ГОДИШНИК НА СОФИЙСКИЯ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Книга 3
Том 88, 1994

ANNUAIRE DE L'UNIVERSITE DE SOFIA „ST. KLIMENT OHRIDSKI“
FACULTE DE MATHEMATIQUES ET INFORMATIQUE
Livre 3
Tome 88, 1994

СЕЧЕНИЕ НА ТИПОВЕ И ПРЕТОВАРВАНЕ

ДИМИТЪР БИРОВ

Димитр Биров. СЕЧЕНИЕ ТИПОВ И ПЕРЕГРУЗКА

Типовые системы, использованые в языках программирования, являются практическим результатом формальной теории типов. Типовые системы, которые базируются на сечении типов, изучаются интенсивно в последние годы как средства анализа чистого λ-исчисления, а также и как основа практических языков программирования. Одна из привлекательных черт сечении типов является его способность описывать конечные куски информации о программе.

Термин „перегрузка“ звучит актуальнее в языках программирования. Арифметическая операция сложения (+) для целых и вещественных чисел является типичным примером перегрузки. В этой статье мы обобщаем это понятие и опишем типовую систему, которая включает перегрузку функции высшего порядка..

Dimitar Birov. INTERSECTION TYPES AND OVERLOADING

Type systems which are employed in programming languages are a practical result obtained by the formal type theory. Type systems based on intersection types are studied extensively during the past years both as tools for analysis of pure λ-calculus and as a foundation for practical programming languages. One of the intriguing properties of intersection types is their ability to express an unbounded finite amount of information about a program.

The notion of overloading sounds very actually in programming languages. We point an addition function (+) for both adding integer and real numbers as a typical example for it. We generalize overloading and describe type system including higher order function overloading in this paper.

премице

Типовите системи, използвани в езиците за програмиране, са практически резултат от формална теория на типовете. Основното им предназначение е диагностика на класове от грешки, които програмистът може да допусне при използване на разнородни величини в един израз.

Основна теорема в теорията на типовете е теоремата за пълнота, която твърди, че ако един израз е синтактично добре типизиран, то той и семантично е добре типизиран. Това гарантира сигурност на програмите и следователно надеждност на типовата система. При коректно използване на типовата система не е възможно да възникнат типови грешки в процеса на изпълнение на програмата.

Типовите системи, базирани на *сечение на типове*, се изучават интензивно през последните години както като средства за анализ на чистото λ-смятане, така и като основа на практически езици за програмиране. Сечението на типове е разработено в края на 70-те години от Coppo и Dezani-Ciancaglini [10] и независимо от тях от Salle [12, 38] и Pottinger [34]. Оттогава те се изучават интензивно от членовете на групата в Тюрин [3, 35, 10, 11, 14, 15, 20, 36, 37] и др. Изследвани са различни разширения на оригиналната типова дисциплина със сечения на типове, включващи понятие за безкрайни сечения [26], дуалното му понятие — *обединение* на типове [4, 19, 32], и връзките между сечение на типове и модели на полиморфизъм [30]. Разработват се разширения на системите за извод на тип в **ML** стил, свързани с понятията за уточняване на типове [17, 18, 32] и гъвкаво типизиране [7, 16]. В съвременните студии на Coppo и Giannini се изучава проблемът, свързан с извод на тип в типова система със сечения на типове с наложени разрешими ограничения [13]. Reynolds първи демонстрира използване на сечение на типове като основа на практически езици за програмиране [35].

Pierce [31, 33] разглежда сечението на типове и свързаното квантуване — комбинация от параметричен полиморфизъм и подтипизиране, като комплементарни механизми, които увеличават изразителната сила на статично типизираните езици за програмиране, и изследва λ-смятане, което ги комбинира. Чрез редица примери демонстрира практическата приложимост на това смятане, наречено $F_\wedge$ (F миит). Съществено място сред тях заема *крайният полиморфизъм* или *кохерентно претоварване*.

Терминът *претоварване* звучи все по-актуално в езиците за програмиране. Най-често той характеризира свойство на операциите — процедури, функции, методи — и означава, че интерпретацията на едно и също име на операция е различна и зависи от контекста, в който това име се среща. Типичен пример за претоварване е аритметичната операция + за събиране. С името + се означава операция за събиране както на цели, така и на реални числа. Типът на тази операция е различен в двата случая. Събиране на цели числа има тип Int → Int → Int, а за реални числа

— Real → Real → Real. Съществената разлика е в последователността от машинните инструкции, които процесорът изпълнява. Претоварване на имената се среща както в съвременните конвенционални езици за програмиране като ADA [41], така и в нетрадиционните апликативни езици — Haskell [21] и Gofer [24].

Претоварването на операциите поражда проблеми както от теоретично естество — свързани със семантиката на типовете, така и от практическа гледна точка — проверка и извод на тип. Изследванията, направени върху семантична интерпретация на претоварването от Kaes [25] за функции от първи ред и Биров [42, 43] за рекурсивни функции и функции от по-висок ред в семантичната област на параметричния полиморфизъм, показват единната основа на параметричния — безкраен — и претоварването като краен полиморфизъм. Строгата теоретична обосновка на Wadler и Blott [40] прави възможна реализацията на претоварването в Haskell [29] чрез използване на класове от типове. Разрешаването на претоварването (или изборът на конкретна операция) се извършва по време на компилация от предпроцесор чрез използване на информация, специфицирана от потребителя. Доколкото ни е известно, типът на аргументите определя избор на код по време на компилация на програмата при използването на явния вид претоварване. Развитие на този подход и естествено обобщение на типовите класове, използвани в Haskell, прави Jones [22–24], като описва гъвкава типова система, съдържаща конструктори на класове и полиморфизъм от по-висок ред в неявно типизираните езици за програмиране.

Търсещи нови теоретични модели на λ-смятане, които изразяват претоварването, са студиите на Castagna, Ghelli и Longo [8, 9]. Те отразяват необходимостта от семантична обосновка на претоварването в широко разпространената напоследък обектно-ориентирана парадигма.

1.1. СЕЧЕНИЕ НА ТИПОВЕ И ПРЕТОВАРВАНЕ НА ФУНКЦИИ

Една от атрактивните черти на сечението на типове е способността му да описва несвързани, крайни късове информация за програмата. Например на функцията за събиране + може да бъде присвоен тип

$$\mathrm{Int} \to \mathrm{Int} \to \mathrm{Int} \ \wedge \ \mathrm{Real} \to \mathrm{Real} \to \mathrm{Real},$$

който описва основните факти, че сумата на две реални числа е винаги реално число и сумата на две цели числа е цяло число. Компилаторът за език със сечение на типове може дори да осигури две различни последователности от обектен код за различните версии на +: една инструкция за събиране на числа с плаваща точка и друга за събиране на цели числа. За всеки екземпляр на операцията + в програмата компилаторът може да определи дали двата аргумента са цели числа и по този начин да генерира по-ефективна кодова последователност.

Сечението на типове специфицира за всяка двойка типове σ и τ най-голяма долна граница $\sigma \wedge \tau$, съответстваща интуитивно на сечението на множествата от стойности, описани чрез σ и τ. В представения пример $\mathrm{Int} \to \mathrm{Int} \to \mathrm{Int} \ \wedge \ \mathrm{Real} \to \mathrm{Real} \to \mathrm{Real}$ описва най-голямата долна граница на типовете $\mathrm{Int} \to \mathrm{Int} \to \mathrm{Int}$ и $\mathrm{Real} \to \mathrm{Real} \to \mathrm{Real}$.

В тази статия ще обобщим претоварването, като ще опишем типова система, която включва и претоварване на функции от по-висок ред.

