UNIVERZITET U NOVOM SADU

FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

mr Jelena Ivetić

INTERSECTION TYPES AND RESOURCE CONTROL IN THE INTUITIONISTIC SEQUENT LAMBDA CALCULUS

TIPOVI SA PRESEKOM I KONTROLA RESURSA U INTUICIONISTIČKOM SEKVENTNOM LAMBDA RAČUNU

- DOKTORSKA DISERTACIJA -

Mentor:
Prof. dr Silvia Gilezan

Novi Sad, 2013

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:	Монографска публикација	
Тип записа, **ТЗ**:	Текстуални штампани материјал	
Врста рада, **ВР**:	Докторска дисертација	
Аутор, **АУ**:	мр Јелена Иветић	
Ментор, **МН**:	Проф. др Силвиа Гилезан	
Наслов рада, **НР**:	Типови са пресеком и контрола ресурса у интуиционистичком секвентном ламбда рачуну	
Језик публикације, **ЈП**:	Енглески	
Језик извода, **ЈИ**:	Српски, Енглески	
Земља публиковања, **ЗП**:	Република Србија	
Уже географско подручје, **УГП**:	Војводина	
Година, **ГО**:	2013	
Издавач, **ИЗ**:	Ауторски репринт	
Место и адреса, **МА**:	ФТН, Трг Доситеја Обрадовића 6, Нови Сад	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)	7/170/75/0/48/0/0	
Научна област, **НО**:	Примењена математика	
Научна дисциплина, **НД**:	Логика у рачунарству	
Предметна одредница/Кључне речи, **ПО**:	Ламбда рачун, секвентни рачун, типови са пресеком, контрола ресурса	
УДК		
Чува се, **ЧУ**:	Библиотека Факултета техничких наука у Новом Саду	
Важна напомена, **ВН**:		
Извод, **ИЗ**:	Ова дисертација се бави рачунским интерпретацијама интуиционистичког секвентног рачуна са имплицитним и експлицитним структурним правилима, са фокусом на типске системе са пресеком. Оригинални резултати тезе су груписани у три целине. У првом делу су типови са пресеком уведени у *lambda Gentzen* рачун. Други део представља проширење *lambda Gentzen* рачуна на формални рачун са контролом ресурса, тј. са експлицитним операторима контракције и слабљења, као и одговарајући типски систем са пресеком који карактерише јаку нормализацију у уведеном рачуну. У трећем делу оба рачуна су интегрисана у заједнички оквир увођењем структуре *resource control cube.*	
Датум прихватања теме, **ДП**:	29.11.2012.	
Датум одбране, **ДО**:	09.10.2013.	
Чланови комисије, **КО**: Председник:	др Јованка Пантовић, редовни професор	
Члан:	др Pierre Lescanne, редовни професор	
Члан:	др José Espírito Santo, редовни професор	Потпис ментора
Члан:	др Silvia Likavec, доцент	
Члан, ментор:	др Силвиа Гилезан, редовни професор	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	Monographic publication
Type of record, **TR**:	Textual printed material
Contents code, **CC**:	PhD thesis
Author, **AU**:	Jelena Ivetić, M.Sc.
Mentor, **MN**:	Prof Silvia Gilezan, Ph.D
Title, **TI**:	Intesection types and resource control in the intuitionistic sequent lambda calculus
Language of text, **LT**:	English
Language of abstract, **LA**:	Serbian, English
Country of publication, **CP**:	Republic of Serbia
Locality of publication, **LP**:	Province of Vojvodina
Publication year, **PY**:	2013.
Publisher, **PB**:	Author’s reprint
Publication place, **PP**:	Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)	7/170/75/0/48/0/0
Scientific field, **SF**:	Applied mathematics
Scientific discipline, **SD**:	Logic in computer science
Subject/Key words, **S/KW**:	Lambda calculus, sequent calculus, intersection types, resource control
UC	
Holding data, **HD**:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, **N**:	
Abstract, **AB**:	This thesis studies computational interpretations of the intuitionistic sequent calculus with implicit and explicit structural rules, with focus on the systems with intersection types. The contributions of the thesis are grouped into three parts. In the first part intersection types are introduced into the lambda Gentzen calculus. The second part presents an extension of the lambda Gentzen calculus to a term calculus with resource control, i.e. with explicit operators for contraction and weakening, and apropriate intersection type assignment system which characterises strong normalisation in the proposed calculus. In the third part both previously studied calculi are integrated into one framework by introducing the notion of the resource control cube.
Accepted by the Scientific Board on, **ASB**:	November 29th, 2012.
Defended on, **DE**:	October 9th, 2013.

Defended Board, **DB**:	President:	Jovanka Pantović, Ph.D, full professor	
	Member:	Pierre Lescanne, Ph.D, full professor	
	Member:	José Espírito Santo, Ph.D, full professor	Menthor's sign
	Member:	Silvia Likavec, Ph.D, ass. professor	
	Member, Mentor:	Silvia Gilezan, Ph.D, full professor	

I dedicate this to my loving grandmother Majka,
from whom I inherited affinity to mathematics,
and hopefully persistence and a need
to continuously improve myself.

Acknowledgements

First and foremost, I want to thank my mentor, professor Silvia Ghilezan, who introduced me to wonders of the world of logics, and unselfishly shared her knowledge and experience with me. Silvia, thank you for all your time, patience, ideas, advices and support. Your influence to my professional and personal development greatly exceeds the regular role of a supervisor.

Next, I want to thank professors Pierre Lescanne, José Espírito Santo, Silvia Likavec and Dragiša Žunić for giving me the opportunity to cooperate with them and learn from them. I hope that our collaboration will continue.

Further, I'm grateful to my defence committee members: professor Jovanka Pantović, professor Silvia Likavec, professor Pierre Lescanne and professor José Espírito Santo, for their interest, time, careful reading and valuable comments. I'm particularly grateful to Silvia for her help with language-related part of the dissertation.

I thank my dear colleagues from the Chair for Mathematics and the Department for Fundamental Disciplines for the great atmosphere and the best working environment. My efforts to complete the thesis were mostly motivated by my wish to stay part of the team.

Finally, I would like to thank my family, for all their love and encouragement, and for providing me the continuous support throughout all these years.
Marko, Damjane, Kalina, Naco, Dedika, Biso, Nešo, Ildi, Ana, Rico - hvala vam!

Rezime

Pravila zaključivanja su bila poznata, a pojedina čak i formalno opisana, još u antička vremena. Najstariji poznati dokument koji se bavi ovom problematikom je Organon, zbirka Aristotelovih zapisa o logici.

Proučavanje logike kao matematičke discipline datira iz druge polovine devetnaestog veka, kada je Gottlob Frege uveo osnovne koncepte savremene logike, proširujući i razvijajući ideje do kojih je došao Gottfried Leibniz. Logika je doživela intenzivan razvoj početkom dvadesetog veka, a najznačajniji logičari tog vremena su bili Gerhard Gentzen, David Hilbert, Bertrand Russel i posebno Kurt Gödel, čije su teoreme o nekompletnosti ukazale na ograničenja formalnih aritmetičkih sistema. Tokom tod perioda su uvedena tri osnovna formalna sistema za dokazivanje: Hilbertov aksiomatski sistem i dva Gentzenova sistema - prirodna dedukcija i sekventni račun.

Dvadesetih godina dvadesetog veka nastaju prvi rezultati iz oblasti teorijskog računarstva, koje se od tada razvija istovremeno sa logikom. Interakcija između ove dve naučne discipline se odvijala na više nivoa, stvarajući pozitivnu spregu koja je značajno doprinela razvoju obe. U to vreme nastaju različiti formalni računi, kao posledica težnje matematičara tog vremena da formalizuju celokupnu matematiku (što se kasnije ispostavilo nemogućim, što je i dokazao Gödel). Tokom ovog perioda se pojavio kombinatorni račun čiji su autori Moses Schönfinkel i Haskell Curry. Ubrzo potom usledio je i λ-račun, koji je uveo Alonso Church u [10]. Church je 1940. godine uveo i osnovne tipove u λ-račun [11], čime je nastao sistem koji je kasnije postao osnovni formalizam na kome su zasnovani funkcionalni programski jezici, počevši od jezika Algol 60, čiji je autor Peter Landin [50].

