ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

Առաքելյան Արա Հայկի

ՖՈՒՆԿՑԻՈՆԱԼ ԾՐԱԳՐԵՐԻ ՏԻՊԱՅԻՆ ԿՈՌԵԿՏՈՒԹՅԱՆ ՄԱՍԻՆ

Ա.01.09 - «Մաթեմատիկական կիբեռնետիկա և մաթեմատիկական տրամաբանություն» մասնագիտությամբ ֆիզիկամաթեմատիկական գիտությունների թեկնածուի գիտական աստիճանի հայցման ատենախոսության

Ս Ե Ղ Մ Ա Գ Ի Ր

Երևան-2011

ЕРЕВАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Аракелян Ара Гайкович

О ТИПОВОЙ КОРРЕКТНОСТИ ФУНКЦИОНАЛЬНЫХ ПРОГРАММ

А В Т О Р Е Ф Е Р А Т

диссертации на соискание ученой степени кандидата физико-математических наук по специальности 01.01.09 – “Математическая кибернетика и математическая логика”

Ереван-2011

Ատենախոսության թեման հաստատվել է Երևանի պետական համալսարանում

Գիտական ղեկավար՝	ֆիզ.մաթ. գիտ. դոկտոր	Ս.Ա. Նիգիյան
Պաշտոնական ընդդիմախոսներ՝	ֆիզ.մաթ. գիտ. դոկտոր ֆիզ.մաթ. գիտ. թեկնածու	Հ.Բ. Մարանջյան Գ.Գ. Հրաչյան
Առաջատար կազմակերպություն՝	ՀՀ ԳԱԱ Ինֆորմատիկայի և ավտոմատացման պրոբլեմների ինստիտուտ	

Պաշտպանությունը կայանալու է 2011 թ. մայիսի 20-ին, ժամը 14^{00}-ին ԵՊՀ-ում գործող ԲՈՀ-ի 044 «Մաթեմատիկական կիբեռնետիկա և մաթեմատիկական տրամաբանություն» մասնագիտական խորհրդի նիստում, հետևյալ հասցեով՝ 0025, Ալեք Մանուկյան 1:

Ատենախոսությանը կարելի է ծանոթանալ ԵՊՀ-ի գրադարանում:

Սեղմագիրն առաքված է 2011 թ. ապրիլի 19-ին:

Մասնագիտական խորհրդի գիտական քարտուղար՝
ֆիզ.մաթ. գիտ. թեկնածու Վ.Ժ. Դումանյան

Тема диссертации утверждена в Ереванском государственном университете

Научний руководитель:	доктор физ.-мат. наук	С.А. Нигиян
Официальные оппоненты:	доктор физ.-мат. наук кандидат физ.-мат. наук	Г.Б. Маранджян Г.Г. Грачян
Ведущая организация:	Институт проблем информатики и автоматизации НАН РА	

Защита состоится 20-го мая 2011 г. в 14^{00} часов на заседании действующего в ЕГУ специализированного совета ВАК 044 “Математическая кибернетика и математическая логика” по адресу: 0025, г. Ереван, ул. Алека Манукяна, 1.

С диссертацией можно ознакомиться в библиотеке ЕГУ.

Автореферат разослан 19-го апреля 2011 г.

Ученый секретарь специализированного совета,
кандидат физ.-мат. наук В.Ж. Думанян

ԱՇԽԱՏԱՆՔԻ ԸՆԴՀԱՆՈՒՐ ԲՆՈՒԹԱԳԻՐԸ

Թեմայի արդիականությունը: Աշխատանքը նվիրված է ֆունկցիոնալ ծրագրավորման խնդիրներին: Հետազոտության առարկան հանդիսանում են առանց տիպերի λ-թերմերը և դրանք օգտագործող ֆունկցիոնալ ծրագրերը, իսկ դիտարկվող հիմնական խնդիրը՝ այդ թերմերի և ֆունկցիոնալ ծրագրերի տիպային կոռեկտության խնդիրը: Տիպերը ծրագրավորման լեզուներում օգտագործվում են տարբեր նպատակներով, որոնցից են ծրագրում առկա տիպերի անհամաձայնության սխալների ավելի վաղ հայտնաբերումը (մինչև ծրագրի աշխատանքը), որոշակի օպտիմիզացիաների կատարումը և այլն: Այս աշխատանքի հիմնական խնդիրը կապված է ֆունկցիոնալ ծրագրավորման լեզուներում վերը նշված նպատակներից առաջինի իրագործման հետ. ստուգելով ծրագրի տիպային կոռեկտությունը մինչև նրա աշխատանքը՝ խուսափել ծրագրի աշխատանքի ընթացքում ի հայտ եկող տիպերի անհամաձայնության սխալներից: Մի շարք ծրագրավորման լեզուներում ծրագրավորողը բացահայտ կերպով չի նշում ինֆորմացիա տիպերի մասին: Այդպիսի ֆունկցիոնալ ծրագրավորման լեզուներից են հետևյալ լեզուները՝ Lisp, SML, OCaml, F# և այլն: Այդ դեպքում անհրաժեշտություն է առաջանում ստեղծել տիպերի արտածման համակարգեր, որոնք օգտագործվում են ծրագրերի տիպայնացման ժամանակ՝ փորձելով վերականգնել տիպերի մասին ծրագրավորողի կողմից չնշված ինֆորմացիան: Տիպայնացման ալգորիթմների կիրառելիության համար կարևոր պայման է հանդիսանում այն, որ այդ ալգորիթմները պետք է ապահովեն իրենց կողմից տիպայնացվող ցանկացած ծրագրի տիպային կոռեկտությունը: Այսինքն, եթե այդ ալգորիթմը տիպայնացնում է տվյալ ծրագիրը, ապա դա նշանակում է, որ այդ ծրագրի համար ինտերպրետատորի աշխատանքի ընթացքում չեն հանդիպելու տիպերի անհամաձայնության սխալներ: Սովորաբար օգտագործվում են ֆունկցիոնալ ծրագրերի ինտերպրետացիայի այնպիսի ալգորիթմներ, որոնք հիմնված են տեղադրումների և ռեդուկցիաների վրա: Այսպիսով, տիպայնացման ալգորիթմները «պարտավոր են մերժել» այն բոլոր ծրագրերը, որոնց ինտերպրետացիայի ընթացքում հանդիպելու են տիպերի անհամաձայնության սխալներ: Մյուս կողմից, տիպի սխալներ պարունակող ծրագրերի հետ միասին, տիպայնացման ալգորիթմները կարող են «մերժել» նաև այնպիսի ծրագրեր, որոնց ինտերպրետացիայի ընթացքում տիպի սխալներ տեղի չեն ունենում: Վերջինս նշանակում է, որ տիպայնացման ալգորիթմների համար կարևոր է նաև, որ իրենց կողմից «մերժվեն» ինչքան հնարավոր է քիչ այնպիսի ծրագրեր, որոնք տիպի սխալներ չեն պարունակում: Ամփոփելով այս ամենը՝ նշենք, որ ֆունկցիոնալ ծրագրերի տիպային կոռեկտությանն առնչվող խնդիրներն արդիական են:

Կիրառվող տիպերի արտածման համակարգերից է [1]աշխատանքում

[1] Milner R. A Theory of Type Polymorphism in Programming. Journal of Computer and System Sciences, Vol. 17, №3, Elsevier Science Pub. Co., USA, 1978, p. 348-375.

ներկայացված համակարգը, որն անվանում են նաև Hidney/Milner-ի համակարգ: Այդ համակարգը կամ նրա ձևափոխված տարբերակն օգտագործվում է SML, OCaml և այլ ծրագրավորման լեզուներում: [1]աշխատանքում ներկայացվում և ամփոփվում են տիպերի արտածման համակարգերի հիմնական տեսակները: Բացի այդ, ներկայացվում և ապացուցվում են դրանց հատկությունները: Տիպերի արտածման համակարգերի համար կարևոր հատկություն է նաև «գլխավոր տիպայնացման» (principal typing) հատկությունը, որը նկարագրված է [2,3]աշխատանքներում: Այդ աշխատանքներում ցույց է տրվում նաև, որ վերը նշված Hidney/Milner-ի համակարգը չի բավարարում այդ հատկությանը: [4,5]աշխատանքներում ներկայացվում է տիպերի արտածման մեկ այլ համակարգ, որը կոչվում է SYSTEM E և բավարարում է «գլխավոր տիպայնացման» հատկությանը: Այդ համակարգը ներկայացվում է միայն հաստատուններ չպարունակող թերմերի համար: Հետևաբար, առաջանում է խնդիր՝ ընդլայնել այդ համակարգը և հետազոտել ընդլայնված տարբերակի հատկությունները հաստատուններ պարունակող թերմերի համար: Օրինակ՝ ընդլայնված SYSTEM E համակարգը բավարարու՞մ է արդյոք գլխավոր տիպայնացման հատկությանը, թե՞ ոչ: Հետաքրքրություն է ներկայացնում նաև հաստատուններ պարունակող թերմերն օգտագործող ֆունկցիոնալ ծրագրերի համար SYSTEM E համակարգի ընդլայնման և ընդլայնված տարբերակի հատկությունների ուսումնասիրման խնդիրը:

Ինչպես նշել էինք վերևում, տիպայնացման ալգորիթմների համար կարևոր է, որ ինչքան հնարավոր է քիչ ծրագրեր «մերժվեն», որոնք իրականում տիպի սխալներ չեն պարունակում: Հետևաբար, առաջանում է հետևյալ հարցը՝ հնարավո՞ր է արդյոք ստեղծել տիպայնացման այնպիսի ալգորիթմ, որը «կմերժի» բոլոր այն ծրագրերը, որոնց աշխատանքի ընթացքում տեղի են ունենում տիպի սխալներ և կտիպայնացնի բոլոր այն ծրագրերը, որոնց աշխատանքի ընթացքում

[1] Barendregt H.P. Lambda Calculi with Types. Handbook of Logic in Computer Science, Vol. 2, Edited by S. Abramsky, D. M. Gabbay, T. S. E. Maibaum, Oxford University Press, New York, 1992, p. 117-309.