1.2. ПРЕТОВАРВАНЕ НА ФУНКЦИИ ОТ ПО-ВИСОК РЕД

Пример за претоварена функция от по-висок ред е функцията **map**. Тя оперира върху линейни последователности от елементи, наречени линейни списъци. Резултатът от нейното действие е линеен списък, който се получава от изходния, като върху всеки от неговите елементи се приложи една и съща едноаргументна функция.

Нека дефинираме типа данни списък List по следния начин:

$$\textbf{data}\ \mathrm{List}\,\alpha = \mathrm{Nil} \mid \mathrm{Cons}(\alpha, \mathrm{List}\,\alpha),$$

където Nil означава списък без елементи, а Cons е типов конструктор, който от елемент от произволен тип α и списък $\mathrm{List}\ \alpha$ конструира линеен списък.

Функцията **map** ще дефинираме рекурсивно чрез две равенства:

$$\begin{aligned} \textbf{map}\ \mathrm{f}\ \mathrm{Nil} &= \mathrm{Nil}, \\ \textbf{map}\ \mathrm{f}\ \mathrm{Cons}(x, xs) &= \mathrm{Cons}((\mathrm{f}\ x),\ (\textbf{map}\ \mathrm{f}\ xs)) \end{aligned}$$

Теорията на полиморфизма позволява за тази функция да бъде изведен следният полиморфен тип [44]:

$$\forall\alpha.\forall\beta.(\alpha \to \beta) \to \mathrm{List}\,\alpha \to \mathrm{List}\,\beta.$$

Нека сега дефинираме типа данни двоично дърво по следния начин:

$$\textbf{data}\ \mathrm{Tree}\,\alpha = \mathrm{Leaf}\,\alpha \mid \mathrm{TCons}(\mathrm{Tree}\,\alpha, \mathrm{Tree}\,\alpha),$$

където $\mathrm{Leaf}\,\alpha$ представлява листо в дървото, а TCons е типов конструктор, който от две поддървета конструира двоично дърво.

Така дефинираното двоично дърво има съвършено различна вътрешна структура в сравнение с линейния списък.

За двоично дърво бихме могли да дефинираме функция от по-висок ред, която прилага дадена едноаргументна функция към всеки елемент от дървото по следния начин:

$$\begin{aligned} \textbf{map}\ \mathrm{f}\ (\mathrm{Leaf}\ x) &= \mathrm{Leaf}\ (\mathrm{f}\ x), \\ \textbf{map}\ \mathrm{f}\ \mathrm{TCons}(\mathrm{l}, \mathrm{r}) &= \mathrm{TCons}((\textbf{map}\ \mathrm{f}\ \mathrm{l}), (\textbf{map}\ \mathrm{f}\ \mathrm{r})). \end{aligned}$$

Очевидна е разликата в поведението на функцията **map** в двата варианта, въпреки че тя има едно и също име. Това поведение зависи от вътрешната структура на типа данни. Полиморфният тип на функцията **map** във втория случай е

$$\forall\alpha.\forall\beta.(\alpha \to \beta) \to \mathrm{Tree}\,\alpha \to \mathrm{Tree}\,\beta.$$

Като използваме сечението на типове, на функцията **map** ще бъде присвоен тип:

$$\forall\alpha.\forall\beta.(\alpha \to \beta) \to \mathrm{List}\,\alpha \to \mathrm{List}\,\beta \ \wedge\ \forall\alpha.\forall\beta.(\alpha \to \beta) \to \mathrm{Tree}\,\alpha \to \mathrm{Tree}\,\beta.$$

Pierce въвежда сечението на типове на ниво термове чрез израза

$$\textbf{for}\ \alpha\ \textbf{in}\ \tau_1..\tau_n.e,$$

където α е типова променлива, а $\tau_1..\tau_n$ са типови изрази. Този израз се интерпретира като инструкция към процедурата за проверка на тип, като посочва да се анализира изразът e поотделно за всяка стойност на типовата променлива α в множеството от типови изрази $\tau_1..\tau_n$ ($\alpha \equiv \tau_1, \ldots, \alpha \equiv \tau_n$) и след това да се образува сечението на получените в резултат типове.

Въвеждането на конструкцията **for** разделя механизма на функционална абстракция от алтернативен избор на тип и разширява изразителната сила на езика, като осигурява *име* за всеки избор, направен от процедурата за проверка на тип. Това дава възможност да се изрази обобщената λ-абстракция, представена от Reynolds [35], която позволява явно контролиране от програмиста алтернативно типизиране на термовете, чрез просто синтактично съкращение

$$\lambda x : \sigma_1..\sigma_n.e =_{\mathrm{def}} \ \textbf{for}\ \alpha\ \textbf{in}\ \sigma_1..\sigma_n.\lambda x : \alpha.e,$$

където α е нова типова променлива.

Тъй като изразът e за тялото на функцията **map** е съвършено различен, както се вижда от дефинициите, не е възможно тази функция да бъде описана чрез използване на конструкцията за алтернативен избор на тип **for**.

Ето защо предлагаме да се разшири граматиката, дефинираща типовите изрази с подобна конструкция, която ще наречем също **for**:

$$\textbf{for}\ f\ \textbf{in}\ \mathrm{List},\ \mathrm{Tree}.\forall\alpha.\forall\beta.(\alpha \to \beta) \to f\alpha \to f\beta.$$

Този типов израз се чете по следния начин: „За всяка типова променлива f, която има за стойности List или Tree, и за всяка типова променлива α и всяка типова променлива β и функция, която изобразява стойностите от α в стойностите от β, получаваме функция, която изобразява стойностите от типа $f\,\alpha$ в стойностите на типа $f\,\beta$.“

Тази конструкция е съкратен запис на сечението на еднакви типови изрази за функцията **map**, дадени по-горе. Представеният типов израз е полиморфен по отношение на всяка конкретна стойност на променливата f и претоварен по отношение на абстракцията, която тя внася. Той може да се разглежда като съкратен аналог на типовия израз

$$\textbf{for}\ \xi\ \textbf{in}\ \mathrm{List},\ \mathrm{Tree}.f : \xi.\forall\alpha.\forall\beta.(\alpha \to \beta) \to f\alpha \to f\beta,$$

където променливата ξ е променлива по отношение на сорта на типовата променлива f. Понеже в граматика, дадена в раздел 2.1, която описва сортовете в използвания модел, не съществуват променливи, а и, от друга страна, типовите изрази List и Tree са от един и същ сорт, явното

описание на сортова променлива за специфициране на сорта на типовата променлива f не е необходимо. Нещо повече, могат да се използват стандартни техники за извеждане на сорта на дадена типова променлива от контекста, в който тази променлива участва в дадения типов израз.

В такъв случай не се налага потребителят да специфицира информация за сорта на типовите изрази, които описва. Той може да бъде изведен автоматично.

Предложената конструкция

$$\textbf{for } \alpha \textbf{ in } \tau_1..\tau_n.\tau$$

е съкратен аналог на обобщена типова λ-абстракция

$$\lambda\alpha : \sigma_1..\sigma_n.\tau = \textbf{for } \xi \textbf{ in } \sigma_1..\sigma_n.\lambda\alpha : \xi.\tau$$

и разделя механизма на функционална, типова абстракция от алтернативен избор на тип, като задава име на всеки избор, направен от процедурата за проверка на тип по начин, аналогичен на конструкцията **for** за термове.

Тя по естествен начин отразява семантичната основа на крайния полиморфизъм (претоварване) и параметричния универсален полиморфизъм.

Очевидно, за да бъде добре формиран типът на претоварената функция **map**, свързаната променлива f трябва да приема стойности, които изобразяват един тип в друг — функции с един аргумент от типове в типове. Понеже f може да приеме стойност List, като я приложим към конкретен тип Int, се получава нов тип List Int, определящ линеен списък от цели числа. Това е прилагане на един типов израз към друг. След като в типовите изрази има прилагане, то ще следва да има и абстракция на типови изрази или възможност за дефиниране на функции от типове в типове. Следователно езикът, който описва типовите изрази, представлява сам по себе си λ-смятане.