Korespondenciju između logike i λ-računa je prvi uočio Curry 1934. godine, kada je pokazao da postoji izomorfizam između skupa formula dokazivih u implikativnom fragmentu intuicionističke logike i tipova koji se u osnovnom tipskom sistemu mogu dodeliti zatvorenim termima. Ovu korespondenciju je 1969. godine proširio Howard. Prema Curry-Howardovoj korespondenciji, koja je zvanično objavljena tek 1980. godine u [40], prirodna dedukcija za intuicionističku logiku i λ-račun sa osnovnim tipovima su višestruko odgovarajući sistemi, u smislu da dokazi

formula odgovaraju specifikaciji programa, dok pojednostavljenja dokaza odgovaraju izvršavanju programa. Ova paradigma je još više naglasila značaj logike za razvoj i formalno utemeljenje računarstva, time što je ukazala na to da je teorija dokaza jedno od glavnih oruđa u analizi programa.

Prirodno, pojavio se interes i za proširivanjem Curry-Howardove korespondencije tj. za stvaranjem računskih interpretacija drugih logičkih sistema dokazivanja. Dok je veza između kombinatornog računa i Hilbertovog aksiomatskog sistema bila veoma očigledna, situacija je bila znatno drugačija kada je Gentzenov sekventni račun u pitanju. Naime, već je sam Gentzen dokazao da su sistemi prirodne dedukcije i sekventnog računa ekvipotentni, u smislu da je svaki prirodno deduktivni dokaz moguće prevesti u sekventni dokaz, kao i obrnuto. Zucker [74] i Pottinger [56] su pokazali da su i procesi pojednostavljivanja dokaza u dva sistema odgovarajući, tačnije da je normalizacija u prirodnoj dedukciji homomorfna slika eliminacije pravila sečenja u sekventnom računu. Stoga je postalo jasno da je moguće proširiti Curry-Howardovu korespondenciju na sekventni račun i neku varijantu λ-računa, nasuprot prethodno zastupljenim mišljenjima[1].

Međutim, budući da se sekventni račun i λ-račun strukturno značajno razlikuju, modifikacija je trebala da bude realizovana i na dubljem, sintaksnom nivou, a ne samo površinski, na nivou pravila tipiziranja. Prvi formalni račun kojim je uspostavljena korespondencija sa intuicionističkim sekventnim računom je bio $\bar{\lambda}$-račun, koji je konstruisao Hugo Herbelin 1995. godine u [38]. U ovom radu, uspostavljena je korespondencija između tipiziranih terma $\bar{\lambda}$-računa i sistema LJT, jedne restrikcije intuicionističkog sekventnog računa.

Prateći ideje na kojima je zasnovan $\bar{\lambda}$-račun, konstruisano je nekoliko drugih formalnih računa odgovarajućih sekventnom računu, u kojima je korespondencija proširena na potpuni intuicionistički sekventni račun LJ. Među ovim računima se ističe λ^{Gtz}-račun, koji je predložio José Espírito Santo [27] 2006. godine. Osnovna karakteristika sintakse λ^{Gtz}-računa je postojanje dve kategorije, terma t i konteksta k, čija je interakcija prisutna u aplikaciji, koja je oblika tk i koja se u ovom računu naziva sečenje (*cut*). Sa stanovišta operacijske semantike, karakterišu ga četiri redukcijska pravila usmerena ka eliminaciji sečenja, tj. smanjivanju termskog dela aplikacije. Značaj λ^{Gtz}-računa leži u činjenici da njegov osnovni tipski sistem u potpunosti odgovara intuicionističkom sekventnom računu, u smislu da su tipizirani izrazi λ^{Gtz}-računa u jedan-na-jedan korespondenciji sa sekventnim dokazima (uključujući i one koji sadrže pravilo sečenja).

[1] "...sekventni račun nema Curry-Howardov izomorfizam, zbog mnoštva načina zapisivanja istog dokaza. To nas sprečava da ga koristimo kao λ-račun sa tipovima, iako se u dubini nazire neka struktura te vrste..." Jean-Yves Girard, [37]

Osnovni predmet istraživanja sprovedenog u ovoj disertaciji su upravo računske interpretacije intuicionističkog sekventnog računa. Početnu tačku istraživanja stoga predstavlja λ^{Gtz}-calculus, kao elegantan i jednostavan sistem koji u potpunosti obuhvata računski sadržaj punog Gentzenovog sistema za implikativni fragment intuicionističke logike. Početni deo istraživanja je bio usmeren ka λ^{Gtz}-računu sa tipovima. Uveden je novi tipski sistem, čiji je osnovni cilj bio da obuhvati skup onih λ^{Gtz}-izraza čije su sve redukcije konačne. Drugim rečima, cilj je bio napraviti sistem u kome su skupovi tipiziranih izraza i izraza sa svojstvom jake normalizacije ekvivalentni. Sa tim ciljem, u λ^{Gtz}-račun su uvedeni *tipovi sa presekom*.

Tipovi sa presekom su prvi put uvedeni u λ-račun kasnih sedamdesetih godina. Nezavisno su ih razmatrali Coppo i Dezani [13], Sallé [63] i Pottinger [57]. Osnovna ideja uvođenja operatora preseka u sintaksu tipova je da se dozvoli mogućnost da neki term ima više od jednog tipa, i da mu se u tom slučaju može dodeliti i novi tip, koji predstavlja presek tih tipova. Ta jednostavna i prirodna ideja je pomogla da se uspešno prevaziđu pojedina ograničenja osnovnog tipskog sistema. Pre svega, novi tipski sistem je omogućio karakterizaciju svih terma sa osobinom jake normalizacije, za razliku od osnovnog tipskog sistema, koji ne može da okarakteriše čak ni skup normalnih formi, koji predstavlja strogi podskup skupa jako normalizovanih izraza.

Najpoznatiji, i prvi semantički kompletan tipski sistem sa presekom su predložili Barendregt, Coppo i Dezani [6]. U njihovom tipskom sistemu, tipovi se grade od tipskih promenljivih i konstante ω (koja predstavlja univerzalni tip) pomoću konstruktora $\rightarrow$ i $\cap$, a osim toga njegov sastavni deo čini i relacija poretka na tipovima $\leq$. Osim prvobitne namene u karakterizaciji jake normalizacije, tipovi sa presekom su se pokazali i kao veoma moćno oruđe u proučavanju bihejvioralnih osobina različitih sistema, preciznije u analizi i sintezi λ-modela. Takođe, našli su i primenu u programiranju, na primer u statičkoj analizi funkcionalnih programa.

Tipski sistem sa presekom za λ^{Gtz}-račun koji je predložen u ovoj disertaciji se razlikuje od standardnih tipskih sistema sa presekom po načinu na koji se tretira operator preseka. U standardnim tipskim sistemima sa presekom, postoje posebna pravila za uvođenje (i, u slučaju prirodno deduktivnih sistema, eliminaciju) operatora $\rightarrow$ koja korespondiraju pravilima za implikaciju u logičkim sistemima dokazivanja, i postoje posebna pravila za uvođenje i eliminaciju preseka, koja nemaju svoje logičke pandane. Zbog toga ovi sistemi nemaju sintaksnu direktnost - osobinu po kojoj svakom pravilu za građenje terma odgovara tačno jedno pravilo za dodelu tipova.

Nasuprot tome, u našem sistemu $\lambda^{\mathsf{Gtz}}\cap$, prikazanom na Slici 4.5, presek je ugrađen u već postojeća pravila osnovnog tipskog sistema $\lambda^{\mathsf{Gtz}}\rightarrow$ (Slika 4.4), čime je očuvan prvobitni broj pravila za dodelu tipova, i samim tim, sintaksna usmerenost sistema. Posledica sintaksne usmerenosti je jednoznačna i trivijalno dokaziva Lema o generisanju 4.16, koja potom u značajnoj meri olakšava dokaze tvrđenja vezanih za ovaj sistem.

Druga razlika predloženog tipskog sistema u odnosu na standardni sistem iz [6] je odsustvo univerzalnog tipa ω. Dokazano je da u predloženom sistemu $\lambda^{\mathsf{Gtz}}\cap$ važi da λ^{Gtz}-izraz ima tip ako i samo ako ima osobinu jake normalizacije (direktna posledica Teorema 4.41 i 4.47), čime je ispunjen osnovni cilj prvog dela ovog istraživanja.

Cilj druge faze istraživanja je bio proširiti λ^{Gtz}-račun kako bi se dobio novi formalni račun kojim bi bila uspostavljena Curry-Howardova korespondencija sa drugom verzijom intuicionističkog sekventnog računa. Reč je o sekventnom računu sa eksplicitnim strukturnim pravilima, koji je takođe uveo Gentzen [29].