[2] Jim T. What are Principal Typings and What are They Good For? Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM, New York, 1996, p. 42-53.

[3] Wells J.B. The Essence of Principal Typings. Proceedings of the 29th International Colloquium on Automata, Languages and Programming, Springer-Verlag, London, 2002, p. 913-925.

[4] Carlier S., Wells J.B. Type Inference with Expansion Variables and Intersection Types in System E and an Exact Correspondence with β-Reduction. Proceedings of the 6th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming, ACM, New York, 2004, p. 132-143.

[5] Carlier S., Polakow J., Wells J.B., Kfoury A.J. System E: Expansion Variables for Flexible Typing with Linear and Non-linear Types and Intersection Types. Proceedings of the 11th European Symposium on Programming Languages and Systems, Springer-Verlag Pub. Co., London, 2004, p. 294-309.

տիպի սխալներ տեղի չեն ունենում: Հետաքրքրություն է ներկայացնում նաև հետևյալ խնդիրը՝ կախվա՞ծ է արդյոք վերը նշված հարցի պատասխանը տվյալ լեզվի ինտերպրետատորից, թե՞ ոչ: Այսպիսով, հետաքրքրական է տիպայնացման ալգորիթմների հնարավորությունների սահմանի ճշգրտման խնդիրը:

Աշխատանքի նպատակն ու խնդիրները: Այս աշխատանքի հիմնական նպատակն ու խնդիրները հետևյալն են.

1. Հետազոտել կառուցվածքային λ-թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի տիպային կոռեկտության խնդիրը, կառուցել տիպերի արտածման համակարգ և տիպայնացման ալգորիթմներ այդ թերմերի ու ծրագրերի համար և հետազոտել կառուցված համակարգի ու ալգորիթմների հնարավորությունները:
2. Հետազոտել առանց տիպերի λ-թերմերի տիպայնացման SYSTEM E համակարգի ընդլայնման հնարավորությունները հաստատուններ պարունակող λ-թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի համար, կառուցել տիպայնացման ալգորիթմներ այդ թերմերի ու ծրագրերի համար և հետազոտել կառուցված ալգորիթմների հնարավորությունները:
3. Հետազոտել հաստատուններ պարունակող λ-թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի տիպայնացման հնարավոր ալգորիթմների հնարավորությունները:

Հետազոտության մեթոդները: Աշխատանքում օգտագործված հետազոտության մեթոդները ներառում են առանց տիպերի λ-հաշվի, ալգորիթմների տեսության, մաթեմատիկական տրամաբանության և հանրահաշվի մեթոդները:

Արդյունքների նորությունը: Ատենախոսության հիմնական արդյունքները հետևյալն են.

1. Կառուցվածքային λ-թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի համար կառուցվել է տիպերի արտածման համակարգ: Ապացուցվել է, որ միևնույն միջավայրում տիպայնացվող ֆունկցիոնալ ծրագրի և թերմի համար ցանկացած ինտերպրետացիայի ալգորիթմի աշխատանքի ընթացքում ստացվող թերմերը չեն պարունակի տիպի սխալ: Ստացվել են նաև կառուցվածքային λ-թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի տիպայնացման այնպիսի ալգորիթմներ, որոնք միշտ կանգ են առնում և հաջող ավարտի դեպքում տիպայնացնում են տրված թերմը կամ ֆունկցիոնալ ծրագիրը ամենաընդհանուր ձևով, իսկ անհաջող ավարտ տեղի է ունենում միայն այն դեպքում, երբ տրված թերմը կամ ֆունկցիոնալ ծրագիրը տիպայնացվող չէ վերը նշված տիպերի արտածման համակարգում:
2. Ընդլայնվել է առանց տիպերի λ-թերմերի տիպերի արտածման SYSTEM E համակարգը հաստատուններ պարունակող թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի համար: Ապացուցվել է, որ «միասին և նորմալ» տիպայնացվող ծրագրի և թերմի համար մեր կողմից դիտարկվող

ինտերպրետացիայի ալգորիթմների աշխատանքի ընթացքում ստացվող թերմերը չեն պարունակի տիպի սխալ: Ընդլայնվել է նաև SYSTEM E համակարգում առանց տիպերի λ-թերմերի տիպայնացման ալգորիթմը հաստատուններ պարունակող թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի համար և ապացուցվել է, որ հաջող ավարտի դեպքում թերմերի տիպայնացման ալգորիթմը դուրս է բերում հաստատուններ պարունակող թերմի գլխավոր տիպայնացումը, իսկ անհաջող ավարտ տեղի է ունենում միայն այն դեպքում, երբ ընդլայնված SYSTEM E համակարգում տրված թերմը նորմալ տիպայնացվող չէ:

3. Ապացուցվել է, որ գոյություն չունի հաստատուններ պարունակող λ-թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի տիպայնացման այնպիսի ալգորիթմ, որն ընդունում է բոլոր այն թերմերը և ֆունկցիոնալ ծրագրերը, որոնց ինտերպրետացիայի ժամանակ տիպի սխալներ տեղի չեն ունենում և մերժում է բոլոր այն թերմերը և ֆունկցիոնալ ծրագրերը, որոնց ինտերպրետացիայի ժամանակ տեղի են ունենում տիպի սխալներ:

Տեսական և կիրառական նշանակությունը: Այս աշխատանքում ստացված արդյունքներն ունեն ինչպես տեսական, այնպես էլ կիրառական նշանակություն: Դրանք կարող են օգտագործվել ֆունկցիոնալ ծրագրավորման նոր համակարգերի ստեղծման ժամանակ, ինչպես նաև՝ ֆունկցիոնալ ծրագրերի տիպային կոռեկտության խնդրի ուսումնասիրությունների համար:

Ստացված արդյունքների ապրոբացիան: Ատենախոսության հիմնական արդյունքները զեկուցվել են ԵՊՀ ծրագրավորման և ինֆորմացիոն տեխնոլոգիաների ամբիոնի սեմինարում, ԵՊՀ ինֆորմատիկայի և կիրառական մաթեմատիկայի ֆակուլտետի ընդհանուր սեմինարում և միջազգային գիտաժողովներում (CSIT-07, Երևան, 2007 և CSIT-09, Երևան, 2009):

Աշխատանքի հիմնական արդյունքներն ընդգրկված են հրատարակված հինգ աշխատանքներում:

Աշխատանքի կառուցվածքն ու ծավալը: Ատենախոսությունը բաղկացած է ներածությունից, երեք գլխից, եզրակացությունից և օգտագործված գրականության ցանկից (45 անուն): Ատենախոսության ծավալը 149 էջ է:

ԱՇԽԱՏԱՆՔԻ ԲՈՎԱՆԴԱԿՈՒԹՅՈՒՆԸ

Ներածությունում նկարագրվում է ատենախոսության հետազոտության հիմնական խնդիրը, հիմնավորվում է ատենախոսության թեմայի արդիականությունը և նորությունը: Համառոտ կերպով ներկայացվում է աշխատանքի բովանդակությունը:

Գլուխ 1-ը նվիրված է կառուցվածքային λ-թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի տիպերի արտածման համակարգին ու տիպայնացման

ալգորիթմներին և դրանց հիմնավորմանը: Առաջին գլուխը բաղկացած է հինգ բաժնից՝ 1.1-1.5:

1.1 բաժնում ներկայացվում են առաջին գլխում օգտագործվող սահմանումները: Դիցուք ունենք $VariableName$ տիպերի փոփոխականների անունների հաշվելի և $TypeConstant$ տիպերի հաստատունների վերջավոր բազմությունները:

Սահմանում 1.1.1 $TypeVariable \subset VariableName \times 2^{TypeConstant}$ զույգերի բազմությունը կանվանենք տիպերի փոփոխականների բազմություն, եթե տեղի ունեն հետևյալ պայմանները՝

1. եթե $n \in VariableName$, ապա գոյություն ունի $S \in 2^{TypeConstant}$ այնպիսին, որ $(n, S) \in TypeVariable$,
2. եթե $(n, S_1) \in TypeVariable$ և $(n, S_2) \in TypeVariable$, ապա $S_1 = S_2$,
3. եթե $S \in 2^{TypeConstant}$, ապա $\{n \mid n \in VariableName \text{ և } (n, S) \in TypeVariable\}$ բազմությունը հաշվելի է:

Եթե $\alpha = (n, S) \in TypeVariable$, ապա n-ը կանվանենք α փոփոխականի անուն և կնշանակենք $Name(\alpha)$-ով, իսկ S-ը կնշանակենք $Range(\alpha)$-ով:

Սահմանում 1.1.2 Տիպերի $Type$ բազմությունը սահմանվում է ինդուկտիվ հետևյալ ձևով՝

1. եթե $\alpha \in TypeVariable$, ապա $\alpha \in Type$,
2. եթե $\tau_1, \dots, \tau_{n+1} \in Type$, որտեղ $n \geq 1$, ապա $(\tau_1 \times \dots \times \tau_n \to \tau_{n+1}) \in Type$:

Սահմանում 1.1.3 Դիցուք $\tau_1, \dots, \tau_n \in Type$, $\alpha_1, \dots, \alpha_n \in TypeVariable$, $n \geq 0$: Այդ դեպքում $\sigma = \{\alpha_1 := \tau_1, \dots, \alpha_n := \tau_n\}$ զույգերի բազմությունը կոչվում է տեղադրություն, եթե տեղի ունեն հետևյալ պայմանները՝

1. $\tau_i \neq \alpha_i$, որտեղ $i = 1, \dots n$,
2. եթե $i \neq j$, ապա $\alpha_i \neq \alpha_j$, որտեղ $i, j = 1, \dots, n$,
3. եթե $Range(\alpha_i) \neq \emptyset$, ապա $\tau_i \in TypeVariable$ և $\emptyset \neq Range(\tau_i) \subset Range(\alpha_i)$, որտեղ $i = 1, \dots n$:

Սահմանում 1.1.4 Դիցուք $\tau \in Type$ և $\sigma = \{\alpha_1 := \tau_1, \dots, \alpha_n := \tau_n\}$-ն տեղադրություն է: $[\sigma]\tau$-ով կնշանակենք այն տիպը, որը ստացվում է τ տիպից՝ նրանում $\alpha_1, \dots, \alpha_n$ փոփոխականների փոխարեն միաժամանակ $\tau_1, \dots, \tau_n$ տիպերը տեղադրելով: Կասենք, որ $[\sigma]\tau$ տիպը ստացվել է τ տիպից՝ նրա վրա σ տեղադրության կիրառման արդյունքում:

Սահմանում 1.1.5 Դիցուք $\sigma_1 = \{\alpha_1 := \tau_1, \dots, \alpha_n := \tau_n\}$-ը և $\sigma_2 = \{\beta_1 := \eta_1, \dots, \beta_m := \eta_m\}$-ը տեղադրություններ են: $\{\alpha_i := [\sigma_2]\tau_i \mid [\sigma_2]\tau_i \neq \alpha_i, i = 1, \dots, n\} \cup \{\beta_j := \eta_j \mid \beta_j \notin V1(\sigma_1), j = 1, \dots, m\}$ բազմությունը կնշանակենք $[\sigma_2]\sigma_1$-ով և կասենք, որ այն ստացվել է σ_1 տեղադրությունից՝ նրա վրա σ_2 տեղադրության կիրառման արդյունքում: Դժվար չէ նկատել, որ $[\sigma_2]\sigma_1$-ը նույնպես հանդիսանում է տեղադրություն:

Սահմանում 1.1.6 Թերմերի $Term$ բազմությունը սահմանվում է ինդուկտիվ հետևյալ ձևով՝

1. $\bot \in Term$ և $\bot$-ին անվանում են տիպի սխալը ներկայացնող թերմ,
2. եթե $x \in TermVariable$, ապա $x \in Term$,
3. եթե $c \in Constant$, ապա $c \in Term$,
4. եթե $x_1, \dots, x_n \in TermVariable$ և $M \in Term$, որտեղ $x_i \neq x_j$, երբ $i \neq j$, $i, j = 1, \dots, n$, $n \geq 1$, ապա $(\lambda x_1 \dots x_n . M) \in Term$,
5. եթե $F, M_1, \dots, M_n \in Term$, որտեղ $n \geq 1$, ապա $F(M_1, \dots, M_n) \in Term$:

Թերմի ազատ փոփոխականի գաղափարը և թերմում փոփոխականի ազատ և կապված մուտքերի գաղափարները սահմանվում են հայտնի եղանակով: M թերմի բոլոր ազատ փոփոխականների բազմությունը կնշանակենք $FV(M)$-ով: Հայտնի եղանակով է սահմանվում նաև երկու թերմերի կոնգրուենտության ($\equiv$), β-ռեդուկցիայի գաղափար, միաքայլ β-ռեդուկցիա ($\longrightarrow_\beta$), β-ռեդուկցիա ($\twoheadrightarrow_\beta$), β-հավասարություն ($=_\beta$) հարաբերություններն ու β-ռեդեքս, β-ռեդեքսի փաթեթ, β-նորմալ ձև գաղափարները:

Սահմանում 1.1.7 $f = M$ կառույցը, որտեղ $f \in TermVariable$, $M \in Term$, կանվանենք հավասարում: $P = \{f_i = M_i \mid i = 1, \dots, n\}$ հավասարումների բազմությունը, որտեղ $f_i \neq f_j$, երբ $i \neq j$, $i, j = 1, \dots, n$, $n \geq 0$, կանվանենք հավասարումների համակարգ: Այն կանվանենք փակ համակարգ, եթե $FV(M_i) \subset \{f_1, \dots, f_n\}$, $i = 1, \dots, n$:

Սահմանում 1.1.8 $\Sigma: Constant \to Type$ արտապատկերումը կոչվում է հաստատունների աղյուսակ: Իմաստային առումով հաստատունների աղյուսակում պահվում են տվյալ ծրագրավորման լեզվում օգտագործվող թերմերի հաստատուններին վերագրված տիպերը: Այսուհետև, որպեսզի ամեն անգամ չնշենք հաստատունների աղյուսակը, կենթադրենք, որ այն ֆիքսված է:

1.2 բաժնում ներկայացվում են տիպերի արտածման կանոնները, իսկ թեորեմ 1.2.1-ում ապացուցվում է, որ եթե տրված հավասարումների համակարգն ու թերմը տիպայնացվող են միևնույն միջավայրում, ապա այդ մուտքի վրա ցանկացած ինտերպրետացիայի ալգորիթմի աշխատանքի ընթացքում ստացվող թերմերը չեն պարունակի տիպի սխալ:

Սահմանում 1.2.1 Դիցուք $x_1, \dots, x_n \in TermVariable$, $\tau_1, \dots, \tau_n \in Type$, որտեղ $n \geq 0$: Այդ դեպքում $H = < x_1 : \tau_1, \dots, x_n : \tau_n >$ հաջորդականությունը կանվանենք միջավայր: Եթե $x \in \{x_1, \dots, x_n\}$, ապա կասենք, որ $x \in H$: Կատարենք նաև հետևյալ նշանակումները՝

1. $H + x' : \tau' = < x_1 : \tau_1, \dots, x_n : \tau_n, x' : \tau' >$, որտեղ $x' \in TermVariable$ և $\tau' \in Type$,
2. $[\sigma]H = < x_1 : [\sigma]\tau_1, \dots, x_n : [\sigma]\tau_n >$, որտեղ σ-ն տեղադրություն է,
3. եթե $x \in H$, $x_k = x$, $x_j \neq x$, որտեղ $j = k + 1, \dots, n$ և $k \in \{1, \dots, n\}$, ապա $H(x) = \tau_k$:

Սահմանում 1.2.2 Միջավայր, թերմ, տիպ եռյակը կանվանենք դատողություն: Այն կգրենք հետևյալ տեսքով՝ $H \vdash M:\tau$ և կկարդանք՝ H միջավայրում M թերմն ունի τ տիպ:

Սահմանում 1.2.3 Դատողությունների արտածման կանոնները հետևյալն են՝

$(variable)$ $\quad \frac{}{H \vdash x:H(x)}$, որտեղ $x \in H$,

$(constant)$ $\quad \frac{}{H \vdash s:[\sigma]\Sigma(s)}$, որտեղ $s \in Constant$ և σ-ն տեղադրություն է,

$(abstraction)$ $\quad \frac{H + x_1:\tau_1 + \dots + x_n:\tau_n \vdash M:\tau}{H \vdash (\lambda x_1 \dots x_n.M):(\tau_1 \times \dots \times \tau_n \to \tau)}$,

$(application)$ $\quad \frac{H \vdash F:(\tau_1 \times \dots \times \tau_n \to \tau),\ H \vdash M_i:\tau_i,\ i=1,\dots,n}{H \vdash F(M_1,\dots,M_n):\tau}$:

Սահմանում 1.2.4 Կասենք, որ τ տիպը տիպայնացնում է M թերմը, եթե գոյություն ունի H միջավայր այնպիսին, որ $H \vdash M:\tau$ դատողությունն արտածվում է: Կասենք, որ M թերմը տիպայնացվող է, եթե գոյություն ունի այն տիպայնացնող տիպ: Կասենք, որ M թերմը տիպայնացվող է H միջավայրում, եթե գոյություն ունի τ տիպ այնպիսին, որ $H \vdash M:\tau$ դատողությունն արտածվում է: Կասենք, որ M թերմը պարունակում է տիպի սխալ, եթե $\perp$-ն հանդիսանում է նրա ենթաթերմ:

Սահմանում 1.2.5 Կասենք, որ $\{f_i = M_i \mid i = 1, \dots, n\}$ հավասարումների համակարգը տիպայնացվող է H միջավայրում, եթե $H \vdash M_i:H(f_i)$ դատողություններն արտածվում են, որտեղ $i = 1, \dots, n$: Կասենք, որ հավասարումների համակարգը տիպայնացվող է, եթե գոյություն ունի H միջավայր այնպիսին, որ այդ միջավայրում այն տիպայնացվող է:

Սահմանում 1.2.6 $\delta \subset Term^2$ հարաբերությունը կանվանենք δ-ռեդուկցիայի գաղափար, եթե այն բավարարում է հետևյալ պայմաններին՝

1. եթե $(M, M') \in \delta$ և $(M, M'') \in \delta$, ապա $M' \equiv M''$,
2. եթե $(M, M') \in \delta$, ապա $M \equiv c(M_1, \dots, M_k)$, որտեղ $c \in Constant$, $M_1, \dots, M_k \in Term$, $k \geq 1$ կամ $M \equiv (\lambda x_1 \dots x_n.M'_0)(M'_1, \dots, M'_m)$, որտեղ $x_1, \dots, x_n \in TermVariable$, $M'_0, M'_1, \dots, M'_m \in Term$, $n \geq 1$, $m \geq 1$, $n \neq m$,
3. եթե $x_1, \dots, x_n \in TermVariable$, $M, M_1, \dots, M_k \in Term$, $n \geq 1$, $k \geq 1$ և $n \neq k$, ապա $((\lambda x_1 \dots x_n.M)(M_1, \dots, M_k), \perp) \in \delta$,
4. եթե $(c(M_1, \dots, M_k), M') \in \delta$ և արտածվում է $H \vdash c(M_1, \dots, M_k) : \tau$ դատողությունը, որտեղ $\tau \in Type$, $c \in Constant$, $M_1, \dots, M_k \in Term$, $k \geq 1$, ապա արտածվում է նաև $H \vdash M' : \tau$ դատողությունը,
5. եթե գոյություն չունի $c(M_1, \dots, M_k)$ թերմը տիպայնացնող տիպ, որտեղ $c \in Constant$, $M_1, \dots, M_k \in Term$, $k \geq 1$, ապա $(c(M_1, \dots, M_k), \perp) \in \delta$,