Както и в традиционния случай ще заменим изразителността на безтиповото λ-смятане с надеждността, която осигурява типизираното λ-смятане. В конкретния случай вида на типовите изрази ще наричаме *сортове*, като следваме терминологията, дефинирана от Barendregt [2] за обобщени типови системи.

2. СИНТАКСИС

В този раздел ще представим формално типизирано λ-смятане, с което ще работим. Ще специфицираме езиците, задаващи термовете, типовете и сортовете.

Сортовете на $F^\omega_\wedge$ са същите както и на F^ω [5]. Константата $\star$ означава сорта на всички типове. Тогава $\tau : \star$ е твърдение, което означава, че „τ е тип“. Изразът $\star \to \star$ не е означение на тип. Сортовете се дефинират от следната абстрактна граматика:

$$\begin{array}{rll} K ::= & \star & \text{типове} \\ & K \to K & \text{типови оператори.} \end{array}$$

Сортове от вида $K_1 \to K_2$ са типови функции — функции, изобразяващи типове в типове, наричани *типови оператори* или *типови конструктори*. Типовите константи — такива като Int и Real, могат да се интерпретират като типови оператори с нулева размерност. Техният сорт е $\star$. Типовите оператори List и Tree са от сорт $\star \to \star$, а операторните константи за конструиране на функционален тип ($\to$) и сечение на типове ($\wedge$) имат сорт $\star \to (\star \to \star)$.

Езикът на *типовете* в $F^\omega_\wedge$ е разширение от по-висок ред на F^ω. Както и F^ω то включва типови променливи, означени с X, функционални $(T \to T)$ и полиморфни типове $\mathbf{all}(X : K)T$. Нещо повече, подобно на F^ω ние ще позволим типова абстракция чрез типове $(\mathbf{fun}(X : K)T)$ и прилагане към аргумент тип (TT'). Като ефект тези форми въвеждат просто типизирано λ-смятане на ниво типове. Накрая ще позволим произволни крайни сечения $(\wedge^K[T_1..T_n])$, където всичките T_i са членове на един и същ сорт K.

$$\begin{array}{rll} T ::= & X & \text{— типови променливи,} \\ & T \to T & \text{— функционален тип,} \\ & \mathbf{all}\ (X : K)T & \text{— полиморфен тип,} \\ & \mathbf{for}\ (X \in T..T)T & \text{— алтернативен избор на тип,} \\ & \mathbf{fun}\ (X : K)T & \text{— операторна абстракция,} \\ & TT & \text{— операторно прилагане,} \\ & \wedge^K[T..T] & \text{— сечение от сорт } K. \end{array}$$

Ще използваме съкращенията $\top^K$ и $S \wedge^K T$ за нулево и бинарно сечение:

$$\top^K =_{\text{def}} \wedge^K[\,], \qquad S \wedge^K T =_{\text{def}} \wedge^K[S, T].$$

По технически съображения ще осигурим сортова анотация за свързаните променливи и сечения, така че всеки тип да има „очевиден сорт“, който може да се прочете направо от неговата структура и декларацията на сорта в контекста.

Езикът от термове включва променливи (x), функционална абстракция $(\mathbf{fun}(x : T)e)$ и прилагане $(e\,e)$ от просто типизирано λ-смятане плюс

типова абстракция $(\mathbf{fun}\,(X : K)e)$ и прилагане $(e\,T)$ от F^ω.

$$\begin{array}{rll} e ::= & x & \text{— променлива,} \\ & \mathbf{fun}\ (x : T)e & \text{— абстракция,} \\ & e\ e & \text{— апликация,} \\ & \mathbf{fun}\ (X : K)e & \text{— типова абстракция,} \\ & e\ T & \text{— типова апликация,} \\ & \mathbf{for}\ (X \in T..T).e & \text{— алтернация.} \end{array}$$

Сечението на типове се въвежда чрез израз от вида $\mathbf{for}\ (X \in T_1..T_n)e$, който може да бъде прочетен като инструкция към процедурата за проверка на тип да анализира израза e поотделно за всяко допускане $X \equiv T_1$, $X \equiv T_2$, ..., и т. н. и да пресече резултатите. Например, ако

$$+ \in \mathbf{for}\ (X \in \mathrm{Int},\ \mathrm{Real}).X \to X \to X,$$

то можем да изведем

$$\mathbf{for}\ (X \in \mathrm{Int},\ \mathrm{Real}).\mathbf{fun}\ (x : X)x + x\ \in\ \mathbf{for}\ (X \in \mathrm{Int},\ \mathrm{Real}).X \to X.$$

Контекст Γ е крайна последователност от типови предположения за множеството от термови и/или типови променливи. Празния контекст ще означаваме $\varnothing$.

$$\begin{array}{ll} \varnothing & \text{— празен контекст,} \\ \Gamma, x : T & \text{— декларация на термова променлива,} \\ \Gamma, X : K & \text{— декларация на типова променлива.} \end{array}$$

Областта на Γ, или всички променливи, формиращи контекста, ще означаваме с $\mathrm{dom}(\Gamma)$. Функциите $\mathrm{FV}(-)$ и $\mathrm{FTV}(-)$ дават множествата от свободни променливи и свободни типови променливи в терм, тип или контекст. Тъй като всяка променлива, която е свързана, се среща точно веднъж в контекст Γ, понякога ще считаме контекстите като крайни функции: $\Gamma(X)$ дава връзката на X в Γ, където за X неявно се предполага, че е в $\mathrm{dom}(\Gamma)$.

Типовете, термовете, контекстите, твърденията и изводите, които се различават само по имената на свързаните променливи, се считат за идентични.

Субституцията, която за всяко срещане на типовата променлива X поставя типовия израз S в типовия израз T, ще означаваме $T[X \leftarrow S]$. По същия начин ще означаваме субституциите за термове и за контекст.

Обичайната едностъпкова β-редукция на типа S в типа T се означава с $\to_\beta$. Конверсията ще означаваме с $=_\beta$.

2.2. ПРАВИЛА ЗА ТИПИЗИРАНЕ

Ще представим правила за присвояване на сорт и типизиране в $F^\omega_\wedge$. Те са организирани като системи, използвани за доказване за трите независими съждителни форми:

$$\begin{array}{ll} \Gamma \vdash \text{дфк} & \text{— добре формиран контекст,} \\ \Gamma \vdash T \in K & \text{— тип от подходящ сорт,} \\ \Gamma \vdash e \in T & \text{— добре типизиран терм.} \end{array}$$

Понякога ще използваме метапроменливата Σ, чрез която ще означаваме твърденията (дясната страна на съжденията) на някоя от тези три форми.

Повечето от правилата, дадени по-долу, включват предпоставки от два различни вида: *структурни предпоставки, които играят съществена* роля в определянето на семантичната сила на дадено правило, и *предпоставки за добра формираност*, които гарантират, че обектите, участващи в правилото, са от очакваните видове.

Контекст. Правилата за добре формиран контекст се състоят от началното правило за празен контекст и правилата, позволяващи даден добре формиран контекст да бъде разширен с термова или типова променлива.

$$\varnothing \vdash \text{дфк} \qquad \text{(C-Empty)}$$

$$\frac{\Gamma \vdash T \in \star \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : T \vdash \text{дфк}} \qquad \text{(C-Var)}$$

$$\frac{\Gamma \vdash T \in K \quad X \notin \mathrm{dom}(\Gamma)}{\Gamma, X : K \vdash \text{дфк}} \qquad \text{(C-TVar)}$$

Формиране на тип. За всеки типов конструктор ще опишем правило, което специфицира как се построяват добре формирани типови изрази. Критични правила са K-TAbs и K-TApp, които формират типовата абстракция и типовата апликация.