Naime, dok logička pravila sekventnog računa uvode logičke veznike (u našem slučaju samo implikaciju) sa leve ili sa desne strane sekventa, strukturna pravila (kontrakcija, slabljenje i zamena) vrše transformacije direktno na sekventima, a ne na pojedinim formulama u okviru sekvenata. U slučaju intuicionističkog sekventnog računa, ove transformacije se vrše samo sa leve strane sekvenata, i podrazumevaju brisanje duplikata formule koja se javlja više puta (kontrakcija), dodavanje proizvoljne formule koja ne utiče na zaključak tj. formulu sa desne strane (slabljenje) i permutaciju redosleda kojim su navedene formule sa leve strane (zamena). Prva dva strukturna pravila stoga vrše kvantitativnu transformaciju sekventa, te se mogu posmatrati i proučavati odvojeno od pravila zamene. U formalnom računu, ova dva pravila su u vezi sa eksplicitnim upravljanjem brojem raspoloživih promenljivih, zato što kontrakcija odgovara dupliranju promenljive, dok slabljenje odgovara brisanju promenljive. Zbog toga se za računski ekvivalent eksplicitnim strukturnim pravilima u logici koristi termin *kontrola resursa*, gde se pod resursima podrazumevaju gradivni sastojci terma, tj. promenljive.

Potreba za kontrolom upotrebe i broja promenljivih u λ-termu se javlja još u λI-računu, koji je predložio Church u [12]. U ovom računu, za razliku od standardnog λ-računa (koji je Church označavao sa λK), promenljiva koja je vezana operatorom λ-apstrakcije bi trebalo da se bar jednom pojavi unutar terma. Stoga, prazna lambda apstrakcija nije prihvatljiva u ovom računu: da bi postojao term $\lambda x.M$ promenljiva x mora da se javi u termu M. Ali, u slučaju da se x ne koristi u M, moguće je primeniti slabljenje (tzv. brisanje) koristeći izraz $x \odot M$, koji jasno implicira da je x dodata termu M, tj. da samo M ne sadrži promenljivu x.

Nalik tome, promenljiva ne bi trebalo da se u termu pojavljuje više od jedanput. Ukoliko, međutim, ipak želimo dva pojavljivanja neke promenljive, potrebno je da je eksplicitno dupliramo. To je moguće uraditi korišćenjem operatora kontrakcije (tzv. dupliranja) $x <^{x_1}_{x_2} M$, koji implicira da se promenljive x_1 i x_2 mogu smatrati jednakim u termu M, te da će obe biti ubuduće označavane sa x. Prateći ove ideje, van Oostrom je predložio proširenje λ-računa [72], dok su Kesner i Lengrand [45] na sličan način proširili $\lambda\mathsf{x}$-račun sa eksplicitnom supstitucijom.

Naše proširenje λ^{Gtz}-računa je bilo inspirisano radom Kesner i Lengranda na $\lambda\mathsf{lxr}$-računu [45], kao i delom Lescannea i Žunića koji su uveli eksplicitnu kontrolu resursa u formalni račun u okruženju klasičnog sekventnog računa [52]. Kao rezultat, nastao je $\lambda_{\circledR}^{\mathsf{Gtz}}$-račun, koji uvodi kontrolu resursa u intuicionističko sekventno okruženje.

Dve bitne razlike između $\lambda\mathsf{lxr}$-računa i $\lambda_{\circledR}^{\mathsf{Gtz}}$-računa su u odgovarajućem logičkom okruženju i u tretmanu supstitucije. Dok $\lambda\mathsf{lxr}$-račun sa osnovnim tipovima odgovara intuicionističkom fragmentu mreža dokaza u linearnoj logici, $\lambda_{\circledR}^{\mathsf{Gtz}}$-račun direktno proširuje λ^{Gtz}-račun, te stoga odgovara intuicionističkom sekventnom računu sa eksplicitnim strukturnim pravilima kontrakcije i slabljenja. Drugo, za razliku od eksplicitne supstitucije u $\lambda\mathsf{lxr}$-računu, supstitucija je u $\lambda_{\circledR}^{\mathsf{Gtz}}$-računu implicitna, tj. izvršava se kao meta-operator. Razlog za ovakav izbor leži u činjenici da je cilj proširenja bilo omogućavanje isključivo eksplicitne kontrole kvantitativnih transformacija promenljivih u termu tj. kontrole resursa, a supstitucija ne spada u tu kategoriju.

Kao što je već sugerisano, $\lambda_{\circledR}^{\mathsf{Gtz}}$-račun predstavlja proširenje λ^{Gtz}-računa eksplicitnim operatorima koji vrše dupliranje i brisanje promenljivih u obe sintaksne kategorije λ^{Gtz}-izraza - u termima t i u kontekstima k. Stoga je sintaksa ovog računa obogaćena novim konstruktima $x <^{x_1}_{x_2} t$, $x <^{x_1}_{x_2} k$, $x \odot t$ i $x \odot k$. Karakteristično za račune sa kontrolom resursa je da izraze računa ne čine svi mogući konstrukti dobijeni kombinovanjem promenljivih i operatora, već je potrebno izdvojiti podskup "pravilnih" izraza. U prisustvu eksplicitne kontrakcije i slabljenja, izraz se smatra pravilnim ako se svaka slobodna promenljiva javlja tačno jednom, i ako svaki vezujući operator vezuje neku slobodnu promenljivu. Dakle, λ^{Gtz}-termi poput $\lambda x.y$ i $x(x :: \widehat{z}.z)$ ne postoje u $\lambda_{\circledR}^{\mathsf{Gtz}}$-računu. Iako na prvi pogled deluje kao da ovi dodatni zahtevi predstavljaju restrikciju u pravom smislu i da postoje λ^{Gtz}-izrazi koji se ne mogu predstaviti u $\lambda_{\circledR}^{\mathsf{Gtz}}$-računu, to nije tačno. Svaki λ^{Gtz}-izraz može preslikati u odgovarajući $\lambda_{\circledR}^{\mathsf{Gtz}}$-izraz, korišćenjem operatora slabljenja i kontrakcije. Na primer, λ^{Gtz}-termu $\lambda x.y$ odgovara $\lambda_{\circledR}^{\mathsf{Gtz}}$-term $\lambda x.x \odot y$, dok λ^{Gtz}-termu $x(x :: \widehat{z}.z)$ odgovara $\lambda_{\circledR}^{\mathsf{Gtz}}$-term $x <^{x_1}_{x_2} x_2(x_1 :: \widehat{z}.z)$.

Kompleksna sintaksa je uslovila i znatno složenija pravila računanja sa $\lambda_{\circledR}^{\mathsf{Gtz}}$-izrazima, koja se razlikuju od pravila za λ^{Gtz}-račun po novim redukcijama koje regulišu ponašanje operatora za kontrolu resursa (Slika 5.2), složenijim meta operatorima supstitucije (Slika 5.4) i spajanja konteksta (Slika 5.3), kao i uvođenjem ekvivalencija (Slika 5.5). Nove redukcije su sistematizovane u tri grupe: (γ) redukcije pomeraju kontrakciju što je moguće dublje u izraz, (ω) redukcije pomeraju slabljenje u suprotnom smeru, ka površini izraza, a $(\gamma\omega)$ redukcije definišu dva moguća načina interakcije različitih operatora za kontrolu resursa. Ovako postavljena pravila redukcije su standardna za račune sa kontrolom resursa, i motivisana optimizacijom računanja, koja nalaže da se kontrakcija vrši što ranije, a slabljenje što kasnije u toku procesa izračunavanja izraza.

Kao i u slučaju λ^{Gtz}-računa, u funkciji ispunjenja sledećeg cilja, a to je karakterizacija jake normalizacije u $\lambda^{\mathsf{Gtz}}_{®}$-računu, u ovaj račun su uvedeni tipovi sa presekom. Predloženi sistem $\lambda^{\mathsf{Gtz}}_{®}\cap$ je prikazan na Slici 5.8.

Operator preseka se nameće kao prirodno rešenje za tipiziranje računa sa eksplicitnom kontrakcijom. To je stoga što je tada omogućeno tipiziranje izraza u kome dve kontrahovane promenljive imaju različite tipove, za razliku od osnovnog tipskog sistema gde se zahteva da one imaju isti tip. Na primer, posmatrajmo term $x <^{x_1}_{x_2} t$. Neka je promenljivoj x_1 dodeljen tip α a promenljivoj x_2 tip β, tada će promenljivoj x koja je nastala kontrakcijom x_1 i x_2 biti dodeljen tip $\alpha \cap \beta$. Budući da je naš presek idempotentan, u slučaju da obe promenljive imaju isti tip, i kontrahovana promenljiva će imati taj tip. Zato je ovaj tipski sistem pravo proširenje osnovnog tipskog sistema, što u slučaju neidempotentnog preseka ne važi.