6. գոյություն ունի ալգորիթմ, որը, մուտքում ստանալով $c(M_1,\dots,M_k)$ թերմը, որտեղ $c \in Constant$, $M_1,\dots,M_k \in Term$, $k \geq 1$, վերադարձնում է M' թերմ այնպիսին, որ $(c(M_1,\dots,M_k),M') \in \delta$ կամ վերադարձնում է ոչ, եթե $\nexists M'$ թերմ այնպիսին, որ $(c(M_1,\dots,M_k),M') \in \delta$:

Հայտնի եղանակով սահմանվում են նաև միաքայլ δ-ռեդուկցիա ($\longrightarrow_\delta$), δ-ռեդուկցիա ($\twoheadrightarrow_\delta$), δ-հավասարություն ($=_\delta$) հարաբերություններն ու δ-ռեդեքս, δ-ռեդեքսի փաթեթ, δ-նորմալ ձև գաղափարները: Այս սահմանման մեջ նշված են միայն մեր կողմից օգտագործվող հատկությունները: δ-ռեդուկցիայի գաղափարն իրականում պետք է օժտված լինի նաև այլ հատկություններով, որոնք անհրաժեշտ են մի շարք կարևոր պնդումներ ապացուցելու համար՝ մասնավորապես $\beta\delta$-նորմալ ձևի միակությունն ապացուցելու համար:

Սահմանում 1.2.7 Դիցուք δ-ն որևէ δ-ռեդուկցիայի գաղափար է: Այդ դեպքում $\beta \cup \delta$ հարաբերությունը կանվանենք $\beta\delta$-ռեդուկցիայի գաղափար: Միաքայլ $\beta\delta$-ռեդուկցիա ($\longrightarrow_{\beta\delta}$), $\beta\delta$-ռեդուկցիա ($\twoheadrightarrow_{\beta\delta}$), $\beta\delta$-հավասարություն ($=_{\beta\delta}$) հարաբերություններն ու $\beta\delta$-ռեդեքս, $\beta\delta$-ռեդեքսի փաթեթ, $\beta\delta$-նորմալ ձև գաղափարները սահմանվում են հայտնի եղանակով:

Սահմանում 1.2.8 A ինտերպրետացիայի ալգորիթմը, մուտքում ստանալով M թերմը և $P = \{f_i = M_i \mid i = 1,\dots,n\}$ հավասարումների համակարգը, վերադարձնում է M' թերմ այնպիսին, որ M'-ը $\beta\delta$-նորմալ ձև է և $FV(M') \cap \{f_1,\dots,f_n\} = \emptyset$ կամ աշխատում է անվերջ: Ընդ որում, սկսելով M թերմից, A ալգորիթմը կատարում է միայն հետևյալ երեք տիպի գործողություններ՝

1. թերմում $f_1,\dots,f_n$ փոփոխականների որոշ ազատ մուտքերի փոխարինում համապատասխանաբար $M_1,\dots,M_n$ թերմերով,
2. թերմում միաքայլ β-ռեդուկցիայի կատարում,
3. թերմում միաքայլ δ-ռեդուկցիայի կատարում:

Թեորեմ 1.2.1 Դիցուք $M \in Term$, $P = \{f_i = M_i \mid i = 1,\dots,n\}$-ն հավասարումների համակարգ է, A-ն ինչ-որ ինտերպրետացիայի ալգորիթմ է, իսկ H-ը միջավայր է: Այդ դեպքում եթե արտածվում են հետևյալ դատողությունները՝ $H \vdash M_i : H(f_i)$, $i = 1,\dots,n$ և $H \vdash M : \tau$, որտեղ τ-ն ինչ-որ տիպ է, ապա M և P մուտքի համար A ինտերպրետացիայի ալգորիթմի աշխատանքի ընթացքում ստացվող թերմերը չեն պարունակի տիպի սխալ:

1.3 բաժնում ներկայացվում են երկու տիպերի ունիֆիկատորի և ամենաընդհանուր ունիֆիկատորի գաղափարները, երկու տիպերի ունիֆիկացման ալգորիթմը, իսկ լեմմա 1.3.1-ում ապացուցվում է, որ ունիֆիկացվող տիպերի դեպքում այդ ալգորիթմը վերադարձնում է տրված տիպերի ամենաընդհանուր ունիֆիկատորը, իսկ այն տիպերի դեպքում, որոնք ունիֆիկացվող չեն՝ վերադարձնում է No:

Սահմանում 1.3.1 Դիցուք $\tau_1, \tau_2 \in Type$: σ տեղադրությանը կանվանենք τ_1 և τ_2 տիպերի ունիֆիկատոր, եթե $[\sigma]\tau_1 = [\sigma]\tau_2$: $Unifiers(\tau_1, \tau_2)$-ով կնշանակենք τ_1 և τ_2 տիպերի բոլոր ունիֆիկատորների բազմությունը:

Սահմանում 1.3.2 Դիցուք $\tau_1, \tau_2 \in Type$: Կասենք, որ τ_1 և τ_2 տիպերը ունիֆիկացվող են, եթե $Unifiers(\tau_1, \tau_2) \neq \emptyset$, այսինքն՝ գոյություն ունի τ_1 և τ_2 տիպերի ունիֆիկատոր:

Սահմանում 1.3.3 $\sigma \in Unifiers(\tau_1, \tau_2)$ տեղադրությանը կանվանենք τ_1 և τ_2 տիպերի ամենաընդհանուր ունիֆիկատոր, եթե ցանկացած $\sigma' \in Unifiers(\tau_1, \tau_2)$ ունիֆիկատորի համար գոյություն ունի σ'' տեղադրություն այնպիսին, որ $[\sigma']\tau_1 = [[\sigma'']\sigma]\tau_1$ և $[\sigma']\tau_2 = [[\sigma'']\sigma]\tau_2$: $Mgu(\tau_1, \tau_2)$-ով նշանակենք τ_1 և τ_2 տիպերի բոլոր ամենաընդհանուր ունիֆիկատորների բազմությունը:

Այնուհետև ներկայացվում է երկու տիպերի ունիֆիկացման $Unify$ ալգորիթմը, որը Ռոբինսոնի ունիֆիկացման ալգորիթմի ձևափոխված և մեր տիպերին հարմարեցված տարբերակն է: $Unify$ ալգորիթմը մուտքում ստանում է երկու տիպ և վերադարձնում է տեղադրություն կամ No:

Լեմմա 1.3.1 Դիցուք $\tau_1, \tau_2 \in Type$: Եթե τ_1 և τ_2 տիպերն ունիֆիկացվող են, ապա $Unify(\tau_1, \tau_2) = \sigma \in Mgu(\tau_1, \tau_2)$ և $\sigma = [\sigma]\sigma$: Իսկ եթե τ_1 և τ_2 տիպերն ունիֆիկացվող չեն, ապա $Unify(\tau_1, \tau_2) = No$:

1.4 բաժնում ներկայացվում է կառուցվածքային λ-թերմերի տիպայնացման $Typify$ ալգորիթմը, թեորեմ 1.4.1-ում և թեորեմ 1.4.2-ում ապացուցվում է, որ հաջող ավարտի դեպքում այդ ալգորիթմը դուրս է բերում տրված թերմի ամենաընդհանուր տիպը, իսկ անհաջող ավարտ կարող է լինել միայն այդ դեպքում, երբ տրված թերմը տիպայնացվող չէ:

Թերմերի տիպայնացման $Typify$ ալգորիթմը մուտքում ստանում է միջավայր ու թերմ և վերադարձնում է տեղադրության ու տիպի զույգ կամ No:

Թեորեմ 1.4.1 Դիցուք ունենք H միջավայրը և M թերմը: H և M մուտքի համար $Typify$ ալգորիթմը վերջավոր քայլերից հետո վերադարձնում է տեղադրության և տիպի զույգ կամ No: Ընդ որում, եթե $Typify(H, M) = (\sigma, \tau)$, ապա $[\sigma]H \vdash M : \tau$-ն արտածվում է:

Թեորեմ 1.4.2 Եթե $[\sigma]H \vdash M : \tau$-ն արտածվում է, ապա $Typify(H, M) = (\sigma', \tau')$ և գոյություն ունի σ'' տեղադրություն այնպիսին, որ $[\sigma]H = [\sigma'']\sigma']H$ և $\tau = [\sigma'']\tau'$:

1.5 բաժնում ներկայացվում է կառուցվածքային λ-թերմերն օգտագործող հավասարումների համակարգի տիպայնացման $Typify$ ալգորիթմը, թեորեմ 1.5.1-ում և թեորեմ 1.5.2-ում ապացուցվում է, որ հաջող ավարտի դեպքում այդ ալգորիթմը տիպայնացնում է տրված համակարգը ամենաընդհանուր ձևով, իսկ անհաջող ավարտ կարող է լինել միայն այն դեպքում, երբ տրված հավասարումների համակարգը տիպայնացվող չէ:

Հավասարումների համակարգի տիպայնացման $Typify$ ալգորիթմը մուտքում ստանում է միջավայր ու համակարգ և վերադարձնում է տեղադրություն կամ No:

Թեորեմ 1.5.1 Դիցուք ունենք H միջավայրը և P հավասարումների համակարգը: H և P մուտքի համար $Typify$ ալգորիթմը վերջավոր քայլերից հետո վերադարձնում է տեղադրություն կամ No: Ընդ որում, եթե $Typify(H,P)=\sigma$, ապա P համակարգը $[\sigma]H$ միջավայրում տիպայնացվող է:

Թեորեմ 1.5.2 Դիցուք P-ն հավասարումների համակարգ է: Եթե բոլոր $f'=M'\in P$ հավասարումների համար $[\sigma]H\vdash M':[\sigma]H(f')$ դատողությունն արտածվում է, ապա $Typify(H,P)=\sigma'$ և գոյություն ունի σ'' տեղադրություն այնպիսին, որ $[\sigma]H=[\sigma'']{[\sigma']H}$:

Գլուխ 2-ը նվիրված է հաստատուններ պարունակող λ-թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի համար ընդլայնված SYSTEM E համակարգին ու տիպայնացման ալգորիթմներին և դրանց հիմնավորմանը: Երկրորդ գլուխը բաղկացած է հինգ բաժնից՝ 2.1-2.5:

2.1 բաժնում ներկայացվում են երկրորդ գլխում օգտագործվող սահմանումները: Դիցուք ունենք $TypeVariable$ տիպերի փոփոխականների հաշվելի բազմությունը, $(ExpansionVariable,\leqslant)$ ընդարձակող փոփոխականների հաշվելի և լրիվ կարգավորված բազմությունը և $TypeConstant$ տիպերի հաստատունների վերջավոր բազմությունը:

Սահմանում 2.1.1 Տիպերի $Type$ բազմությունը սահմանվում է ինդուկտիվ հետևյալ ձևով՝

1. $\omega\in Type$ և կոչվում է անորոշ տիպ,
2. եթե $\alpha\in TypeVariable$, ապա $\alpha\in Type$,
3. եթե $s\in TypeConstant$, ապա $s\in Type$,
4. եթե $e\in ExpansionVariable$ և $\tau\in Type$, ապա $e\tau\in Type$,
5. եթե $\tau_1\in Type$ և $\tau_2\in Type$, ապա $(\tau_1\to\tau_2)\in Type$,
6. եթե $\tau_1\in Type$ և $\tau_2\in Type$, ապա $(\tau_1\cap\tau_2)\in Type$:

Ընդ որում, բավարարում են հետևյալ պայմանները՝

1. $(\tau_1\cap(\tau_2\cap\tau_3))=((\tau_1\cap\tau_2)\cap\tau_3)$ 2. $(\tau_1\cap\tau_2)=(\tau_2\cap\tau_1)$ 3. $(\omega\cap\tau)=\tau$
4. $e(\tau_1\cap\tau_2)=(e\tau_1\cap e\tau_2)$ 5. $e\omega=\omega$,

որտեղ $\tau_1,\tau_2,\tau_3,\tau\in Type$ և $e\in ExpansionVariable$:

Սահմանում 2.1.2 Ընդարձակիչների $Expansion$ բազմությունը սահմանվում է ինդուկտիվ հետևյալ ձևով՝

1. $\omega\in Expansion$ և կոչվում է անորոշ ընդարձակիչ,

2. եթե σ-ն տեղադրություն է (տեղադրության սահմանումը կտանք ստորև), ապա $\sigma \in Expansion$,

3. եթե $e \in ExpansionVariable$ և $E \in Expansion$, ապա $eE \in Expansion$,

4. եթե $E_1 \in Expansion$ և $E_2 \in Expansion$, ապա $(E_1 \cap E_2) \in Expansion$:

Սահմանում 2.1.3 Դիցուք $\alpha_1, \dots, \alpha_n \in TypeVariable$, $\tau_1, \dots, \tau_n \in Type$, $e_1, \dots, e_m \in ExpansionVariable$, $E_1, \dots, E_m \in Expansion$, $n, m \geq 0$: Այդ դեպքում $\sigma = \{\alpha_1 := \tau_1, \dots, \alpha_n := \tau_n, e_1 := E_1, \dots, e_m := E_m\}$ զույգերի բազմությունը կոչվում է տեղադրություն, եթե տեղի ունեն հետևյալ պայմանները՝

1. եթե $i \neq j$, ապա $\alpha_i \neq \alpha_j$, որտեղ $i, j = 1, \dots, n$,

2. եթե $i \neq j$, ապա $e_i \neq e_j$, որտեղ $i, j = 1, \dots, m$:

Սահմանում 2.1.4 Դիցուք $e_1, \dots, e_n \in ExpansionVariable$, որտեղ $n \geq 0$: Այդ դեպքում $e_1 e_2 \dots e_n$-ը կանվանենք E-ճանապարհ և կնշանակենք $\vec{e}$-ով՝ $\vec{e} = e_1 e_2 \dots e_n$:

Սահմանում 2.1.6 Սահմանափակումների $Constraint$ բազմությունը սահմանվում է ինդուկտիվ հետևյալ ձևով՝

1. $\omega \in Constraint$ և կոչվում է անորոշ սահմանափակում,

2. եթե $\tau_1 \in Type$ և $\tau_2 \in Type$, ապա $(\tau_1 \doteq \tau_2) \in Constraint$,

3. եթե $e \in ExpansionVariable$ և $\Delta \in Constraint$, ապա $e\Delta \in Constraint$,

4. եթե $\Delta_1 \in Constraint$ և $\Delta_2 \in Constraint$, ապա $(\Delta_1 \cap \Delta_2) \in Constraint$:

Ընդ որում, բավարարում են հետևյալ պայմանները՝

1. $(\Delta_1 \cap (\Delta_2 \cap \Delta_3)) = ((\Delta_1 \cap \Delta_2) \cap \Delta_3)$ 2. $(\Delta_1 \cap \Delta_2) = (\Delta_2 \cap \Delta_1)$ 3. $(\omega \cap \Delta) = \Delta$

4. $e(\Delta_1 \cap \Delta_2) = (e\Delta_1 \cap e\Delta_2)$ 5. $e\omega = \omega$,

որտեղ $\Delta_1, \Delta_2, \Delta_3, \Delta \in Constraint$ և $e \in ExpansionVariable$:

Սահմանում 2.1.7 Կասենք որ Δ սահմանափակումը լուծված է, եթե այն ունի հետևյալ տեսքը՝ $\Delta = \vec{e}_1(\tau_1 \doteq \tau_1) \cap \dots \cap \vec{e}_n(\tau_n \doteq \tau_n)$, $n \geq 1$ կամ $\Delta = \omega$: $unsolved(\Delta)$-ով կնշանակենք Δ–ի այն ամենափոքր մասը (ամենափոքր ասելով՝ այստեղ հասկանում ենք ամենաքիչ $\cap$ գործողություն պարունակողը) այնպիսին, որ $\Delta = unsolved(\Delta) \cap \Delta'$ և Δ'–ը լուծված է: Ստացվում է, որ Δ' սահմանափակումն էլ Δ-ի ամենամեծ լուծված մասն է: Վերջինս էլ կնշանակենք $solved(\Delta)$:

Դիցուք ունենք $TermVariable$ թերմի փոփոխականների հաշվելի բազմությունը և $Constant$ թերմի հաստատունների հաշվելի բազմությունը:

Սահմանում 2.1.8 Թերմերի $Term$ բազմությունը սահմանվում է ինդուկտիվ հետևյալ ձևով՝

1. $\perp \in Term$ և $\perp$-ին անվանում են տիպի սխալը ներկայացնող թերմ,

2. եթե $x \in TermVariable$, ապա $x \in Term$,

3. եթե $c \in Constant$, ապա $c \in Term$,

4. եթե $x \in TermVariable$ և $M \in Term$, ապա $(\lambda x M) \in Term$,

5. Եթե $M_1 \in Term$ և $M_2 \in Term$, ապա $(M_1 M_2) \in Term$:

Հայտնի եղանակով է սահմանվում երկու թերմերի կոնգրուենտության ($\equiv$) գաղափարը: Թերմի ազատ փոփոխականի գաղափարը և թերմում փոփոխականի ազատ և կապված մուտքերի գաղափարները նույնպես սահմանվում են հայտնի եղանակով: M թերմի բոլոր ազատ փոփոխականների բազմությունը կնշանակենք $FV(M)$-ով: Հայտնի եղանակով են սահմանվում նաև β-ռեդուկցիայի գաղափար, միաքայլ β-ռեդուկցիա ($\longrightarrow_\beta$), β-ռեդուկցիա ($\twoheadrightarrow_\beta$), β-հավասարություն ($=_\beta$) հարաբերություններն ու β-ռեդեքս, β-ռեդեքսի փաթեթ, β-նորմալ ձև գաղափարները: Բոլոր β-նորմալ ձևերի բազմությունը կնշանակենք β-NF-ով:

Սահմանում 2.1.9 Կմախքների $Skeleton$ բազմությունը սահմանվում է ռեկուրսիվ հետևյալ ձևով՝

1. եթե $M \in Term$, ապա $\omega^M \in Skeleton$,

2. եթե $c \in Constant$ և $\tau \in Type$, ապա $c^{:\tau} \in Skeleton$,

3. եթե $x \in TermVariable$ և $\tau \in Type$, ապա $x^{:\tau} \in Skeleton$,

4. եթե $e \in ExpansionVariable$ և $Q \in Skeleton$, ապա $eQ \in Skeleton$,

5. եթե $x \in TermVariable$ և $Q \in Skeleton$, ապա $(\lambda x.\, Q) \in Skeleton$,

6. եթե $Q_1 \in Skeleton$ և $Q_2 \in Skeleton$, ապա $(Q_1 \cap Q_2) \in Skeleton$,

7. եթե $Q_1 \in Skeleton$ և $Q_2 \in Skeleton$ և $\tau \in Type$, ապա $(Q_1 Q_2)^{:\tau} \in Skeleton$:

2.1.12 և 2.1.13 սահմանումներում սահմանվում է, թե ինչ է նշանակում կիրառել ընդարձակիչը տիպի, կմախքի, սահմանափակման կամ ընդարձակիչի վրա: X օբյեկտի վրա E ընդարձակիչի կիրառումը նշանակվում է $[E]X$-ով:

Սահմանում 2.1.14 $CType \subset Type$ բազմությունը սահմանվում է ինդուկտիվ հետևյալ ձևով՝

1. եթե $s \in TypeConstant$, ապա $s \in CType$,

2. եթե $s \in TypeConstant$ և $\tau \in CType$, ապա $(s \to \tau) \in CType$:

Սահմանում 2.1.15 $\Sigma: Constant \to CType$ արտապատկերումը կոչվում է հաստատունների աղյուսակ: Իմաստային առումով հաստատունների աղյուսակում պահվում են տվյալ ծրագրավորման լեզվում օգտագործվող թերմերի հաստատուններին վերագրված տիպերը: Այսուհետև, որպեսզի ամեն անգամ չնշենք հաստատունների աղյուսակը, կենթադրենք, որ այն ֆիքսված է:

2.2 բաժնում ներկայացվում են տիպերի արտածման կանոնները, իսկ թեորեմ 2.2.1-ում ապացուցվում է, որ «նորմալ» տիպայնացվող թերմի համար մեր կողմից դիտարկվող ռեդուկցիոն ստրատեգիաների աշխատանքի ընթացքում տիպի սխալ տեղի չի ունենում:

Սահմանում 2.2.1 $A: TermVariable \to Type$ արտապատկերումը կոչվում է միջավայր, եթե $\{x \mid x \in TermVariable \text{ և } A(x) \neq \omega\}$ բազմությունը պարունակում է վերջավոր թվով էլեմենտներ: Այսինքն՝ միջավայրը միայն վերջավոր թվով թերմի փոփոխականների է արտապատկերում ω-ից տարբեր տիպերի վրա: Այն կարելի է գրել զույգերի բազմության տեսքով նաև՝ $A = \{(x, A(x)) \mid x \in TermVariable\}$: Գատարենք հետևյալ նշանակումները՝

1. $A[x \to \tau] = \{(y, A(y)) \mid y \in TermVariable \text{ և } y \neq x\} \cup \{(x, \tau)\}$,
2. $A \cap B = \{(x, (A(x) \cap B(x))) \mid x \in TermVariable\}$,
3. $eA = \{(x, eA(x)) \mid x \in TermVariable\}$,
4. $[E]A = \{(x, [E]A(x)) \mid x \in TermVariable\}$,
5. $env_\omega = \{(x, \omega) \mid x \in TermVariable\}$,

որտեղ A, B-ն միջավայրեր են, $e \in ExpansionVariable$, $x \in TermVariable$, $\tau \in Type$ և $E \in Expansion$:

Սահմանում 2.2.2 Թերմ, կմախք, միջավայր, տիպ, սահմանափակում հնգյակը կանվանենք դատողություն: Այն կգրենք հետևյալ տեսքով՝ $(M \rhd Q) : (A \vdash \tau)/\Delta$: Դատողության իմաստը կայանում է հետևյալում՝ Q-ն հանդիսանում է ապացույցն այն բանի, որ $(A \vdash \tau)$ զույգը տիպայնացնում է M թերմը՝ պայմանով, որ Δ-ն լուծված է:

Սահմանում 2.2.3 Դատողությունների արտածման կանոնները հետևյալն են՝

$(variable)$ $\dfrac{}{(x \rhd x^{:\tau}) : (env_\omega[x \to \tau] \vdash \tau)/\omega}$

$(constant)$ $\dfrac{}{(c \rhd c^{:\tau}) : (env_\omega \vdash \tau)/\omega}$, որտեղ $\tau = \Sigma(c)$

$(omega)$ $\dfrac{}{(M \rhd \omega^M) : (env_\omega \vdash \omega)/\omega}$, որտեղ M-ը չի պարունակում $\bot$ ենթաթերմ

$(E - variable)$ $\dfrac{(M \rhd Q) : (A \vdash \tau)/\Delta}{(M \rhd eQ) : (eA \vdash e\tau)/e\Delta}$

$(abstraction)$ $\dfrac{(M \rhd Q) : (A \vdash \tau)/\Delta}{((\lambda x.M) \rhd (\lambda x.Q)) : (A[x \to \omega] \vdash (A(x) \to \tau))/\Delta}$

$(application)$ $\dfrac{(M_1 \rhd Q_1) : (A_1 \vdash \tau_1)/\Delta_1,\ \ (M_2 \rhd Q_2) : (A_2 \vdash \tau_2)/\Delta_2}{((M_1 M_2) \rhd (Q_1 Q_2)^{:\tau}) : (A_1 \cap A_2 \vdash \tau)/\Delta_1 \cap \Delta_2 \cap (\tau_1 \doteq (\tau_2 \to \tau))}$

$(intersection)$ $\dfrac{(M \rhd Q_1) : (A_1 \vdash \tau_1)/\Delta_1,\ \ (M \rhd Q_2) : (A_2 \vdash \tau_2)/\Delta_2}{(M \rhd (Q_1 \cap Q_2)) : (A_1 \cap A_2 \vdash (\tau_1 \cap \tau_2))/\Delta_1 \cap \Delta_2}$:

Տրվում են նաև «թերմը տիպայնացվող է», «թերմը պարունակում է տիպի սխալ», «նորմալ տիպայնացում» հասկացությունների և թերմը տիպայնացնող միջավայրի ու տիպի զույգի սահմանումները:

Սահմանում 2.2.7 Կասենք, որ $(A \vdash \tau)$ զույգը M թերմի գլխավոր տիպայնացումն է, եթե այն տիպայնացնում է M թերմը, և M թերմը տիպայնացնող կամայական $(A' \vdash \tau')$ զույգի համար գոյություն ունի E ընդարձակիչ այնպիսին, որ $A' = [E]A$ և $\tau' = [E]\tau$:

1.2.6 և 1.2.7 սահմանմանումներին նմանատիպ եղանակով սահմանվում են նաև δ-ռեդուկցիայի գաղափարը և $\beta\delta$-ռեդուկցիայի գաղափարը:

Սահմանում 2.2.13 $R: Term \to Term$ արտապատկերումը կոչվում է $\beta\delta$-ռեդուկցիոն ստրատեգիա, եթե ցանկացած M թերմի համար $M \twoheadrightarrow_{\beta\delta} R(M)$: Ընդ որում, եթե M թերմը $\beta\delta$-նորմալ ձև չէ, ապա գոյություն ունի M_1 թերմ այնպիսին, որ $M \to_{\beta\delta} M_1 \twoheadrightarrow_{\beta\delta} R(M)$: R ստրատեգիան կանվանենք միաքայլ, եթե ցանկացած M թերմի համար, որը $\beta\delta$-նորմալ ձև չէ, $M \longrightarrow_{\beta\delta} R(M)$:

Տրվում են նաև նորմալ կմախքի և $\beta\delta$-ռեդուկցիոն ստրատեգիաների (*) հատկության սահմանումները: Երկրորդ գլխում դիտարկվում են միայն այնպիսի $\beta\delta$-ռեդուկցիոն ստրատեգիաներ, որոնք բավարարում են (*) հատկությանը:

Թեորեմ 2.2.1 Դիցուք $M \in Term$, R-ը որևէ $\beta\delta$-ռեդուկցիոն ստրատեգիա է և Q նորմալ կմախքը M թերմի տիպայնացման կմախք է: Այդ դեպքում M թերմի համար R ստրատեգիայի աշխատանքի ընթացքում ստացվող թերմերը չեն պարունակի տիպի սխալ:

2.3 բաժնում ներկայացվում են ունիֆիկացման կանոնները, որոնք են $unify_\beta$, $unify_x$ և $unify_c$: Այնուհետև ներկայացվում է ունիֆիկացման $Unify$ ալգորիթմը, որն օգտագործելով վերը նշված երեք ունիֆիկացման կանոնները, փորձում է լուծել տրված սահմանափակումը: Այդ ալգորիթմը մուտքում ստանում է չլուծված սահմանափակում և վերադարձնում է այն լուծող տեղադրություն, կամ ավարտում է աշխատանքն անհաջողությամբ, կամ էլ աշխատում է անվերջ:

2.4 բաժնում ներկայացվում է հաստատուններ պարունակող թերմերի տիպայնացման $Typify$ ալգորիթմը, իսկ թեորեմ 2.4.1-ում ապացուցվում է, որ հաջող ավարտի դեպքում ներկայացված ալգորիթմը դուրս է բերում տրված թերմի գլխավոր տիպայնացումը:

$Typify$ ալգորիթմը մուտքում ստանում է թերմ և վերադարձնում է տրված թերմը տիպայնացնող միջավայրի և տիպի զույգ, կամ ավարտում է աշխատանքն անհաջողությամբ, կամ էլ աշխատում է անվերջ:

Թեորեմ 2.4.1 Դիցուք $M \in Term$ և գոյություն ունի M' թերմ այնպիսին, որ $M \twoheadrightarrow_\beta M'$ և $M' \in \beta\text{-}NF$: Այդ դեպքում տեղի ունի հետևյալը՝

1. եթե գոյություն ունի M'-ի այնպիսի տիպայնացում, որի արտածման ժամանակ չի օգտագործվում $(omega)$ կանոնը, ապա $Typify(M)$ կանչն ավարտվում է հաջողությամբ,
2. եթե $Typify(M) = (A \vdash \tau)$, ապա $(A \vdash \tau)$ զույգը M-ի գլխավոր տիպայնացումն է:

2.5 բաժնում ներկայացվում է հաստատուններ պարունակող թերմերով կազմված հավասարումների համակարգի տիպայնացման $Typify$ ալգորիթմը, թեորեմ 2.5.1-ում ապացուցվում է, որ «միասին և նորմալ» տիպայնացվող հավասարումների համակարգի և թերմի համար մեր կողմից դիտարկվող ինտերպրետացիայի ալգորիթմների աշխատանքի ընթացքում տիպի սխալ տեղի չի ունենում, իսկ թեորեմ 2.5.2-ում ապացուցվում է, որ հաջող ավարտի դեպքում վերը նշված ալգորիթմն իրոք տիպայնացնում է տրված հավասարումների համակարգը:

Հավասարումների համակարգը սահմանվում է 1.1.7 սահմանման համաձայն: Ինտերպրետացիայի ալգորիթմը սահմանվում է 1.2.8 սահմանման համաձայն: Տրվում են նաև կմախքների համատեղելիության, համակարգը նորմալ տիպայնացնող կմախքների և ինտերպրետացիայի ալգորիթմների (**) հատկության սահմանումները: Երկրորդ գլխում դիտարկվում են միայն այնպիսի ինտերպրետացիայի ալգորիթմներ, որոնք բավարարում են (**) հատկությանը:

Թեորեմ 2.5.1 Դիցուք $M \in Term$, $P = \{f_i = M_i \mid i = 1, \dots, n\}$-ն հավասարումների համակարգ է, A-ն ինտերպրետացիայի ալգորիթմ է, $Q_1, \dots, Q_n$-ը P համակարգի համար նորմալ տիպայնացում է և գոյություն ունի $Q_1, \dots, Q_n$-ի հետ համատեղելի նորմալ Q կմախք այնպիսին, որ այն M-ի տիպայնացման կմախք է: Այդ դեպքում M և P մուտքի համար A ալգորիթմի աշխատանքի ընթացքում ստացվող թերմերը չեն պարունակի տիպի սխալ:

Հավասարումների համակարգի տիպայնացման $Typify$ ալգորիթմը մուտքում ստանում է հավասարումների համակարգ և վերադարձնում է այն տիպայնացնող կմախքները, կամ աշխատում է անվերջ, կամ էլ ավարտում է աշխատանքն անհաջողությամբ:

Թեորեմ 2.5.2 Դիցուք ունենք $P = \{f_i = M_i \mid i = 1, \dots, n\}$ համակարգը: Այդ դեպքում, եթե $Typify(P) = (Q_1, \dots, Q_n)$, ապա $Q_1, \dots, Q_n$-ը P-ի նորմալ տիպայնացում է:

Գլուխ 3-ը նվիրված է ֆունկցիոնալ ծրագրերի տիպային կոռեկտության խնդրի համար ստացված անլուծելիության արդյունքներին: Երրորդ գլուխը բաղկացած է երկու բաժնից՝ 3.1-3.2:

3.1 բաժնում ներկայացվում են երրորդ գլխում օգտագործվող սահմանումները և այն դասական արդյունքները, որոնք ընկած են 3.2 բաժնում ստացված անլուծելիության արդյունքների ապացուցման հիմքում:

Թերմերի $Term$ բազմությունը սահմանվում է համաձայն 2.1.8 սահմանման: Հայտնի եղանակով սահմանվում է խիստ β-նորմալիզացվող թերմի գաղափարը: 1.2.6 և 1.2.7 սահմանումներին նմանատիպ եղանակով սահմանվում են նաև δ-ռեդուկցիայի գաղափարը և $\beta\delta$-ռեդուկցիայի գաղափարը, իսկ ռեդուկցիոն ստրատեգիան սահմանվում է 2.2.13 սահմանման համաձայն:

Սահմանում 3.1.3 $TermI \subset Term$ բազմությունը սահմանվում է ինդուկտիվ հետևյալ ձևով՝

1. եթե $x \in TermVariable$, ապա $x \in TermI$,

2. եթե $x \in TermVariable$, $M \in TermI$ և $x \in FV(M)$, ապա $\lambda x.M \in TermI$,

3. եթե $M_1, M_2 \in TermI$, ապա $M_1 M_2 \in TermI$:

Թեորեմ 3.1.1 (Church, Rosser) Դիցուք $M \in TermI$: Եթե M թերմն ունի β-նորմալ ձև, ապա այն խիստ β-նորմալիզացվող է:

Ներկայացնենք բնական թվերի կոդավորումը $TermI$ բազմության թերմերի միջոցով և մի քանի այլ օգտագործվող նշանակումներ:

1. $I \equiv \lambda x.x$,

2. $M^0 M' \equiv M'$, $M^{n+1} M' \equiv M M^n M'$, որտեղ $M, M' \in TermI$, $n \geq 0$,

3. $C_0 \equiv \lambda xy.xIIy$, $C_n \equiv \lambda fx.f^n x$, որտեղ $n > 0$:

Սահմանում 3.1.4 Դիցուք ունենք $\varphi: A \to N$ թվաբանական ֆունկցիան, $A \subset N^k$ և $k > 0$: Կասենք, որ $M \in TermI$ թերմը ներկայացնում է φ ֆունկցիան կամ φ-ն ներկայացվում է M թերմով, եթե՝

1. եթե $n_1, \dots, n_k \in A$ և $\varphi(n_1, \dots, n_k) = m$, ապա $M C_{n_1} \dots C_{n_k} =_\beta C_m$,

2. եթե $n_1, \dots, n_k \notin A$, ապա $M C_{n_1} \dots C_{n_k}$ թերմը չունի β-նորմալ ձև:

Թեորեմ 3.1.2 (Kleene) $\varphi: A \to N$ թվաբանական ֆունկցիան, որտեղ $A \subset N^k$ և $k > 0$, ներկայացվում է $TermI$ բազմության որևէ թերմի միջոցով այն և միայն այն դեպքում, երբ φ-ն մասնակի կարգընթաց ֆունկցիա է:

3.2 բաժնում ապացուցվում են ֆունկցիոնալ ծրագրերի տիպային կոռեկտության խնդրի համար ստացված անլուծելիության արդյունքները (թեորեմ 3.2.1, թեորեմ 3.2.2, թեորեմ 3.2.3 և թեորեմ 3.2.4):

Կատարենք հետևյալ նշանակումը՝ $R^0(M) \equiv M$, $R^{n+1}(M) \equiv R(R^n(M))$, որտեղ R-ը հանդիսանում է ինչ-որ $\beta\delta$-ռեդուկցիոն ստրատեգիա, $M \in Term$ և $n \geq 0$:

Սահմանում 3.2.1 Դիցուք $M \in Term$ և R-ը հանդիսանում է որևէ $\beta\delta$-ռեդուկցիոն ստրատեգիա: Կասենք, որ M թերմը պարունակում է տիպի սխալ R ստրատեգիայում, եթե գոյություն ունի $n \geq 0$ այնպիսին, որ $\perp$-ն հանդիսանում է $R^n(M)$ թերմի ենթաթերմ:

Սահմանում 3.2.2 Դիցուք $M \in Term$ և R-ը հանդիսանում է որևէ $\beta\delta$-ռեդուկցիոն ստրատեգիա: Կասենք, որ R ստրատեգիայի աշխատանքը M թերմի վրա վերջավոր է, եթե գոյություն ունի $n \geq 0$ այնպիսին, որ $R^{n+1}(M) \equiv R^n(M)$: Հակառակ դեպքում կասենք, որ R-ի աշխատանքը M թերմի վրա անվերջ է:

Թեորեմ 3.2.1 Դիցուք R-ը հանդիսանում է որևէ $\beta\delta$-ռեդուկցիոն ստրատեգիա: Այդ դեպքում գոյություն չունի այնպիսի ալգորիթմ, որը, մուտքում ստանալով M թերմը, վերադարձնում է այո, եթե M-ը չի պարունակում տիպի սխալ R ստրատեգիայում և վերադարձնում է ոչ, եթե M-ը պարունակում է տիպի սխալ R ստրատեգիայում:

Թեորեմ 3.2.2 Գոյություն չունի այնպիսի ալգորիթմ, որը, մուտքում ստանալով M թերմը, վերադարձնում է այո, եթե այն չի պարունակում տիպի սխալ որևէ $\beta\delta$-ռեդուկցիոն ստրատեգիայում և վերադարձնում է ոչ հակառակ դեպքում՝ եթե M-ը պարունակում է տիպի սխալ ցանկացած $\beta\delta$-ռեդուկցիոն ստրատեգիայում:

Թեորեմ 3.2.3 Գոյություն չունի այնպիսի ալգորիթմ, որը, մուտքում ստանալով M թերմը, վերադարձնում է այո, եթե M թերմը չի պարունակում տիպի սխալ ցանկացած $\beta\delta$-ռեդուկցիոն ստրատեգիայում և վերադարձնում է ոչ հակառակ դեպքում՝ եթե M-ը պարունակում է տիպի սխալ որևէ $\beta\delta$-ռեդուկցիոն ստրատեգիայում:

Թեորեմ 3.2.4 Դիցուք R-ը հանդիսանում է որևէ $\beta\delta$-ռեդուկցիոն ստրատեգիա: Այդ դեպքում գոյություն չունի այնպիսի ալգորիթմ, որը, մուտքում ստանալով M թերմը, վերադարձնում է այո, եթե M-ը չի պարունակում տիպի սխալ R ստրատեգիայում և R ստրատեգիայի աշխատանքը M-ի վրա վերջավոր է և վերադարձնում է ոչ հակառակ դեպքում, այսինքն՝ եթե M թերմը պարունակում է տիպի սխալ R ստրատեգիայում կամ R ստրատեգիայի աշխատանքը M թերմի վրա անվերջ է:

ԱՇԽԱՏԱՆՔԻ ՀԻՄՆԱԿԱՆ ԱՐԴՅՈՒՆՔՆԵՐԸ

Ատենախոսությունում ստացվել են հետևյալ հիմնական արդյունքները.

1. Կառուցվածքային λ-թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի համար կառուցվել է տիպերի արտածման համակարգ: Ապացուցվել է, որ միևնույն միջավայրում տիպայնացվող ֆունկցիոնալ ծրագրի և թերմի համար ցանկացած ինտերպրետացիայի ալգորիթմի աշխատանքի ընթացքում ստացվող թերմերը չեն պարունակի տիպի սխալ: Ստացվել են նաև կառուցվածքային λ-թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի տիպայնացման այնպիսի ալգորիթմներ, որոնք միշտ կանգ են առնում և հաջող ավարտի դեպքում տիպայնացնում են տրված թերմը կամ ֆունկցիոնալ ծրագիրը ամենաընդհանուր ձևով, իսկ անհաջող ավարտ տեղի է ունենում միայն այն դեպքում, երբ տրված թերմը կամ ֆունկցիոնալ ծրագիրը տիպայնացվող չէ վերը նշված տիպերի արտածման համակարգում, [1]:
2. Ընդլայնվել է առանց տիպերի λ-թերմերի տիպերի արտածման SYSTEM E համակարգը հաստատուններ պարունակող թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի համար: Ապացուցվել է, որ «միասին և նորմալ» տիպայնացվող ծրագրի և թերմի համար մեր կողմից դիտարկվող ինտերպրետացիայի ալգորիթմների աշխատանքի ընթացքում ստացվող թերմերը չեն պարունակի տիպի սխալ: Ընդլայնվել է նաև SYSTEM E համակարգում առանց տիպերի λ-թերմերի տիպայնացման ալգորիթմը հաստատուններ պարունակող թերմերի և դրանք օգտագործող ֆունկցիոնալ