Предпоставката $\Gamma \vdash$ дфк за добре формиран контекст в правилата K-Meet-I и T-Meet-I по-долу е необходима, когато $n = 0$.

$$\frac{\Gamma_1, X : K, \Gamma_2 \vdash \text{дфк}}{\Gamma_1, X : K, \Gamma_2 \vdash X \in K} \qquad \text{(K-TVar)}$$

$$\frac{\Gamma \vdash T_1 \in \star \quad \Gamma \vdash T_2 \in \star}{\Gamma \vdash T_1 \to T_2 \in \star} \qquad \text{(K-Arrow)}$$

$$\frac{\Gamma, X : K \vdash T \in \star}{\Gamma \vdash \mathbf{all}(X : K)T \in \star} \qquad \text{(K-All)}$$

$$\frac{\Gamma, X : K_1 \vdash T \in K_2}{\Gamma \vdash \mathbf{fun}(X : K_1)T \in K_1 \to K_2} \qquad \text{(K-TAbs)}$$

$$\frac{\Gamma \vdash S \in K_1 \to K_2 \quad \Gamma \vdash T \in K_1}{\Gamma \vdash ST \in K_2} \qquad \text{(K-TApp)}$$

$$\frac{\Gamma \vdash T[X \leftarrow S] \in K \quad S \in \{S_1..S_n\}}{\Gamma \vdash \mathbf{for}(X \in S_1..S_n)T \in K} \qquad \text{(K-For)}$$

$$\frac{\Gamma \vdash \text{дфк} \quad \text{за } \forall i, \Gamma \vdash T_i \in K}{\Gamma \vdash \wedge^K[T_1..T_n] \in K} \qquad \text{(K-Meet-I)}$$

$$\text{за } \forall i \quad \frac{\Gamma \vdash \wedge^K[T_1..T_n] \in K}{\Gamma \vdash T_i \in K} \qquad \text{(K-Meet-E)}$$

Формиране на терм. Правилата за формиране на термове са същите като тези за смятане от втори ред с изключение на T-Meet-I и T-For. T-For осигурява алтернативни предположения за проверката на типа за всеки тип от множеството. За всяко S_i типът, изведен за екземпляр на тялото e, когато X се замени с S_i, е правилният тип на **for**-израза като цяло. Правилото T-Meet-I може да бъде използвано да събере тези отделни типизирания в единично сечение. От гледна точка на типовата теория T-Meet-I е правило за въвеждане на конструктора $\wedge$. Съответното правило за елиминиране е T-Meet-E.

$$\frac{\Gamma_1, x : T, \Gamma_2 \vdash \text{дфк}}{\Gamma_1, x : T, \Gamma_2 \vdash x \in T} \qquad \text{(T-Var)}$$

$$\frac{\Gamma, x : T_1 \vdash e \in T_2}{\Gamma \vdash \mathbf{fun}\,(x : T_1)e \in T_1 \to T_2} \qquad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash f \in T_1 \to T_2 \quad \Gamma \vdash a \in T_1}{\Gamma \vdash f\,a \in T_2} \qquad \text{(T-App)}$$

$$\frac{\Gamma, X : K \vdash e \in T}{\Gamma \vdash \mathbf{fun}\,(X : K)e \ \in \ \mathbf{all}\,(X : K)T} \qquad \text{(T-TAbs)}$$

$$\frac{\Gamma \vdash f \in \mathbf{all}\,(X : K)T \quad \Gamma \vdash S \in K}{\Gamma \vdash f\,S \in T[X \leftarrow S]} \qquad \text{(T-TApp)}$$

$$\frac{\Gamma \vdash e[X \leftarrow S] \in T \quad S \in \{S_1..S_n\}}{\Gamma \vdash \mathbf{for}\,(X \in S_1..S_n)e \ \in \ T} \qquad \text{(T-For)}$$

$$\frac{\Gamma \vdash \text{дфк} \quad \text{за } \forall i, \Gamma \vdash e \in T_i}{\Gamma \vdash e \in \wedge^\star[T_1..T_n]} \qquad \text{(T-Meet-I)}$$

$$\text{за } \forall i \quad \frac{\Gamma \vdash e \in \wedge^\star[T_1...T_n]}{\Gamma \vdash e \in T_i} \qquad \text{(T-Meet-E)}$$

2.3. ОСНОВНИ СВОЙСТВА

В този раздел са представени част от структурните свойства на дефиницията на $F^\omega_\wedge$. Доказателствата се извършват чрез структурна индукция.

Лема 2.3.1. *Ако* $\Gamma \vdash \Sigma$ *и* Γ_1 *е префикс на* Γ, *то* $\Gamma_1 \vdash$ дфк *е подизвод. Нещо повече, с изключение на случая, когато* $\Gamma_1 \equiv \Gamma$ *и* $\Sigma \equiv$ дфк, *този подизвод е по-кратък.*

Лема 2.3.2 (лема за свободните променливи). 1. *Ако* $\Gamma \vdash T \in K$, *то* $\mathrm{FTV}(T) \subseteq \mathrm{dom}(\Gamma)$.

2. *Ако* $\Gamma \vdash$ дфк, *то всяка термова променлива или типова променлива в* $\mathrm{dom}(\Gamma)$ *е декларирана точно веднъж.*

Лема 2.3.3. *Ако* $\Gamma_1, x : T, \Gamma_2 \vdash$ дфк, *то може да се изведе, че* $\Gamma_1 \vdash T \in \star$.

Лема 2.3.4. *Нека* Γ *и* Γ' *са контексти, такива че* $\Gamma \subseteq \Gamma'$ *и* $\Gamma' \vdash$ дфк. *От* $\Gamma \vdash \Sigma$ *следва* $\Gamma' \vdash \Sigma$.

Лема 2.3.5 (усилване на контекст, сорт и терм). 1. *Ако* $\Gamma_1, X : K, \Gamma_2 \vdash$ дфк *и* $X \notin \mathrm{FTV}(\Gamma_2)$, *то* $\Gamma_1, \Gamma_2 \vdash$ дфк.

2. *Ако* $\Gamma_1, X : K, \Gamma_2 \vdash S \in K'$ *и* $X \notin \mathrm{FTV}(\Gamma_2) \cup \mathrm{FTV}(S)$, *то* $\Gamma_1, \Gamma_2 \vdash S \in K'$.

3. *Ако* $\Gamma_1, x : T, \Gamma_2 \vdash \Sigma$ *и* $x \notin FV(\Sigma)$, *то* $\Gamma_1, \Gamma_2 \vdash \Sigma$.

Нещо повече, изводите в заключенията са по-кратки от изводите в предпоставките.

Лема 2.3.6 (уникалност на сорта). *Ако* $\Gamma \vdash T \in K$ *и* $\Gamma \vdash T \in K'$, *то* $K \equiv K'$.

Лема 2.3.7 (типова субституция). *Нека* $\Gamma_1 \vdash T \in K_U$. *Тогава*:
1. *Ако* $\Gamma_1, X : K_U, \Gamma_2 \vdash S \in K_S$, *то* $\Gamma_1, \Gamma_2[X \leftarrow T] \vdash S[X \leftarrow T] \in K_S$.
2. *Ако* $\Gamma_1, X : K_U, \Gamma_2 \vdash$ дфк, *то* $\Gamma_1, \Gamma_2[X \leftarrow T] \vdash$ дфк.

Лема 2.3.8 (инвариантност на сорта при типова конверсия). *Ако* $\Gamma \vdash S \in K_S$, *и* $\Gamma \vdash T \in K_T$ *и* $S =_\beta T$, *то* $K_S \equiv K_T$.