Sa druge strane, slabljenjem se u izraz uvodi nova promenljiva, koja nema poseban značaj pri izračunavanju tog izraza. Zbog toga se takvim promenljivama dodeljuje posebna tipska konstanta, $\top$, da bi naznačila irelevantnost te promenljive. Ova konstanta je neutralni element za presek, tako da će prilikom kontrakcije, u slučaju da je jedna od kontrahovanih promenljivih uvedena slabljenjem, rezultujuća promenljiva naslediti tip tj. ulogu one druge promenljive.

Svim ostalim promenljivama se u sistemu $\lambda^{\mathsf{Gtz}}_{®}\cap$ dodeljuju striktni tipovi, posebna vrsta tipova sa presekom u kojima je glavni veznik strelica, a presek se javlja samo sa njene leve strane. Ovo je regulisano formulacijom aksiome. Takođe, tipska pravila su tako formulisana da su i svim $\lambda^{\mathsf{Gtz}}_{®}$-izrazima dodeljeni isključivo striktni tipovi.

Dakle, jedna od najvažnijih osobina predloženog tipskog sistema je da on razlikuje tri vrste promenljivih u izrazu na osnovu njihove uloge, i dodeljuje im različite vrste tipova. Ovo svojstvo značajno povećava ekspresivnost sistema, i čini ga pogodnim kandidatom za potencijalnu implementaciju. Osim toga, sistem $\lambda^{\mathsf{Gtz}}_{®}\cap$ se razlikuje od sistema $\lambda^{\mathsf{Gtz}}\cap$ i po načinu zadavanja baza, budući da rad sa eksplicitnim pravilima za kontrolu resursa zahteva multiplikativni stil zadavanja pravila sa dve premise. Uprkos povećanju kompleksnosti sistema, lepe osobine koje su odlikovale sistem $\lambda^{\mathsf{Gtz}}\cap$, poput sintaksne usmerenosti, su u ovom sistemu očuvane. Takođe, dokazano je da sistem $\lambda^{\mathsf{Gtz}}_{®}\cap$ ispunjava osnovnu svrhu sa kojom je konstruisan, tj. da je u njemu moguće dodeliti tip tačno onim $\lambda^{\mathsf{Gtz}}_{®}$-izrazima koji imaju svojstvo jake normalizacije (Teorema 5.70), čime su ispunjeni ciljevi druge faze istraživanja.

Koliko nam je na osnovu dostupne literature poznato, ovo je prvi rad koji kombinuje idempotentan presek i formalne račune sa kontrolom resursa. Drugačiji pristup, motivisan prvenstveno razvojem procesnih računa, je primenio Boudol u [9]. Umesto uvođenja eksplicitnih operatora za kontrolu resursa, Boudol je predložio nedeterministički račun sa uopštenim pojmom aplikacije. U njegovom radu, termi se apliciraju na strukture zvane torba (*bag*), koje sadrže promenljive sa dodeljenim mnogostrukostima, tj. vrednostima koje ukazuju na maksimalni broj mogućih korišćenja te promenljive. Mnogostrukost 1 tako odgovara eksplicitnim strukturnim pravilima, a mnogostrukost ∞ odgovara implicitnim strukturnim pravilima. U ovakav račun su uvedeni neidompotentni[2] tipovi sa presekom od strane Pagani i Ronchi della Rocca [55], koji su korišćeni za karakterizaciju solvabilnih terma, a to su oni termi koji mogu da intereaguju sa okruženjem.

Cilj trećeg i poslednjeg dela istraživanja je bilo povezati i uopštiti prethodno proučene račune i tipske sisteme. Ovaj cilj je ispunjen uvođenjem kocke sa kontrolom resursa (*resource control cube*), strukture koja se sastoji od osam formalnih računa, među kojima su i λ^{Gtz} i $\lambda^{\mathsf{Gtz}}_{®}$. Formalni računi koji formiraju kocku sa kontrolom resursa se međusobno razlikuju po logičkoj osnovi (prirodna dedukcija ili sekventni račun) i po tretmanu svakog od dva operatora za kontrolu resursa (implicitan ili eksplicitan). Računi sa zajedničkom logičkom osnovom čine tzv. bazu kocke, tako da razlikujemo sekventnu (*LJ*-bazu) i prirodno deduktivnu (*ND*-bazu). U okviru svake baze, posmatramo četiri računa, označena sa $\lambda_{\mathcal{R}}$ (odnosno $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$), gde $\mathcal{R} \subseteq \{\mathsf{c}, \mathsf{w}\}$ i c označava prisustvo eksplicitne kontrakcije, dok w označava prisustvo eksplicitnog slabljenja. Ukoliko neki operator za kontrolu resursa nije eksplicitno naveden u imenu računa, podrazumeva se da je u tom računu implicitan.

Računi koji su predmet proučavanja ove disertacije, λ^{Gtz} i $\lambda^{\mathsf{Gtz}}_{®}$, su na taj način povezani u celinu, budući da predstavljaju dva temena *LJ*-baze. λ^{Gtz}-račun je sada obeležen sa $\lambda^{\mathsf{Gtz}}_{\emptyset}$, čime je naglašeno da su u ovom računu strukturna pravila implicitna. Sa druge strane, $\lambda^{\mathsf{Gtz}}_{®}$-račun je obeležen sa $\lambda^{\mathsf{Gtz}}_{\mathsf{cw}}$, zato što su u ovom računu i kontrakcija i slabljenje eksplicitni. Preostala temena sekventnog dela kocke čine dva nova računa sa parcijalnom kontrolom resursa, $\lambda^{\mathsf{Gtz}}_{\mathsf{c}}$ i $\lambda^{\mathsf{Gtz}}_{\mathsf{w}}$, u kojima se jedno strukturno pravilo tretira eksplicitno, a drugo implicitno. Proučavanje ovih računa, koji su iteresantni prvenstveno sa aspekta računskih interpretacija substrukturalnih logika, izlazi van okvira ove disertacje.

Druga, *ND*-baza kocke sa kontrolom resursa se sastoji od četiri odgovarajuća prirodno deduktivna formalna računa koje su predložili Kesner i Renaud u [46]. Kesner i Renaud su uveli sličnu strukturu, prizmoid resursa, sistem od osam for-

[2]kod kojih ne važi da je $\alpha \cap \alpha = \alpha$

malnih računa dobijenih kombinovanjem eksplicitnog i implicitnog tretmana supstitucije, kontrakcije i slabljenja. Oni su na taj način uopštili $\lambda\mathsf{lxr}$-račun, koji se sa sva tri eksplicitna operatora nalazi na vrhu njihovog prizmoida, dok je nasuprot njemu standardni λ-račun na dnu. Iako je prizmoid resursa predstavljao osnovni motiv za uvođenje kocke sa kontrolom resursa, fokus je u našem istraživanju pomeren ka sekventnim formalnim računima, i isključivo ka operatorima za kontrolu resursa, te je zbog toga supstitucija u celoj kocki implicitna.

Takođe, tipski sistemi sa presekom koji su uvedeni u kocku na uniforman način za svaku bazu, predstavljaju originalan rezultat, obzirom da su u prizmoid resursa bili uvedeni samo osnovni tipski sistemi. Sistemi $\lambda_{\mathcal{R}}\cap$ i $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}\cap$, prikazani Slikama 6.14 i 6.16 su zadržali sintaksnu usmerenost sistema $\lambda^{\mathsf{Gtz}}_{®}\cap$ na kom su zasnovani, i svi dodeljuju striktne tipove. Sličnosti i razlike među sistemima, koje su posledica različitog tretmana operatora za kontrolu resursa, su ilustrovane primerom tipiziranja sličnog terma u svakom od osam temena kocke sa kontrolom resursa.

Ovim je kompletirano i završeno istraživanje realizovano okviru ove disertacije, koja je posvećena tipovima sa presekom i kontroli resursa u intuicionističkom sekventnom formalnom računu.

Struktura disertacije

Disertacija je podeljena u 7 glava.

Prve tri glave su preglednog karaktera.

Glava 1 predstavlja uvod u problematiku obrađenu tezom.

U Glavi 2 je prikazan Gentzenov sekventni račun za implikatvni fragment intuicionističke logike, i to oba sistema čije su računske interpretacije predmet ovog istraživanja - sa implicitnim (sistem $G3$) i sa eksplicitnim (sistem $G1$) strukturnim pravilima.

