզյունկտ է, ապա սոսնձումը կոչվում է *միալիտերալ սոսնձում*:

Օրինակ՝ ենթադրենք $C = \underline{P(x)} \vee \underline{P(f(y))} \vee \neg Q(x)$: Այդ դեպքում առաջին և երկրորդ լիտերալները (ընդգծված) ունեն ամենաընդհանուր ունիֆիկատոր $\sigma = \{f(y)/x\}$: Ուստի, $C\sigma = P(f(y)) \vee \neg Q(f(y))$ -ը C -ի սոսնձումն է:

Սահմանում Եթե C_1 -ը և C_2 -ը երկու դիզյունկտներ են (կոչվում են դիզյունկտ-նախադրյալներ), որոնք չունեն ընդհանուր փոփոխականներ: Թող L_1 -ը և L_2 -ը լինեն երկու լիտերալներ համապատասխանաբար C_1 -ում և C_2 -ում: Եթե L_1 -ը և $\neg L_2$ -ն ունեն ամենաընդհանուր ունիֆիկատոր σ , ապա դիզյունկտը՝

$$(C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma)$$

կոչվում է C_1 -ի և C_2 -ի (երկուական) ռեզոլվենտ: L_1 -ը և L_2 -ը կոչվում են *հեռացվող լիտերալներ*:

Օրինակ՝ ենթադրենք $C_1 = P(x) \vee Q(x)$, $C_2 = \neg P(a) \vee R(x)$: Քանի որ x -ը ներառված է և՛ C_1 -ում, և՛ C_2 -ում, մենք փոխարինում ենք C_2 -ում x -ը y -ով՝ $C_2 = \neg P(a) \vee R(y)$: Ընտրում ենք՝ $L_1 = P(x)$, $L_2 = \neg P(a)$: Քանի որ $\neg L_2 = P(a)$., ապա L_1 -ը և $\neg L_2$ -ն ունեն ամենաընդհանուր ունիֆիկատոր $\sigma = \{a/x\}$: Այսպիսով՝

$$\begin{aligned} & (C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma) \\ &= (\{P(a), Q(a)\} - \{P(a)\}) \cup (\{\neg P(a), R(y)\} - \{\neg P(a)\}) \\ &= (\{Q(a)\} \cup \{R(y)\}) = (\{Q(a), R(y)\}) = Q(a) \vee R(y) \end{aligned}$$

Ուստի, $Q(a) \vee R(y)$ -ն C_1 -ի և C_2 -ի երկուական ռեզոլվենտն է: $P(x)$ -ը և $\neg P(a)$ -ն հեռացվող լիտերալներն են:

Թեորեմ 1.4^[2] (ռեզոլյուցիա մեթոդի լրիվություն)՝ S բազմությունը անհամատեղելի է այն և միայն այն դեպքում, երբ գոյություն ունի դատարկ դիզյունկտի \square դուրս բերում S -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 «ոլորին → 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՝ $\text{fof}(\langle \text{անուն} \rangle, \langle \text{դեր} \rangle, \langle \text{բանաձև} \rangle)$, $\text{cnf}(\langle \text{անուն} \rangle, \langle \text{դեր} \rangle, \langle \text{դիզյունկտ} \rangle)$, 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, այսինքն 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 կոնտեյնրերի 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 կոմպիլացիան, 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 վերսիսների 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 Կոնվոլյացիոն 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 ($j \rightarrow i$), հաղորդագրություն $m_{j \rightarrow i}$ հաշվարկվում է հարևան j -ի հատկանիշներից (և հնարավոր է նաև i -ի կամ կողի բնութագրերից): Օրինակ, պարզ հաղորդագրություն կարող է լինել ուղղակի h_j (հարևանի ընթացիկ ներկայացումը), կամ ավելի բարդ ֆունկցիա՝ $m_{j \rightarrow i} = f_e(h_i, h_j, e_{ij})$, որը ներառում է նաև կողի բնութագրերը e_{ij} -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 $\sum_{j \in N(i)} m_{j \rightarrow i}$, 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՝ $h_i^{(new)} = \sigma(W \cdot [h_i^{(old)} \parallel AGG(\{m_{j \rightarrow i}\})])$ կամ ավելի բարդ ֆունկցիա, ինչպիսին է gated recurrent unit-ը (GRU): Որտեղ, σ -ն ոչ գծային ակտիվացիայի ֆունկցիա է (օր. ReLU, sigmoid), \parallel -ը նշանակում է վեկտորների կոնկատենացիա (միավորում), W -ը ուսուցվող կշիռների մատրիցն է: Որոշ ձևակերպումներում թարմացումը գրվում է հետևյալ կերպ՝ $h_i^{(k+1)} = COMBINE(h_i^{(k)}, AGG_{j \in N(i)}(m_{j \rightarrow i}^{(k)}))$: 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՝ $H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})$, որտեղ՝ $\tilde{A} = A + I$ -ին և \tilde{D} -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 ինժինեռինգ:
- *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՝