Лема 2.3.9 (добро сортиране на типизирането). *Ако* $\Gamma \vdash e \in T$, *то* $\Gamma \vdash T \in \star$.

3. СЕМАНТИКА

Едни от най-простите модели за типизирано λ-смятане са базираните на релации на частична еквивалентност (РЧЕ). В този модел термовете се интерпретират чрез изтриване на цялата типова информация и полученият чист λ-терм се интерпретира като елемент на модела. Типът в този случай е само подмножество на модела заедно с подходящо понятие за еквивалентност на елементите.

Нашият модел от РЧЕ за $F^\omega_\wedge$ разширява модела на $F_\wedge$, даден в [31], който се базира на модела на Bruce и Longo [5]. Обичайната интерпретация на квантувания тип $\mathbf{all}(X : K)T$ в модела на РЧЕ е индексирано по отношение на РЧЕ сечение на всички възможни екземпляри на T. $\wedge^*[T_1..T_n]$ се интерпретира като сечение на релациите на частична еквивалентност, интерпретиращи всяко T_i. Ще използваме обобщение на този модел чрез интерпретиране на типовите оператори като функции върху РЧЕ.

Нотацията на фундаменталните дефиниции, използвани тук, се базира на статията на Bruce и Longo [5]. Полезен основен справочник за модели на РЧЕ на λ-смятане от втори ред е [28]; вж. също [6] за по-обща дискусия на модели за втори ред и [1, 27] за обобщена дискусия на комбинаторни модели.

Тотална комбинаторна алгебра представлява четворка $\mathcal{D} = \langle D, \bullet, k, s\rangle$, съдържаща множество D от *елементи*, функция за прилагане $\bullet$, която е от тип $D \to (D \to D)$, и различни елементи k и $s \in D$, такива че за всяко $d_1, d_2, d_3 \in D$ са изпълнени

$$k \bullet d_1 \bullet d_2 = d_1,$$
$$s \bullet d_1 \bullet d_2 \bullet d_3 = (d_1 \bullet d_3) \bullet (d_2 \bullet d_3).$$

В този раздел ще работим с фиксирана, но не специфицирана тотална комбинаторна алгебра $\mathcal{D}$ (вж. [39] например).

Множеството от *чисти λ-термове* се дефинира от следната граматика:

$$M ::= x \mid \mathbf{fun}\,(x)M \mid M_1 M_2.$$

Множеството от *комбинаторни термове* е

$$C ::= x \mid C_1 C_2 \mid K \mid S.$$

Абстракция на комбинаторния терм C по отношение на променливата x, която ще означаваме с $\mathbf{fun}^\star x.C$, се дефинира по следния начин:

$$\begin{array}{lll} \mathbf{fun}^\star x.C & = KC, & \text{където } x \notin \mathrm{FV}(C), \\ \mathbf{fun}^\star x.x & = SKK, & \\ \mathbf{fun}^\star x.C_1C_2 & = S(\mathbf{fun}^\star x.C_1)(\mathbf{fun}^\star x.C_2), & \text{където } x \notin \mathrm{FV}(C_1C_2). \end{array}$$

Комбинаторна транслация на чист λ-терм M, означена $|M|$, се дефинира, както следва:

$$\begin{array}{ll} |x| & = x, \\ |\mathbf{fun}\,(x).M| & = \mathbf{fun}^\star(x)|M|, \\ |M_1M_2| & = |M_1||M_2|. \end{array}$$

Терм-среда η е крайна функция, която изобразява термовите променливи в елементи от D. Когато $x \notin \mathrm{dom}(\eta)$, ще пишем $\eta[x \leftarrow d]$ за среда, която изобразява x в d и съвпада с η навсякъде другаде. Ще пишем $\eta\backslash x$ за среда, която е същата, каквато е и η с изключение, че $\eta(x)$ е недефинирано; $\eta\backslash\Gamma$ е среда, подобна на η, но недефинирана върху всички променливи в $\mathrm{dom}(\Gamma)$. Ще казваме, че η' *разширява* η, когато $\mathrm{dom}(\eta) \subseteq \mathrm{dom}(\eta')$ и η и η' съвпадат в $\mathrm{dom}(\eta)$.

Нека C е комбинаторен терм, а η е терм-среда, такава че $\mathrm{FV}(C) \subseteq \mathrm{dom}(\eta)$. *Интерпретация* на C под η, означено като $[\![C]\!]_\eta$, се дефинира, както следва:

$$\begin{array}{ll} [\![x]\!]_\eta & = \eta(x), \\ [\![C_1C_2]\!]_\eta & = [\![C_1]\!]_\eta \bullet [\![C_2]\!]_\eta, \\ [\![K]\!]_\eta & = k, \\ [\![S]\!]_\eta & = s. \end{array}$$

Лема 3.1.1. 1. *Ако η' разширява η и $\mathrm{FV}(C) \subseteq \mathrm{dom}(\eta)$, то $[\![C]\!]_\eta = [\![C]\!]_{\eta'}$.*
2. $[\![\mathbf{fun}^\star x.C]\!]_\eta \bullet m = [\![C]\!]_{\eta[x \leftarrow m]}$.

3.2. РЕЛАЦИИ НА ЧАСТИЧНА ЕКВИВАЛЕНТНОСТ ОТ ПО-ВИСОК РЕД

Релация на частична еквивалентност (РяЧЕ) върху комбинаторната алгебра $\mathcal{D}$ е симетрична и транзитивна релация A върху D. Ще записваме $m\,\{A\}\,n$, където A свързва m и n. *Дефиниционната област* на A, ще пишем $\mathrm{dom}(A)$, е множеството $\{n \mid n\,\{A\}\,n\}$. Отбележете, че от $m\,\{A\}\,n$ следва $m \in \mathrm{dom}(A)$. Ще означим с **PER** класа на всички РиЧЕ.

Ако A и B са релации, то $A \to B$ е релация, където $m\,\{A \to B\}\,n$, ако и само ако за всяко $p, q \in D$, от $p\,\{A\}\,q$ следва $m \bullet p\,\{B\}\,n \bullet q$.

Не е трудно да се покаже, че $A \to B$ е РяЧЕ, когато A и B са РиЧЕ, и че сечението на РиЧЕ е РяЧЕ.

За да интерпретираме типовите оператори, ще трябва да имаме предвид не само РиЧЕ, но и произволни функционални пространства, построени върху РиЧЕ. Елемент на такова функционално пространство (включвайки като специален елемент самото **PER**) се нарича *релация на частична еквивалентност от по-висок ред* (РиЧЕВР). Интерпретацията на сорт K е подходящо пространство от РиЧЕВР:

$$\begin{aligned} [\![\star]\!] &= \mathbf{PER}, \\ [\![K_1 \to K_2]\!] &= [\![K_1]\!] \to [\![K_2]\!]. \end{aligned}$$

Нека $\{A_i \in [\![K]\!]\}_{i \in I}$ е множество от РиЧЕВР, индексирани чрез множеството I. Тогава $\cap^K_{i \in I} A_i$ е РяЧЕВР, дефинирана чрез

$$\begin{aligned} &m\,\{\cap^\star_{i \in I} A_i\}\,n, \qquad \text{ако и само ако, когато за всяко } i,\ m\,\{A_i\}\,n, \\ &\cap^{K_1 \to K_2}_{i \in I} A_i = \lambda P \in [\![K_1]\!].\, \cap^{K_2}_{i \in I} A_i P. \end{aligned}$$

Всяка колекция $[\![K]\!]$ от РиЧЕВР има максимален елемент, който се означава с $\mathbf{Top}^K$ и може да се пресметне по следния начин:

- $\mathbf{Top}^\star$ е тотална релация върху D.
- $\mathbf{Top}^{K_1 \to K_2} = \lambda P \in [\![K_1]\!].\mathbf{Top}^{K_2}$.