Glava 3 sadrži kratak pregled osnova četiri intuicionistička formalna računa čiji je uticaj na razvoj računa proučavanih u originalnom delu disertacije bio od najvećeg značaja: λ-račun, $\bar{\lambda}$-račun, $\lambda\mathsf{lxr}$-račun i $\lambda_{®}$-račun. Svakom od ovih računa posvećeno je po jedno poglavlje ove glave.

Neizbežna polazna tačka za svako istraživanje u oblasti računskih interpretacija logike je svakako standardni Churchov λ-račun. Budući da je naš interes u ovom radu bio usmeren ka tipovima sa presekom i sekventnom računu, iz obilja rezultata vezanih za λ-račun su odabrana i prikazana dva tipska sistema - sistem $\mathcal{D}$ sa tipovima sa presekom, čiji je autor Krivine [49] i jedan od prvih pokušaja približavanja λ-računa seventnom računu, sistem λLJ, koji su predložili Barendregt i Gilezan [7].

Naredni prikazani formalni račun, koji je narelevantniji za razvoj računskih interpretacija intuicionističkog sekventnog računa, je Herbelinov $\bar{\lambda}$-račun, uveden u [38]. Ovo je prvi intuicionistički formalni račun kojim je obuhvaćena suština sekventnog stila računanja, te je zbog toga poslužio kao inspiracija za sve ostale intuicionističke sekventne formalne račune, pa i za λ^{Gtz}-račun.

U delu istraživanja koje se bavi eksplicitnim tretmanom računske interpretacije strukturnih pravila, time što u terme uvodi operatore za kontrolu resursa, tj. za dupliranje i brisanje promenljivih, osnovnu inspiraciju je pradstavljao $\lambda\mathsf{lxr}$-račun koji su Kesner i Lengrand uveli u radu [45], u kome je konstruisan račun sa eksplicitnom supstitucijom, kontrakcijom i slabljenjem, i ispitane najbitnije osobine osnovnog tipskog sistema za ovaj račun.

Konačno, pojedini tehnički rezultati vezani za jaku normalizaciju sistema $\lambda_{®}^{\mathsf{Gtz}}\cap$ (Poglavlje 5.3) se oslanjaju na analogne rezultate dokazane za $\lambda_{®}\cap$ - $\lambda_{®}$-račun sa tipovima sa presekom. Za razliku od prethodna tri prikazana računa koji su prethodili $\lambda_{®}^{\mathsf{Gtz}}$-računu i uticali na njegovo zasnivanje, ovaj račun je nastao kasnije, kao pomoćni sistem čija je osnovna svrha bila da se kreira jednostavnije okruženje sa kontrolom resursa i tipovi- ma sa presekom, u koje bi potom sistem $\lambda_{®}^{\mathsf{Gtz}}\cap$ bio preveden. Sintaksa i operacijska semantika ovog sistema predstavljaju varijantu λ_{cw}-računa, jednog od konstitutivnih računa Kesner and Renaudovog Prizmoida resursa [46, 47], dok je tipski sistem sa striktnim tipovima uveden u radu Gilezan et al. [32].

Centralni deo ove disertacije, koji sadrži originalne rezultate, predstavljaju Glave 4, 5 i 6.

Glava 4 je u celosti posvećena λ^{Gtz}-računu.

Poglavlje 4.1 prikazuje λ^{Gtz}-račun bez tipova: sintaksu, slobodne promenljive i operacijsku semantiku, uključujući pravila redukcije i dva meta-operato-ra ovog računa. Završava se primerima izračunavanja, i napomenom o odsustvu konfluentnosti.

Tipovi, kako osnovni tako i sa presekom, su uvedeni u λ^{Gtz}-račun u Poglavlju 4.2. Najpre je definisana sintaksa osnovnih tipova, i dve vrste izraza za dodelu tipova, što je karakteristika tipskih sistema za sekventne formalne račune. Posebna pažnja je usmerena ka vezi između pravila tipskog sistema i pravila *LJ* sekventnog računa, te je objašnjena Curry-Howard korespondencija između sistema $\lambda^{\mathsf{Gtz}}\to$ i *LJ*. Zatim je dat primer dodele tipova, i pojašnjena je motivacija za uvođenje tipova sa presekom.

Tipski sistem sa presekom $\lambda^{\mathsf{Gtz}}\cap$ predstavlja prvi originalni doprinos ove disertacije. Definisana je sintaksa tipova sa presekom, potom relacije poretka i ekvivalencije na skupu tipova. Pravila tipskog sistema su uvedena (Slika 4.5), i dokazane su osnovne osobine predloženog tipskog sistema: tvrđenje o pravilu uvođenja preseka sa leve strane (Tvrđenje 4.17), tvrđenja o proširenju baze (Tvrđenje 4.19) i o preseku baza (Tvrđenje 4.21), leme o ponašanju meta-operatora supstitucije (Lema 4.22) i spajanja konteksta (Lema 4.23), i konačno teorema o očuvanju tipa prilikom redukcije tipiziranog λ^{Gtz}-izraza (Subject reduction teorema 4.24). Zatim je naveden primer dodele tipa termu u sistemu $\lambda^{\mathsf{Gtz}}\cap$. Na kraju poglavlja, konstrukcija sistema $\lambda^{\mathsf{Gtz}}\cap$ je dodatno motivisana i objašnjena pomoću analize dva neuspešna pokušaja uvođenja tipova sa presekom u λ^{Gtz}-račun.

Najvažnija osobina koju tipski sistemi sa presekom treba da zadovoljavaju, Karakterizacija jake normalizacije, je za sistem $\lambda^{\mathsf{Gtz}}\cap$ dokazana u Poglavlju 4.3. Prvo je pokazano da svi izrazi kojima se može dodeliti tip u sistemu $\lambda^{\mathsf{Gtz}}\cap$ imaju i osobinu jake normalizacije, tj. da su im sve redukcije konačne (Teorema 4.41). Dokaz ove teoreme koristi utapanje λ^{Gtz}-izraza u λ-terme za koje je dokazano da očuvava tipove (Tvrđenje 4.35), i poznati rezultat o jakoj normalizaciji λ-terma koji imaju tip u sistemu sa presekom $\mathcal{D}$ (Teorema 3.7). Potom je dokazan drugi smer tvrđenja, da svi λ^{Gtz}-izrazi koji imaju osobinu jake normalizacije imaju i tip u sistemu $\lambda^{\mathsf{Gtz}}\cap$ (Teorema 4.47). Dokaz koristi dva pomoćna tvrđenja: (i) λ^{Gtz} normalne forme imaju tip (Tvrđenje 4.42); (ii) proces ekspanzije[3] λ^{Gtz}-izraza očuvava tip, ukoliko se podrazumeva da se ekspanzija odigrava na površini terma (Tvrđenje 4.46). Konačno, potpuna karakterizacija jako normalizovanih λ^{Gtz}-izraza u sistemu $\lambda^{\mathsf{Gtz}}\cap$ je direktna posledica upravo dokazanih Teorema 4.41 i 4.47.

Poslednje poglavlje ove glave, Poglavlje 4.4, usmerava pažnju nazad ka λ^{Gtz}-računu bez tipova, i ispravljanju već utvrđene nepoželjne osobine ovog računa, odsustva konfluentnosti (tzv. Church-Rosserove osobine). Predloženi način za postizanje konfluentnosti se bazira na restrikciji jednog od dva pravila redukcije koja formiraju kritični par redukcija, koji je izvor ne-konfluentnosti. Na taj način su dobijena dva striktna pod-računa, pod nazivom λ_V^{Gtz} i λ_L^{Gtz}. Na kraju je formalno dokazano da dobijeni računi zaista zadovoljavaju Church-Rosserovu osobinu korišćenjem modifikacije Takahashijeve tehnike paralelnih redukcija koju je predložila Likavec.

Originalni doprinosi predstavljeni u Glavi 4 su ostvareni u saradnji sa José Espírito Santom, Silviom Gilezan i Silviom Likavec, i objavljeni u radovima [24, 31, 41, 42, 25].

U Glavi 5, osnovni predmet proučavanja je $\lambda_{®}^{\mathsf{Gtz}}$-račun (*resource control lambda Gentzen calculus*), intuicionistički sekventni formalni račun sa kontrolom resursa, tj. sa eksplicitnom kontrakcijom (dupliranjem) i slabljenjem (brisanjem) promenljivih.