$$a_{ij} = \text{softmax}_j(\text{LeakyReLU}(a^T [Wh_i \parallel Wh_j]))$$

Հարևանների հաղորդագրությունները դառնում են կշռված գումարներ՝

$$h_i^{(l+1)} = \sigma(\sum_j a_{ij} Wh_j)$$

Բազմապլուխ 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 ագրեգիկներին:
- *Մեկնաբանելիություն* – սովորած $\{a_{ij}\}$ 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 ենք 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 ռեգուլյուցիայի կիրառում

- Ենթադրենք $C_0 \dots C_N$ -ը դիզյունկտների հաջորդականություն է:
- Գործընթացը սկսվում է C_0 դիզյունկտից, որը պատահականորեն ընտրվում է աքսիոմների բազայից:

- Յուրաքանչյուր $t = 1 \dots N$ քայլի համար՝
 - Վերցվում է նախորդ C_{t-1} դիզյունկտորը:
 - Իրականացվում է ռեգուլուցիա ցանկացած այլ դիզյունկտորի հետ, որի հետ հնարավոր է այն իրականացնել: Դիզյունկտոր կարող է լինել կամ ակսիոմների ցանկից կամ արդեն ստացված շղթայի դիզյունկտորների ցանկից:
 - Ստեղծվում է նոր C_t դիզյունկտ, որը հանդիսանում է նախորդ երկու դիզյունկտորների ռեգուլվենտը:
 - Եթե C_t -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 ռեգուլուցիան պահպանում է ամբողջականությունը, այսինքն՝ տեսականորեն այն կարող է հանգել ցանկացած դիզյունկտի, որին կարող է հանգել ռեգուլուցիան:

3. Դիզյունկտի չափի օպտիմալացում

Միատեսակ նմուշառումը ռեգուլուցիաների 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՝

$$P(i) = \frac{\exp(-|C_i|/T)}{\sum_j \exp(-|C_j|/T)}$$

Որտեղ T -ն փոփոխական է, $|C_i|$ -ին եզրակացություն i -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 C_N դիզյունկտը դառնում է ապացուցման թեզ, որը Vampire ATP-ն պետք է լուծի ռեգուլուցիաներ անելով:
- Ստացվում է վավեր խնդիր՝ $Axioms \vdash C_N$:

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 $\geq 500,000$ ունիկալ թեորեմ:
- Ընտրվում է ամենադժվար տարբերակը սահմանված պայմաններում:

6. Ուսուցման կորպուսի ստեղծում

- Ընտրված պարամետրերով գեներացնում են 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 ռեգուլյուցիայի օժանդակ մոդուլ