3.3. ИНТЕРПРЕТАЦИЯ НА $F^\omega_\wedge$ ЧРЕЗ РиЧЕВР

Среда η е крайна функция от типови променливи в РиЧЕВР и от термовите променливи в елементите на D. Нотациите за разширяване, ограничение и съгласуване на средата са пренесени от терм-средата. Чрез „претоварване“ с нотацията типовите среди ще се използват вместо терм-средите отсега нататък.

Изтриване на $F^\omega_\wedge$-терм e, означено $\mathrm{erase}(e)$, е чист λ-терм, дефиниран, както следва:

$$\begin{aligned} \mathrm{erase}(x) &= x, \\ \mathrm{erase}(\mathbf{fun}\,(x : T)e) &= \mathbf{fun}\,(x)\mathrm{erase}(e), \\ \mathrm{erase}(e_1\, e_2) &= \mathrm{erase}(e_1)\mathrm{erase}(e_2), \\ \mathrm{erase}(\mathbf{all}\,(X : K)e) &= \mathrm{erase}(e), \\ \mathrm{erase}(e\,T) &= \mathrm{erase}(e), \\ \mathrm{erase}(\mathbf{for}\,(X \in T_1..T_n)e) &= \mathrm{erase}(e). \end{aligned}$$

Нека η е терм-среда, а e е израз, такъв че $\mathrm{FV}(e) \subseteq \mathrm{dom}(\eta)$. Тогава *интерпретация* на e под η, записано с $[\![e]\!]_\eta$, е $[\![\mathrm{erase}(e)]\!]_\eta$.

Нека η е оценка, а T е типов израз, такъв че $\mathrm{FTV}(T) \subseteq \mathrm{dom}(\eta)$. *Интерпретация* $[\![T]\!]_\eta$ на T под η е релация на частична еквивалентност от по-висок ред, дефинирана както следва:

$$
\begin{array}{ll}
[\![X]\!]_\eta & = \eta(X), \\
[\![T_1 \to T_2]\!]_\eta & = [\![T_1]\!]_\eta \to [\![T_2]\!]_\eta, \\
[\![\mathbf{all}\,(X : K)T]\!]_\eta & = \cap^\star_{P \subseteq^K [\![T]\!]_\eta} [\![T_2]\!]_{\eta[X \leftarrow P]}, \\
[\![\mathbf{for}\,(X \in T_1..T_n)T]\!]_\eta & = \cap^\star_{1 \le i \le n} [\![T]\!]_{\eta[X \leftarrow [\![T_i]\!]_\eta]}, \\
[\![\wedge^K [T_1..T_n]]\!]_\eta & = \cap^K_{1 \le i \le n} [\![T_i]\!]_\eta, \\
[\![ST]\!]_\eta & \doteq [\![S]\!]_\eta [\![T]\!]_\eta, \\
[\![\mathbf{fun}\,(X : K)T]\!]_\eta & = \lambda P \in [\![K]\!].[\![T]\!]_{\eta[X \leftarrow P]}.
\end{array}
$$

Разбира се, за някои η-ти и T-та интерпретацията $[\![T]\!]_\eta$, описана с тези равенства, е безсмислена. Ще пишем $[\![T]\!]_\eta \downarrow$, когато $[\![T]\!]_\eta$ е добре дефинирано, и $[\![T]\!]_\eta \uparrow$ — в противен случай.

Ще казваме, че оценката η *удовлетворява* контекст Γ, което ще означаваме $\eta \models \Gamma$, ако $\mathrm{dom}(\eta) = \mathrm{dom}(\Gamma)$ и

1. $\Gamma \equiv \varnothing$, или
2. $\Gamma \equiv \Gamma_1, x : T$, където $\eta \backslash x$ удовлетворява Γ_1 и или $[\![T]\!]_{\eta\backslash x}\uparrow$, или $\eta(x) \in \mathrm{dom}([\![T]\!]_{\eta\backslash x})$, или
3. $\Gamma \equiv \Gamma_1, X : K$, където $\eta \backslash X$ удовлетворява Γ_1 и или $[\![T]\!]_{\eta\backslash X}\uparrow$, или $\eta(X) \subseteq^K [\![T]\!]_{\eta\backslash X}$.

От дефиницията веднага следва, че или $[\![T]\!]_{\eta\backslash\Gamma_2\backslash X}\uparrow$, или $[\![T]\!]_{\eta\backslash\Gamma_2\backslash X} \in [\![K]\!]$, когато $\eta \models \Gamma_1, X : K, \Gamma_2$. Също отбележете, че

Лема 3.3.1. *Ако η' разширява η и $\mathrm{FTV}(T) \subseteq \mathrm{dom}(\eta)$, то или $[\![T]\!]_\eta\uparrow$ и $[\![T]\!]_{\eta'}\uparrow$, или и двете са дефинирани и $[\![T]\!]_\eta = [\![T]\!]_{\eta'}$.*

Лема 3.3.2. *Нека T е тип, Γ е контекст, а η е оценка, такава че $\mathrm{FTV}(T) \subseteq \mathrm{dom}(\eta)$ и $\eta \models \Gamma$. Ако $\Gamma \vdash T \in K$, то $[\![T]\!]_\eta\downarrow$ и $[\![T]\!]_\eta \in [\![K]\!]$.*

Лема 3.3.3. *Нека η е оценка с $X \notin \mathrm{dom}(\eta)$ и такова, че $\mathrm{FTV}(S[X \leftarrow T]) \subseteq \mathrm{dom}(\eta)$ и $[\![S]\!]_{\eta[X \leftarrow [\![T]\!]_\eta]}\downarrow$. Тогава $[\![S[X \leftarrow T]]\!]_\eta = [\![S]\!]_{\eta[X \leftarrow [\![T]\!]_\eta]}$.*

Лема 3.3.4. *Нека η е оценка, такава че $\mathrm{FTV}(S) \subseteq \mathrm{dom}(\eta)$. Ако $S =_\beta T$ и $[\![S]\!]_\eta\downarrow$, то $[\![S]\!]_\eta = [\![T]\!]_\eta$.*

Типов контекст Γ/TV, *получен от контекста* Γ, се дефинира по следния очевиден начин:

$$
\begin{array}{l}
\varnothing/\mathrm{TV} = \varnothing, \\
(\Gamma, x : T)/\mathrm{TV} = \Gamma/\mathrm{TV}, \\
(\Gamma, X : K)/\mathrm{TV} = \Gamma/\mathrm{TV}, X : K.
\end{array}
$$

Теорема 3.3.5. *Нека* $\eta_1 \models \Gamma$ *и* $\eta_2 \models \Gamma$ *са такива, че* $\eta_1/\mathrm{TV} = \eta_2/\mathrm{TV}$ *и за всяко* $x \in \mathrm{dom}(\Gamma)$, $\eta_1(x)\,\{[\![\Gamma(x)]\!]_\eta\}\,\eta_2(x)$, *където* $\eta = \eta_1/\mathrm{TV}$. *Тогава* $[\![e]\!]_{\eta_1}\,\{[\![T]\!]_\eta\}\,[\![e]\!]_{\eta_2}$.

Доказателство. От $\Gamma \vdash e \in T$ и лема 2.3.9 следва, че $\Gamma \vdash T \in \star$. Ще отбележим, че $\mathrm{FTV}(T) \subseteq \mathrm{dom}(\Gamma/\mathrm{TV})$. От лема 2.3.5 (3) за усилването на контекст имаме $\Gamma/\mathrm{TV} \vdash T \in \star$. Ще отбележим също, че $\eta \models \Gamma/\mathrm{TV}$. Съгласно лема 3.3.2 $[\![T]\!]_\eta$ е дефинирано и е в $[\![\star]\!]$. Този факт ще използваме неявно в доказателството, което ще извършим чрез индукция по извода $\Gamma \vdash e \in T$.