U Poglavlju 5.1 je uveden $\lambda_{®}^{\mathsf{Gtz}}$-račun bez tipova. Prvo su definisani pre-izrazi $\lambda_{®}^{\mathsf{Gtz}}$-računa, tako što je sintaksa λ_V^{Gtz}-računa proširena operatorima koji vrše kontrakciju i slabljenje promenljivih u termima i kontekstima. Potom je iz skupa pre-izraza izdvojen podskup koji sadrži $\lambda_{®}^{\mathsf{Gtz}}$-izraze, putem inferentnih pravila sadržanih u Slici 5.1. Ovaj skup sadrži samo "pravilne" pre-izraze (u prisustvu eksplicitne kontrakcije i slabljenja, izraz se smatra pravilnim ako se svaka slobodna promenljiva javlja tačno jednom, i ako svaki vezujući operator vezuje neku slobodnu promenljivu). Pokazano je da ovi dodatni zahtevi ne predstavljaju restrikciju u pravom smislu, budući da se svaki λ^{Gtz}-izraz može preslikati u odgovarajući $\lambda_{®}^{\mathsf{Gtz}}$-izraz.

Nakon što su definisani izrazi, uvedena je i operacijska semantika. Pravila računanja uključuju četiri grupe pravila redukcije, zatim pravila ekvivalencije, kao i proširene definicije dva meta-operatora iz λ^{Gtz}-računa. Uveden je i pojam paralelne supstitucije, koji je omogućen jasnom distribucijom slobodnih promenljivih u $\lambda_{®}^{\mathsf{Gtz}}$-izrazu. Na kraju ovog poglavlja, dokazano je da su slobodne promenljive očuvane prilikom izračunavanja, tj. tokom redukcija i ekvivalencija (Tvrđenje 5.12).

Poglavlje 5.2 je posvećeno $\lambda_{®}^{\mathsf{Gtz}}$-računu sa osnovnim tipovima. Tipski sistem $\lambda_{®}^{\mathsf{Gtz}} \to$ je dobijen prirodnim proširenjem sistema $\lambda^{\mathsf{Gtz}} \to$ sa četiri nova pravila koja

[3]Ekspanzija je proces obrnutog smera od redukcije.

dodeljuju tipove novo-uvedenim sintaksnim konstruktima. Pravila sistema $\lambda_{®}^{\mathsf{Gtz}} \rightarrow$ (Slika 5.6) su prezentovana u multiplikativnom stilu, za razliku od aditivnog stila zastupljenog u sistemu $\lambda^{\mathsf{Gtz}} \rightarrow$, što je u skladu sa prelaskom sa implicitna na eksplicitna strukturna pravila. Formulisana su osnovna svojstva predloženog sistema, i dokazana je jaka normalizacija sistema $\lambda_{®}^{\mathsf{Gtz}} \rightarrow$ (Teorema 5.42). Dokaz ove teoreme se zasniva na interpretaciji $\lambda_{®}^{\mathsf{Gtz}}$-računa u $\lambda\mathsf{lxr}$-račun sa osnovnim tipovima, za koji su Kesner i Lengrand dokazali jaku normalizaciju.

U Poglavlju 5.3, pažnja je usmerena ka tipovima sa presekom u okruženju sa kontrolom resursa. Uvedena je posebna forma tipova sa presekom, tzv. striktni tipovi, koji će biti dodeljeni $\lambda_{®}^{\mathsf{Gtz}}$-izrazima, kao i posebna tipska konstanta $\top$ (neutralni element za presek) koja će biti dodeljena isključivo onim promenljivama koje su uvedene slabljenjem. Predloženi tipski sistem sa presekom $\lambda_{®}^{\mathsf{Gtz}}\cap$, prikazan Slikom 5.8, je sintaksno usmeren, prezentovan u multiplikativnom stilu, i dodeljuje isključivo striktne tipove $\lambda_{®}^{\mathsf{Gtz}}$-izrazima. Poseban značaj ovog sistema leži u činjenici da se na osnovu vrste tipa dodeljenog promenljivoj može prepoznati koju od tri uloge ta promenljiva vrši u izrazu. Dokazana su i osnovna tvrđenja vezana za ovaj sistem: korespondencija domena (Tvrđenje 5.45), lema o generisanju (Lema 5.46), leme o tipiziranju zamene i spojenih konteksta (Leme 5.47 i 5.48) i konačno očuvanje tipa prilikom izračunavanja (Tvrđenje 5.50).

Najvažniji rezultat u vezi sistema $\lambda_{®}^{\mathsf{Gtz}}\cap$ - teorema o karakterizaciji jake normalizacije u $\lambda_{®}^{\mathsf{Gtz}}$-računu sa tipskim sistemom $\lambda_{®}^{\mathsf{Gtz}}\cap$ (Teorema 5.70) je dokazana u nastavku ovog poglavlja. Dokaz smera tvrđenja da svi termi sa tipom imaju svojstvo jake normalizacije je zasnovan na interpretaciji $\lambda_{®}^{\mathsf{Gtz}}$-računa u $\lambda_{®}$-račun, za koji je dokazano da zadovoljava jaku normalizaciju u tipskom sistemu sa presekom (Teorema 3.12). Obrnuti smer tvrđenja, u kome se tvrdi da jaka normalizacija implicira tipiziranost u sistemu $\lambda_{®}^{\mathsf{Gtz}}\cap$, je dokazan prateći postupak analogan dokazu ekvivalentnog tvrđenja u λ^{Gtz}-računu.

Formalni račun sa kontrolom resursa predstavljen u Glavi 5 i njegovi tipski sistemi, sa osnovnim i striktnim tipovima, uključujući i sve dokazane osobine, predstavljaju originalan doprinos ove disertacije. Oni su nastali u saradnji sa Pierreom Lescanneom, Silviom Gilezan, Silviom Likavec i Dragišom Žunićem, i objavljeni su u radovima [35, 32].

U Glavi 6 su generalizovani formalni računi i tipski sistemi uvedeni u prethodne dve glave. Predložena je kocka sa kontrolom resursa (*resource control cube*), struktura koja se sastoji od osam intuicionističkih formalnih računa sa implicitnim ili eksplicitnim strukturnim pravilima, i sa prirodnom dedukcijom ili sekventnim računom kao logičkom osnovom. Dve strane ove strukture zasnovane na različitim logikama predstavljaju *ND*-bazu i *LJ*-bazu kocke.

U Poglavlju 6.1 je predstavljena kocka bez tipova - njena sintaksa i operacijska semantika. Na početku su definisani pre-termi *ND*-baze (odnosno pre-izrazi, u slučaju *LJ*-baze) i slobodne promenljive, da bi potom inferentnim pravilima bili definisani termi, odnosno izrazi. Zatim je uvedena operacijska semantika: pravila redukcije, ekvivalencije i definicije meta-operatora. Na kraju, preciziranjem logičke osnove i elemenata skupa $\mathcal{R}$ specificirano je svih osam računa sadržanih u kocki sa kontrolom resursa.

Poglavlje 6.2 sumira tipske sisteme sa osnovnim tipovima, koji su već uvedeni za većinu računa kocke (tačnije, za sve osim dva nova računa *LJ* baze sa parcijalnom kontrolom resursa - $\lambda_{\mathsf{c}}^{\mathsf{Gtz}}$-račun i $\lambda_{\mathsf{w}}^{\mathsf{Gtz}}$-račun). Tipski sistemi za svaku bazu kocke, $\lambda_{\mathcal{R}} \rightarrow$ i $\lambda_{\mathcal{R}}^{\mathsf{Gtz}} \rightarrow$, su prikazani na uniforman način.

U Poglavlju 6.3, su u kocku uvedeni tipski sistemi sa presekom, $\lambda_{\mathcal{R}}\cap$ i $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}\cap$. Ovi sistemi su zasnovani na sistemu $\lambda_{®}^{\mathsf{Gtz}}\cap$, uvedenom u delu 5.2.2 ove disertacije, koji je ugrađen u kocku pod nazivom $\lambda_{\mathsf{cw}}^{\mathsf{Gtz}}\cap$. Svi sistemi su sintaksno usmereni, i svi dodeljuju striktne tipove. Poglavlje se završava primerima tipiziranja u svakom od osam tipskih sistema sa presekom.

Sva tri poglavlja ove glave su podeljena u dve celine - jednu koja predstavlja prirodno deduktivnu stranu kocke, i drugu koja se bavi kockinom sekventnom stranom.