Խնդիրներ գեներացնելու ընթացքում գործարկվում է հատուկ օժանդակ մոդուլ, որը կատարում է Ռոբինսոնի ունիֆիկացիայի դասական ալգորիթմը և դրան հաջորդող ռեգուլյուցիայի 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:
- *Ռեգուլյուցիա* - եթե լիտերալների զույգը կոնտրար են (օր. $P(a)$ և $\neg P(a)$ 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 $C_0 \rightarrow C_1 \rightarrow \dots \rightarrow C_N$ յուրաքանչյուր քայլում հենց այս մեխանիզմով է ստացվում հաջորդ դիզյունկտը (ռեգուլվենտը):

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՝ $p(X) \mid \neg p(X)$):
- 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 C_0 աքսիոմ և սկսում է հաջորդականություն կառուցել՝

- Արագորեն ստանում է բոլոր հնարավոր ռեզոլյուցիոն քայլերը C_0 -ի և մնացած աքսիոմների միջև:
- Հավանականությունների հիման վրա ընտրում է մեկ «առաջնահերթ» ելք՝ ստանալով C_1 դիզյունկտոր:
- Նույն գործընթացը կրկնում է մինչև հասնում է նախորոշված N երկարությանը (օրինակ՝ 10 քայլ)

Այսպես ձևավորվում է $C_0 \rightarrow C_1 \rightarrow \dots \rightarrow C_N$ շղթան, որտեղ յուրաքանչյուր հաջորդ դիզյունկտ հանդիսանում է աքսիոմների և նախորդ ստացված դիզյունկտների տրամաբանական հետևանքը:

ՔԱՅԼ 3 - վարկածի ժխտում:

Վերջնական C_N դիզյունկտորը ստանալուց հետո այն տրոհվում է առանձին լիտերալների: Յուրաքանչյուր լիտերալ ժխտվում է (օր. $p(X) \rightarrow \neg p(X)$), և այդ նոր, մեկական դիզյունկտները ֆայլում գրանցվում են «negated_conjecture» դերով: Ստացված ֆայլում այս մասը կունենա հետևյալ տեսքը՝

`cnf(cn_neg1, negated_conjecture, ($\neg p(X)$)).`

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 $\neg P$) 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, ...)
- *Մերմի 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 խտերատիվ 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* - պատրաստ 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 համակարգից:
- 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:
- Զանի, 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 $O(N \times L)$ 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),
- Լիտերալների արգումենտները ($\text{max_args}=8$) - 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 ռեգուլյուցիայի 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 ռեգուլյուցիայի 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 ռեգուլյուցիայի 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՝ $\text{weight}=[1.0, 3.0]$, 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 ռեգուլյացիոն 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-խնդրես համապատասխանեցման բառարան:
- Համապատասխանեցնում է այս ինդեքսները լիտերալների ներկայացման մեջ:

Այս մոտեցումը թույլ է տալիս մոդելին հարմարվել տարբեր պրեդիկատների հավաքածուների և նոր, չտեսված 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 (կոյր) մոտեցումը ապացուցման գործընթացում:

Թեստավորումը հիմնված է երկու տարբեր մոտեցումների համեմատության վրա՝

- *Կոյր 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 դիզոնկտները: Թեստավորումն իրականացվել է այն խնդիրների և արքիտեկտուրայի 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 եականորեն 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 ավգորիթը 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 ընթացքում կատարված քայլերի քանակների համեմատում:
- *Հարաբերական բարելավում* - հաշվարկվում է հարաբերական բարելավումը տոկոսային հարաբերությամբ:
- *Արագագործության բարելավում* - այն խնդիրների տոկոսը, որոնցում GNN-ն ավելի արագ է ապահանջում 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 աքսիոնների ցանկը կարելի է գեներացնել սինթետիկ եղանակով կամ վերցնել TPTP գրադարանի աքսիոնների ցանկից, որոնցից յուրաքանչյուրը վերաբերում է կոնկրետ ոլորտի: Առկա աքսիոնների 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 յուրաքանչյուր 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, MIT 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/>