ծրագրերի համար և ապացուցվել է, որ հաջող ավարտի դեպքում թերմերի տիպայնացման ալգորիթմը դուրս է բերում հաստատուններ պարունակող թերմի գլխավոր տիպայնացումը, իսկ անհաջող ավարտ տեղի է ունենում միայն այն դեպքում, երբ ընդլայնված SYSTEM E համակարգում տրված թերմը նորմալ տիպայնացվող չէ, [2], [3], [4]:

3. Ապացուցվել է, որ գոյություն չունի հաստատուններ պարունակող λ-թերմերի և դրանք օգտագործող ֆունկցիոնալ ծրագրերի տիպայնացման այնպիսի ալգորիթմ, որն ընդունում է բոլոր այն թերմերը և ֆունկցիոնալ ծրագրերը, որոնց ինտերպրետացիայի ժամանակ տիպի սխալներ տեղի չեն ունենում և մերժում է բոլոր այն թերմերը և ֆունկցիոնալ ծրագրերը, որոնց ինտերպրետացիայի ժամանակ տեղի են ունենում տիպի սխալներ, [5]:

ԱՏԵՆԱԽՈՍՈՒԹՅԱՆ ՇՐՋԱՆԱԿՆԵՐՈՒՄ ՀՐԱՏԱՐԱԿՎԱԾ ԱՇԽԱՏԱՆՔՆԵՐԻ ՑԱՆԿԸ

1. Arakelyan A.H. On Type Correctness of Polymorphic Functional Programs. *Proceedings of the Conference on Computer Science and Information Technologies (CSIT-2007), Publishing House of NAS of RA, Yerevan, 2007, p. 20-22.*
2. Arakelyan A.H. Type Inference System of Polymorphic λ-Terms. *Proceedings of the Conference on Computer Science and Information Technologies (CSIT-2009), Publishing House of NAS of RA, Yerevan, 2009, p. 15-18.*
3. Arakelyan A.H. On the Type Correctness of Polymorphic λ-Terms 1. *Proceedings of the Yerevan State University, Physical and Mathematical Sciences, Yerevan State University Press, Yerevan, 2009, №3, p. 42-51.*
4. Arakelyan A.H. On the Type Correctness of Polymorphic λ-Terms 2. *Proceedings of the Yerevan State University, Physical and Mathematical Sciences, Yerevan State University Press, Yerevan, 2010, №1, p. 37-46.*
5. Arakelyan A.H. Unsolvability of Type Correctness Problem for Functional Programs. *Proceedings of the Yerevan State University, Physical and Mathematical Sciences, Yerevan State University Press, Yerevan, 2011, №1, p. 28-35.*

РЕЗЮМЕ

Аракелян Ара Гайкович

“О типовой корректности функциональных программ”

Диссертационная работа посвящена проблемам функционального программирования. Объектами исследования являются бестиповые λ-термы и функциональные программы использующие такие термы, а основной проблемой является задача о типовой корректности этих термов и программ. В языках программирования типы используются в разных целях: для обнаружения ошибок несоответсвия типов в программе до начала её работы, для выполнения некоторых оптимизаций и т.д. Основная задача данной работы связана с первой из вышеупомянутых целей для функциональных программ: избегать ошибок несоответсвия типов во время выполнения программы путём проверки её типовой корректности до начала работы программы. В некоторых языках программирования не даётся информация о типах явным образом. Примерами таких функциональных языков являются Lisp, SML, OCaml, F# и т.д. В этих случаях появляется необходимость создания систем вывода типов и алгоритмов типизации, которые пытаются восстанавить недостающую информацию о типах программ. Для алгоритмов типизации важно, чтобы они гарантировали типовую корректность типизируемых ими программ. То есть, если некоторый алгоритм типизации типизирует программу, то во время функционирования интерпретатора, в случае данной программы, ошибка несоответсвия типов не должна возникнуть. В основном используются алгоритмы интерпретации основанные на подстановках и редукциях. Из вышесказанного следует, что алгоритмы типизации должны “отвергать” все такие программы, во время интерпретации которых происходит ошибка несоответсвия типов. С другой стороны, кроме программ содержащих ошибку типа, алгоритмы типизации могут “отвергать” ещё и такие программы, во время интерпретации которых ошибка несоответсвия типов не происходит. Это означает, что для алгоритмов типизации важно ещё и то, чтобы они “отвергали” как можно меньше таких программ, которые на самом деле не содержат ошибку типа. Естественно возникает следующий вопрос: возможно ли построение такого алгоритма типизации, который “отвергал” бы все программы, во время интерпретации которых происходит ошибка типа и “принимал” бы все программы, во время интерпретации которых ошибка типа не происходит? Резюмируя вышесказанное, следует отметить, что задачи связанные с типовой корректностью функциональных программ актуальны.

В диссертационной работе рассматриваются следующие основные задачи.

- Исследовать задачу типовой корректности структурированных λ-термов и функциональных программ использующих такие термы. Построить систему вывода типов и алгоритмы типизации этих термов и программ. Исследовать возможности построенных системы вывода типов и алгоритмов типизации.

- Исседовать возможности расширения системы вывода типов безтиповых λ-термов - SYSTEM E для термов содержащих константы и функциональных программ использующих такие термы. Построить алгоритмы типизации таких термов и программ. Исследовать возможности расширенной системы и построенных алгоритмов типизации.

- Исследовать возможности всех возможных алгоритмов типизации λ-термов содержащих константы и функциональных программ использующих такие термы.

В диссертационноий работе получены следующие основные результаты:

1. Построена система вывода типов для структурированных λ-термов и функциональных программ использующих такие термы. Доказано, что во время функционирования любого интерпрататора в случае типизируемых (в одной и той же среде) терма и программы, полученные термы не будут содержать ошибку типа. Получены такие алгоритмы типизации структурированных термов и функциональных программ использующих эти термы, которые всегда останавливаются и при удачном завершении типизируют эти термы и программы самым общим образом, а неудачное завершение происходит только тогда, когда входной терм или программа не типизируемы в системе вывода типов, [1].
2. Расширена система вывода типов бестиповых λ-термов - SYSTEM E для термов содержащих константы и функциональных программ использующих такие термы. Доказано, что во время функционирования рассматриваемых нами интерпретаторов в случае типизируемых терма и программы, полученные термы не будут содержать ошибку типа. Расширен алгоритм типизации термов системы SYSTEM E для термов содержащих константы и для функциональных программ использующих такие термы и доказано, что расширенный алгоритм типизации термов при удачном завершении выводит главную типизацию этих термов, а неудачное завершение происходит только тогда, когда входной терм не типизируем в расширенной системе, [2], [3], [4].
3. Доказано, что не существует такого алгоритма типизации λ-термов содержащих константы и функциональных программ использующих эти термы, который бы “принимал” все термы и программы, во время интерпретации которых ошибка типа не происходит и “отвергал” бы все термы и программы, во время интерпретации которых происходит ошибка типа, [5].

ABSTRACT

Ara H. Arakelyan

"On Type Correctness of Functional Programs"

The thesis is dedicated to the problems of functional programming. The main objects of the research are the untyped λ-terms and functional programs that use such terms and the main problem of the research is the problem of type correctness of that terms and functional programs. Types are used in programming languages for different purposes, such as detecting type errors earlier (before execution of the program), doing some optimizations etc. The main problem of the thesis is concerning to the first one of the objectives mentioned above: avoid type errors occurring during the program execution by checking its type correctness before the program execution. In some programming languages there is no explicit type information provided by the programmer. Such functional programming languages are Lisp, SML, OCaml, F# etc. In that case it is required to create type inference systems and typification algorithms, which are used for the typification of the programs and try to recover type information missed by the programmer. For the usage of the typification algorithms it is important to guarantee type correctness of the programs typified by them, i.e. if that typification algorithm typifies given program, than during the work of interpretation algorithm (generally interpretation algorithms based on the substitutions and reductions are considered) for that program no type error must occur. Hereby typification algorithms must reject all the programs, during interpretation of which type error will occur. On the other hand, except the programs containing type errors, typificiation algorithms can reject also some other programs, during interpretation of which type error will not occur. Hence, for the typification algorithms it is also important to reject as few programs as possible, which don't contain type errors indeed. The following question is appeared: is it possible to construct such algorithm of typification, which rejects all the programs during the interpretation of which type error will occur and accepts all the programs during the interpretation of which type error will not occur? By summarizing we can conclude, that the problems concerning to the type correctness of the functional programs are actual.

The following main problems are considered in the thesis.

- Investigate type correctness problem for the structured λ-terms and functional programs that use such terms. Construct type inference system and typification algorithms for that terms and programs. Investigate abilities of constructed type inference system and typification algorithms.

- Investigate abilities of extending type inference system of untyped λ-terms - SYSTEM E for the terms containing constants and for the functional programs that use such terms. Construct algorithms of typification for that terms and programs. Investigate abilities of extended system and constructed typification algorithms.

- Investigate abilities of possible typification algorithms for λ-terms containing constants and functional programs that use that terms.

The main results of the thesis are the following:

1. Type inference system has been constructed for the structured terms and functional programs that use such terms. It has been proven, that during the work of any interpretation algorithm for the term and program, that are typifiable in the same environment, no type error will occur. Algorithms of typification has been constructed for the structured terms and functional programs that use such terms, which always stop and in case of success typify given term or program in the most common way and fail only in cases, when given term or program is not typifable in the type inference system, [1].
2. Type inference system of the untyped λ-terms - SYSTEM E has been extended for the terms containing constants and for the functional programs that use such terms. It has been proven, that during the work of any interpretation algorithm considered by us for the typifiable term and program no type error will occur. Typification algorithm of SYSTEM E has been extended for the terms containing constants and for the functional programs that use such terms. It has been proven, that in case of success extended typification algorithm infers principal typing of the given term and fails only in cases, when given term is not typifiable in the extended system, [2], [3], [4].
3. It has been proven, that there is no algorithm of typification of the λ-terms containing constants and functional programs that use that terms, which accepts all the terms and programs during the interpretation of which type error will not occur and rejects all the terms and programs during the interpretation of which type error will occur, [5].