Konačno, Glava 7 je poslednja i zaključna glava ove disertacije. U njoj su rezimirani originalni rezultati izneti u prethodnim glavama, i diskutovane kako mogućnosti primene ovih rezultata, tako i mogući nastavci istraživanja u oblasti obuhvaćenoj ovim istraživanjem.

Abstract

The subject of the research presented in this thesis is computational interpretations of the intuitionistic sequent calculus with implicit and explicit structural rules, with focus on the systems with intersection types.

The contributions of the thesis are grouped into three parts.

The first part introduces intersection types into the λ^{Gtz}-*calculus*, a calculus that captures the computational content of intuitionistic sequent calculus with implicit structural rules. The proposed type assignment system, denoted $\lambda^{\mathsf{Gtz}}\cap$, differs from the standard type systems with intersection types in the management of the introduced intersection operator. The intersection is here integrated into already existing rules of the simply typed system, hence the number of type assignment rules remains unchanged, resulting in the real syntax directness of the system which made the most of the related propositions easier to prove. The main result of this part is the proof that the system $\lambda^{\mathsf{Gtz}}\cap$ characterises the set of strongly normalising λ^{Gtz}-expressions.

The second part represents an extension of the λ^{Gtz}-calculus to a term calculus with *resource control*, denoted $\lambda_{®}^{\mathsf{Gtz}}$, which computationally corresponds to the intuitionistic sequent calculus with explicit structural rules. The main novelty with respect to the λ^{Gtz}-calculus are operators that perform duplication and erasure of variables, and correspond to the structural rules of contraction and weakening, respectively. In order to characterise all strongly normalising $\lambda_{®}^{\mathsf{Gtz}}$-expressions, the intersection types were introduced. Intersection naturally works together with contraction, because it enables the contraction of two variables typed with, say types α and β, into one variable typed with both α and β, i.e. $\alpha \cap \beta$. On the other hand, since weakening introduces a new variable, which is not significant for the rest of the computation, a specific type constant is assigned to that variable to mark its irrelevance. Therefore, in the system $\lambda_{®}^{\mathsf{Gtz}}\cap$, we distinguish three types of variables according to their role in an expression and each one receives a particular kind of type. Desired properties of the system $\lambda_{®}^{\mathsf{Gtz}}\cap$ are proved.

The third and final part of the research was unification and generalisation of the previously obtained results. This goal was achieved by introducing the *resource control cube*, a system consisting of eight formal calculi. The calculi of the resource control cube differ in logical framework (natural deduction/sequent calculus) and in the treatment of resource control (implicit/explicit). In that way, both calculi studied in this thesis - λ^{Gtz} and $\lambda^{\mathsf{Gtz}}_{®}$ were integrated into one framework, as well as their versions with intersection types.

Contents

Chapter 1

Introduction

Rules of reasoning were known and some of them even formally described since ancient times. The earliest known document on the subject was the Organon, a collection of Aristotle's works on logic. Logic as a mathematical discipline dates back to the second half of the nineteenth century when Gottlob Frege introduced the basic concepts of modern logic, extending and developing the ideas of Gottfried Leibniz. Logic experienced its intensive development in the early twentieth century, and some of the most significant logicians of that time were Gerhard Gentzen, David Hilbert, Bertrand Russel and especially Kurt Gödel, whose incompleteness theorems showed the limitations of formal arithmetical systems. During that period three fundamental formal logical proof systems were introduced - Hilbert's axiomatic system and two Gentzen's systems, namely natural deduction and sequent calculus.

The third decade of the twentieth century was the starting point for the development of theoretical computer science, which since then continued to develop simultaneously with logic and interact with it on many levels. Various formal calculi were proposed, following the efforts of the mathematicians of that time directed towards formalisation of mathematics (which turned out to be unsuccessful, as proved by Gödel). During that period, combinatorial calculus created by Moses Schönfinkel and Haskell Curry appeared, followed by Alonso Church's λ-calculus, introduced in [10]. In 1940 Church introduced simple types to λ-calculus [11], and this system has later become the main formalism underlying functional programming, starting from Peter Landin's work on Algol 60 [50].

The correspondence between logic and λ-calculus was initially observed by Curry in 1934, who showed that there is an isomorphism between the provable formulae of intuitionistic implicational logic and the type expressions assignable to closed terms. The correspondence was extended by Howard in 1969. According to the Curry-Howard correspondence, eventually published in [40], natural deduction for intuitionistic logic and λ-calculus with simple types are mutually exchangeable systems, meaning that proofs of formulae correspond to the program specifications, while proof simplifications correspond to the program executions. This paradigm, the core of computation, even strengthened the importance of logic in computer science, since proof theory became one of the main tools for program analysis.

Naturally, the problem of extending the Curry-Howard correspondence and creating the computational interpretations of other logical systems appeared. While the correspondence between combinators and axiomatic system was obvious, the situation with the sequent calculus was rather different. Already Gentzen proved that the systems of natural deduction and sequent calculus are equipotent, meaning that any proof written in the style of natural deduction can be translated into the proof written in the sequent calculus style and vice versa. Zucker [74] and Pottinger [56] showed that the proof simplifications in the two systems, namely normalisation in natural deduction and cut-elimination in sequent calculus are corresponding, more precisely that the former is a homomorphic image of the latter. Therefore, it became clear that the Curry-Howard correspondence could be extended to the sequent calculus and a variant of λ-calculus, contrary to the previous views on the topic[1].

However, since the sequent calculus and λ-calculus significantly differ in structure, modifications had to be done on the syntactic level, not only in the form of changed typing rules. The first formal calculus that in the sense of Curry-Howard corresponded to the intuitionistic sequent calculus was $\bar{\lambda}$-calculus, proposed by Hugo Herbelin in 1995 [38]. In this paper, the isomorphism was obtained between the set of terms of $\bar{\lambda}$-calculus and the set of proofs of the restricted sequent system LJT. Following the $\bar{\lambda}$-calculus, several other formal calculi based on the sequent calculus were proposed. One of them is λ^{Gtz}-calculus, designed by José Espírito Santo in 2007 [27]. The significance of the λ^{Gtz}-calculus lies in the fact that its simply typed system corresponds to the full (non-restricted) intuitionistic sequent calculus *LJ*.

The main subject of research of this thesis are exactly computational interpretations of the intuitionistic sequent calculus. The starting point of the research was Espírito Santo's λ^{Gtz}-calculus, as a neat and simple system that fully captures computational content of full Gentzen's *LJ* system. The focus of this initial part of the research was on the typed λ^{Gtz}-calculus. A new type assignment system was

[1] "...the sequent calculus has no Curry-Howard isomorphism, because of the multitude of ways of writing the same proof. This prevents us from using it as a typed λ-calculus, although we glimpse some deep structure of this kind..." Jean-Yves Girard, [37]

developed in order to capture the set of terminating λ^{Gtz}-expressions by means of typeability. For that purpose, the intersection type operator was introduced.

Intersection types were introduced to λ-calculus in the late seventies, independently by Coppo and Dezani [13], Sallé [63] and Pottinger [57]. The basic idea is that a term could have more than one type assigned to it, and that in that case an intersection of these types is also a type assigned to the term. That simple and natural idea turned to be successful in overcoming some limitations of the simple type discipline. Particularly, it enabled the characterisation of all strongly normalising λ-terms, while simple types failed even in characterisation of normal forms. The most well-known and first semantically complete type assignment system with intersection types was proposed by Barendregt, Coppo and Dezani [6]. Its types were built from type variables and the constant ω (representing universal type) by means of constructors $\rightarrow$ and $\cap$, and it contained a pre-order relation $\leq$ on types. Intersection types turned out to be a powerful tool for studying the behavioural properties of various systems, more precisely for the analysis and the synthesis of λ-models. They also found application in programming, for example in static analysis of functional programs. More recently, program synthesis finds its theoretical foundations in inhabitation.

The intersection type assignment system for the λ^{Gtz}-calculus proposed in this thesis differs from the standard type systems with intersection types in the management of the introduced intersection operator. In our system $\lambda^{\mathsf{Gtz}}\cap$, intersection is integrated into already existing rules of the simply typed system $\lambda^{\mathsf{Gtz}}\rightarrow$, hence the number of type assignment rules remains unchanged, resulting in the real syntax directness of the system, which made the most of the related propositions easier to prove. The other difference with respect to the system from [6] is the lack of the universal type ω. We proved that the proposed system $\lambda^{\mathsf{Gtz}}\cap$ characterises the set of strongly normalising λ^{Gtz}-expressions, thus accomplishing the goal set for this part of the research.