1. T-Var ($e = x$). От условието на теоремата имаме, че за всяко $x \in \mathrm{dom}(\Gamma)$ е изпълнено, че $\eta_1\,\{[\![\Gamma(x)]\!]_\eta\}\,\eta_2(x)$. Като имаме предвид, че $\Gamma(x) = T$, и от дефиницията на семантичната функция $[\![x]\!]_{\eta'} = \eta(x)$ за всяко η получаваме $[\![x]\!]_{\eta_1}\,\{[\![T]\!]_\eta\}\,[\![x]\!]_{\eta_2}$, което е твърдението в този случай.

2. T-Abs ($e = \mathbf{fun}\,(x : T_1)e$). В правилото T-Abs е дадено, че $\Gamma, x : T_1 \vdash e \in T_2$. Да предположим, че $p \in D$ и $q \in D$ са такива, че $p\,\{[\![T_1]\!]_\eta\}\,q$. Тогава $\eta_1[x \leftarrow p] \models \Gamma, x : T_1$ и $\eta_2[x \leftarrow q] \models \Gamma, x : T_1$. От индукционната хипотеза получаваме $[\![e]\!]_{\eta_1[x\leftarrow p]}\,\{[\![T_2]\!]_\eta\}\,[\![e]\!]_{\eta_2[x\leftarrow q]}$, което е

$$[\![|\mathrm{erase}(e)|]\!]_{\eta_1[x\leftarrow p]}\,\{[\![T_2]\!]_\eta\}\,[\![|\mathrm{erase}(e)|]\!]_{\eta_2[x\leftarrow q]}.$$

От лема 3.1.1 (2) следва, че

$$[\![\mathbf{fun}^\star(x)|\mathrm{erase}(e)|]\!]_{\eta_1} \bullet p\,\{[\![T_2]\!]_\eta\}\,[\![\mathbf{fun}^\star(x)|\mathrm{erase}(e)|]\!]_{\eta_2} \bullet q,$$

което е

$$[\![\mathbf{fun}\,(x : T_1)e]\!]_{\eta_1} \bullet p\,\{[\![T_2]\!]_\eta\}\,[\![\mathbf{fun}\,(x : T_1)e]\!]_{\eta_2} \bullet q.$$

Понеже p и q са произволно избрани, имаме

$$[\![\mathbf{fun}\,(x : T_1)e]\!]_{\eta_1}\,\{[\![T_1]\!]_\eta \to [\![T_2]\!]_\eta\}\,[\![\mathbf{fun}\,(x : T_1)e]\!]_{\eta_2},$$

с други думи,

$$[\![\mathbf{fun}\,(x : T_1)e]\!]_{\eta_1}\,\{[\![T_1 \to T_2]\!]_\eta\}\,[\![\mathbf{fun}\,(x : T_1)e]\!]_{\eta_2}.$$

3. T-App ($e = f\,a$). Като използваме, че $[\![f\,a]\!]_\eta = [\![f]\!]_\eta [\![a]\!]_\eta$, твърдението следва от индукционната хипотеза.

4. T-TAbs ($e = \mathbf{fun}(X : K)e$). От правилото T-TAbs имаме, че $\Gamma, X : K \vdash e \in T$. Да предположим, че P е произволна РяЧЕВР от сорт K. Тогава $\eta_1[X \leftarrow P] \models \Gamma, X : K$ и $\eta_2[X \leftarrow P] \models \Gamma, X : K$. Понеже $[\![e]\!]_{\eta_1} = [\![e]\!]_{\eta_1[X\leftarrow P]}$, от индукционната хипотеза имаме, че $[\![e]\!]_{\eta_1}\,\{[\![T]\!]_{\eta[X\leftarrow P]}\}\,[\![e]\!]_{\eta_2}$. Тъй като P беше избрано произволно, имаме, че $[\![e]\!]_{\eta_1}\,\{\cap^\star_{P\subseteq[\![K]\!]}[\![T]\!]_{\eta[X\leftarrow P]}\}\,[\![e]\!]_{\eta_2}$. Резултатът следва от дефиницията на $[\![-]\!]_\eta$ и факта, че $[\![\mathbf{fun}\,(X : K)e]\!]_\eta = [\![e]\!]_\eta = [\![e]\!]_{\eta[X\leftarrow P]}$.

5. T-TApp ($e = f\,S$). От правилото T-TApp ни е дадено, че $\Gamma \vdash f \in \mathbf{all}\,(X : K)T$ и $\Gamma \vdash S \in K$. Според индукционната хипотеза

$$[\![f]\!]_{\eta_1}\,\{[\![\mathbf{all}\,(X : K)T]\!]_\eta\}\,[\![f]\!]_{\eta_2},$$

което е

$$[\![f]\!]_{\eta_1}\,\{\cap^\star_{P\subseteq[\![K]\!]}[\![T]\!]_{\eta[X\leftarrow P]}\}\,[\![f]\!]_{\eta_2}.$$

Понеже $\Gamma \vdash S \in K$, получаваме $[\![f]\!]_{\eta_1}\{[\![T]\!]_{\eta[X\leftarrow[\![S]\!]_\eta]}\}[\![f]\!]_{\eta_2}$. От лема 3.3.3 следва $[\![T]\!]_{\eta[X\leftarrow[\![S]\!]_\eta]} = [\![T[X \leftarrow S]]\!]_\eta$. Като използваме, че $[\![f]\!]_\eta = [\![f\,S]\!]_\eta$ за всяко η, получаваме $[\![f\,S]\!]_{\eta_1}\{[\![T[X \leftarrow S]]\!]_\eta\}[\![f\,S]\!]_{\eta_2}$.

6. T-For ($e = \mathbf{for}(X \in S_1..S_n)e$). Правилото T-For ни дава, че $\Gamma \vdash e[X \leftarrow S] \in T$ за всяко $S \in \{S_1..S_n\}$. От индукционната хипотеза и факта, че $\mathrm{erase}(e[X \leftarrow S]) = \mathrm{erase}(e)$, получаваме твърдението.

7. T-Meet-I. Дадено е, че $\Gamma \vdash$ дфк и за всяко $i = 1..n$, $\Gamma \vdash e \in T_i$. От индукционната хипотеза следва, че за всяко $i = 1..n$ имаме $[\![e]\!]_{\eta_1}\{[\![T_i]\!]_\eta\}[\![e]\!]_{\eta_2}$. От дефиницията на $\cap^\star$ получаваме, че $[\![e]\!]_{\eta_1}\{\cap^\star_{1\le i\le n}[\![T_i]\!]_\eta\}[\![e]\!]_{\eta_2}$, а от дефиницията на семантичната функция следва, че $[\![e]\!]_{\eta_1}\{[\![\wedge^\star[T_1..T_n]]\!]_\eta\}[\![e]\!]_{\eta_2}$.

8. T-Meet-E. Твърдението следва веднага, като разсъжденията се извършват в обратен ред на 7. T-Meet-I.

Следствие 3.3.6 (пълнота на типизирането). *Нека η е оценка, такава че $\eta \models \Gamma$. Тогава от $\Gamma \vdash e \in T$ следва, че $[\![e]\!]_\eta \in \mathrm{dom}([\![T]\!]_\eta)$.*

Доказателство. $\eta_1 = \eta_2 = \eta$.

4. ЗАКЛЮЧЕНИЕ

Езикът от типови конструктори е система от комбинатори без редукционни правила. Това позволява да се използва стандартна техника, за да се изведе сортът на конструкторните променливи и константи, въведени чрез новите типови дефиниции. Съществен момент е, че не е необходимо потребителят да специфицира информация за сорта явно. В такъв случай задачата за определяне на сорта на даден типов израз се изпълнява от специална част от процедурата за проверка на тип, наречена *извод на сорт*.

Автоматичното извеждане на сорта на даден типов израз считаме за значително предимство, тъй като това означава, че програмистът се освобождава от необходимостта да специфицира сорта на типовите изрази.

Съчетанието на претоварване с параметричен полиморфизъм повишава изразителността на език със сечение на типове.

ЛИТЕРАТУРА

1. Barendregt, H. P. The Lambda Calculus. North-Holland, (revised edition), 1984.
2. Barendregt, H. Introduction to generalized type systems. *Journal of Functional Programming*, **1** (2), April 1991, 125–154.
3. Barendregt, H., M. Coppo, M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, **48** (4), 1983, 931–940,
4. Barbanera, F., M. Dezani-Ciancaglini. Intersection and union types. Theoretical aspects of Computer Software (Sendai Japan). *LNCS*, **526**, Springer Verlag, September, 1991, 651–674.

5. Bruce, K. B., G. Longo. A modest model of records, inheritance, and bounded quantification. *Information and Computation*, **87**, 1990, 196–240.
6. Bruce, K. B., A. R. Meyer, J. Mitchell. The semantics of second-order lambda calculus. *Information and Computation*, **85** (1), 1990, 76–134.
7. Cartwright, R., M. Fagan. Soft Typing. In: Proceedings of the SIGPLAN'91 Conference on Language Design and Implementation, Toronto, Ontario. ACM Press, June 26-28, 1991, 278–292.
8. Castagna, G., G. Ghelli, G. Longo. A calculus for overloaded functions with subtyping. In: ACM Conference on Lisp and Functional Programming, San Francisco, July 1992, ACM Press. Extended abstract, 182–192.
9. Castagna, G., G. Ghelli, G. Longo. A semantics for $\lambda\&$-early: a calculus with overloading and early binding. In: Proceedings of International Conference on Typed Lambda Calculi and Applications, TLCA'93, March 1993, Utrecht, The Netherlands, *LNCS*, **664**, 107–123.
10. Coppo, M., M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre-Dame Journal of Formal Logic*, **21** (4), October 1980, 685–693.
11. Coppo, M., M. Dezani-Ciancaglini, M. Zacchi. Type theories, normal forms and D_∞-lambda-models. *Information and Computation*, **72**, 1987, 85–116.
12. Coppo, M., M. Dezani-Ciancaglini, P. Salle. Functional characterization of some semantic equalities inside λ-calculus. *LNCS*, **81**, Springer-Verlag, 1979, 133–146.
13. Coppo, M., P. Giannini. A complete type inference algorithm for simple intersection types. In: ESOP'92, 1992.
14. Dezani-Ciancaglini, M., I. Margaria. F-semantics for intersection type discipline. In: G. Kahn, D. B. MacQueen and G. Plotkin, Semantics of Data Types. *LNCS*, **173**, Springer-Verlag, 1984, 279–300.
15. Dezani-Ciancaglini, M., I. Margaria. A characterization of F-complete type assignments. *Theoretical Computer Science*, **45**, 1986, 121–157.
16. Fagan, M. Soft Typing: An Approach to Type Checking for Dynamically Typed Languages. PhD Thesis, Rice University, December 1990.
17. Freeman, T., F. Pfenning. Refinement types for ML. In: Proceedings of the SIGPLAN'91 Conference on Language Design and Implementation. Toronto, Ontario. ACM Press, June 26–28, 1991, 268–277.
18. Hayashi, S. Singleton, union and intersection types for program extraction. Theoretical aspects of Computer Software (Sendai Japan). *LNCS*, **526**, Springer Verlag, September, 1991, 701–730.
19. Hayashi, S., Y. Takayama. Extended projection method with Kreisel-Troelstra realizability. Information and Computation, 1990.
20. Hindley, J. R. The simple semantics for Coppo-Dezani-Salle types. In: Proceedings of the International Simposium on Programming, *LNCS*, **137**, 1982, 212–226.
21. Hudak, P., S. L. P. Jones, P. Wadler. Report on the programming language Haskell, version 1.2. *ACM SIGPLAN Notices*, **27** (5), May 1992.
22. Jones, M. P. A theory of qualified types. In: European Symposium on programming, *LNCS*, **582**, Springer Verlag, 1992.
23. Jones, M. P. Qualified types: Theory and Practice. PhD Thesis, Programming Research Group, Oxford University Computing Laboratory, July 1992.
24. Jones, M. P. A system of constructor classes: overloading and implicit higher-order polymorphism. In: Proceedings of the ACM Conference on Functional Programming and Computer Architectures, Copenhagen, Danemark, June 1993, 52–61.
25. Kaes, S. Parametric overloading in polymorphic programming languages. *LNCS*, **300**, 1988, 131–144.
26. Leivant, D. Discrete polymorphism (summary). In: Proceedings of the 1990 ACM Conference on Lisp and Functional Programming, 1990, 288–297.
27. Hindley, J. R., J. P. Seldin. Introduction to Combinators and λ-calculus. London mathematical Society Student Texts, vol. 1, Cambridge University Press, 1986.
28. Mitchell, J. A type-innference approach to reduction properties and semantics of polymorphic expressions. In: Logical Foundations of Functional Programming, Gerard Huet (ed.),

University of Texas at Austin Year of Programming Series, Addison-Wesley, 1990, 195–212.
29. Peterson, J., M. Jones. Implementing type classes. In: Proceedings of the ACM SIGPLAN Conference on Programming Languages, Design and Implementation, Albuquerque, June 1993, 227–236.
30. Pierce, B. C. A decision procedure for the subtype relation on intersection types with bounded variables. Technical Report CMU-CS-89-169, School of Computer Science, Carnegie Mellon University, September 1989.
31. Pierce, B. C. Programming with intersection types and bounded polymorphism. PhD thesis, Carnegie Mellon University, December 1991.
32. Pierce, B. C. Programming with intersection types, union types and polymorphism. Technical Report CMU-CS-91-106, Carnegie Mellon University, February 1991.
33. Pierce, B. C. Intersection types and bounded polymorphism. In: Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93, March 1993, Utrecht, The Netherlands. *LNCS*, **664**, 346–360.
34. Pottinger, G. A type assignment for the strongly normalizable λ-terms. In: To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism, Academic Press, New York, 1980, 561–577.
35. Reynolds, J. Preliminary design of the programming language Forsythe. Technical Report CMU-CS-88-159, Carnegie Mellon University, June 1988.
36. della Rocca, S. Ronchi. Principal type scheme and unification for intersection type discipline. *Theoretical Computer Science*, **59**, 1988, 181–209.
37. della Rocca, S. Ronchi, B. Venneri. Principal type schemes for ann extended type theory. *Theoretical Computer Science*, **28**, 1984, 151–169.
38. Salle, P. Une extension de la theorie des types en λ-calcul. *LNCS*, **62**, Springer-Verlag, 1982, 398–410.
39. Scott, D. Data types as lattices. *SIAM Journal on Computing*, **5** (3), 1976, 522–587.
40. Wadler, P., S. Blott. How to make "ad-hoc" polymorphism less "ad-hoc". In: 16th Ann. ACM Symp. on Principles of Programming Languages, 1989, 60–76.
41. DoD ANSI/MIL-STD-1815 A-1983. Reference Manual for the Ada Programming Language.
42. Биров, Д. Параметрично претоварване на рекурсивни дефиниции. 23-та пролетна конференция на Съюза на математиците в България, Стара Загора, 1–4 април 1994.
43. Биров, Д. Параметрично претоварване на функции от по-висок ред. 23-та пролетна конференция на Съюза на математиците в България, Стара Загора, 1–4 април 1994.
44. Тодорова, М., Д. Биров. Алгоритъм за извод на тип в езика за програмиране с равенства и унификация W. В този том на *Годишника*.

Постъпила 28.03.1994