The second step was to extend the λ^{Gtz}-calculus to a term calculus that would correspond in the Curry-Howard way to another version of the sequent calculus, also proposed by Gentzen [29], namely the sequent calculus with explicit structural rules.

While the logical rules of the sequent calculus introduce the logical connectives to the right hand side (respectively left hand side) of the sequents, structural rules on the other hand (namely contraction, weakening and exchange), operate directly on sequents. In the intuitionistic sequent calculus, they produce changes only on the left hand side of the sequents, where they perform erasure of the multiple occurrences of the same formula (contraction), addition of an arbitrary formula irrelevant for the conclusion (weakening) and permutation of the order of the formulas (exchange). In a formal calculus, the first two rules are related to the control of available resources (i.e. term variables), since contraction corresponds to the duplication of a variable, whereas weakening corresponds to the erasure.

The need to control the use of variables in a λ-term can be traced back to λI-calculus, proposed by Church in [12]. In this calculus, contrary to the standard λ-calculus (Church denoted it by λK) the variables bound by λ-abstraction should occur in the body of the abstraction at least once. Therefore, a void lambda abstraction is not acceptable, so in order to have $\lambda x.M$ the variable x has to occur in M. But if x is not used in M, one can perform an erasure (a.k.a weakening) by using the expression $x \odot M$, which implies that the term M does not contain the variable x. Similarly, a variable should not occur twice. If nevertheless, we want to have two positions for the same variable, we have to explicitly duplicate it. This could be done by using the operator $x <^{x_1}_{x_2} M$, called duplication (a.k.a contraction) which creates two fresh variables x_1 and x_2. Following these ideas, van Oostrom proposed to extend the λ-calculus [72], and Kesner and Lengrand [45] proposed to extend the λx-calculus with operators that control the use of variables (resources).

Our work was inspired by Kesner and Lengrand's work on the λlxr-calculus [45], as well as by Lescanne and Žunić's work on the calculi with explicit resource control in the classical setting [52]. The result is the $\lambda^{\mathsf{Gtz}}_{®}$-calculus, which introduces resource control into the intuitionistic sequent setting. This calculus represents an extension of the λ^{Gtz}-calculus with explicit operators for performing duplication and erasure on both syntactic categories of λ^{Gtz}-expressions, namely terms and contexts. The two main differences between the λlxr-calculus and the $\lambda^{\mathsf{Gtz}}_{®}$-calculus are in the corresponding logical framework and in the treatment of substitution. The simply typed version of the λlxr-calculus corresponds to the intuitionistic fragment of Linear Logic proof-nets, while the $\lambda^{\mathsf{Gtz}}_{®}$-calculus directly extends the λ^{Gtz}-calculus, hence it corresponds to the intuitionistic sequent calculus with explicit structural rules. The substitution in the $\lambda^{\mathsf{Gtz}}_{®}$-calculus is performed implicitly, i.e. as a meta-operator, since the goal of the extension was to enable the explicit resource control.

Just as in the case of the λ^{Gtz}-calculus, in order to fulfill our next goal, namely the characterisation of all strongly normalising $\lambda^{\mathsf{Gtz}}_{®}$-expressions, we introduced intersection types to the $\lambda^{\mathsf{Gtz}}_{®}$-calculus. Intersection naturally works together with contraction, because it enables the contraction of two variables typed with, say types α and β, into one variable typed with both α and β, i.e. $\alpha \cap \beta$. Our intersection is idempotent, so if both contracted variables are of the same type, the resulting variable is of that type, as well. On the other hand, since weakening introduces a new variable, which is not significant for the rest of the computation, a specific type constant is assigned to that variable to mark its irrelevance. Therefore, in typed $\lambda_{\circledR}^{\mathsf{Gtz}}$-calculus, we distinguish three types of variables according to their role in an expression. The novelty of the proposed system with respect to the λ^{Gtz}-calculus with intersection types is the treatment of bases, since resource-awareness demands context-splitting style of the rules. However, nice properties of the system $\lambda^{\mathsf{Gtz}}\cap$, like syntax-directness, are preserved.

To the best of our knowledge, this is the first work that combines idempotent intersection types and resource aware λ-calculi. A different approach to the λ-calculus with resource control was investigated by Boudol in [9]. Instead of introducing the explicit resource operators, Boudol proposed a non-deterministic calculus with a generalised notion of application. In his work, a function is applied to a bag, a structure that contains variables and their multiplicities, i.e. values representing the maximum possible number of the variable usage. This approach was motivated mostly by the development of process calculi. Non-idempotent intersection types were introduced to this calculus by Pagani and Ronchi della Rocca in [55], and used for the characterisation of solvable terms, the ones that can interact with an environment.

The third and final part of the research was unification and generalisation of all the obtained results. This goal was achieved by introducing the resource control cube, a system consisting of eight formal calculi. The calculi of the resource control cube differ in logical framework (natural deduction/sequent calculus) and in the treatment of resource control (implicit/explicit). In that way, both calculi studied in this thesis - λ^{Gtz} and $\lambda_{\circledR}^{\mathsf{Gtz}}$ are integrated into one framework, since they represent two vertices of the so-called "*LJ* base" of the cube. Remaining vertices of the sequent base are two new calculi with partial resource control, meaning that one of the resource control operators (contraction or weakening) is explicit, while the other one is implicit. The other base of the resource control cube, the so-called "*ND* base", consists of four natural deduction counterpart calculi, proposed by Kesner and Renaud in [46]. Kesner and Renaud introduced the prismoid of resources, a system of eight calculi obtained by combining explicit/implicit treatment of substitution, contraction and weakening, thus generalising the $\lambda\mathsf{lxr}$-calculus. Although their work on prismoid of resources was the major motivation for creating the resource control cube, the focus of the later system is moved to the sequent-style calculi, and exactly on the control of duplication and erasure of resources, leaving the substitution implicit. Moreover, the intersection type assignment system, given in a uniform way for each basis, is a novel result for all eight calculi of the cube, since only simple types were introduced to the prismoid of resources. This completes and concludes the description of the research on the subject of the thesis - intersection types and resource control in the intuitionistic sequent formal calculi.

Structure of the thesis

The thesis is divided into seven chapters.

The first three chapters are introductory: Chapter 1 is an introduction, Chapter 2 revisits Gentzen's sequent calculus systems for the implicative fragment of intuitionistic logic, whereas the most relevant term calculi for the main part of the thesis are briefly revisited in Chapter 3.

The central part of the thesis, containing its contributions, is presented in Chapters 4, 5 and 6.

Chapter 4 is devoted to the λ^{Gtz}-calculus.

Section 4.1 presents the untyped λ^{Gtz}: its syntax, free variables and operational semantics, including reduction rules and meta-operators. It ends with two examples of computation in the λ^{Gtz}-calculus where the lack of confluence is showed and discussed.

Types, both simple and intersection, are introduced to the λ^{Gtz}-calculus in Section 4.2. Arrow types are defined and two kinds of type assignments are given. Particular focus is on the connection of simple types with the rules of *LJ* sequent calculus. The Curry-Howard correspondence between $\lambda^{\mathsf{Gtz}} \rightarrow$ and *LJ* is established. The section ends with an example of a typing and the motivation for the introduction of intersection types. The intersection type assignment system $\lambda^{\mathsf{Gtz}}\cap$ is the first original contribution of the thesis. Intersection types are defined, followed by the definitions of a pre-order and equivalence relations on the set of types. The rules of the system $\lambda^{\mathsf{Gtz}}\cap$ are introduced and some basic properties of the system proved: an admissible rule for the left intersection introduction, basis expansion, bases intersection, generation lemma, substitution lemma, append lemma and finally preservation of types under reductions (Subject reduction). An example of typing with the rules of the system $\lambda^{\mathsf{Gtz}}\cap$ is given. The system $\lambda^{\mathsf{Gtz}}\cap$ is further motivated and explained by means of an analysis of two unsuccessful attempts of introducing intersection types into the λ^{Gtz}-calculus.

The most important feature of the intersection type systems, i.e. the characterisation of strong normalisation, is proved for the system $\lambda^{\mathsf{Gtz}}\cap$ in Section 4.3. We prove that all expressions typeable in the system $\lambda^{\mathsf{Gtz}}\cap$ are strongly normalising with respect to the reduction rules of λ^{Gtz}-calculus. The proof relies on the embedding of λ^{Gtz}-expressions into λ-terms which preserves types and the result of strong normalisation for λ-terms typeable with the intersection system $\mathcal{D}$. The proof that all strongly normalisable λ^{Gtz}-expressions are typeable in the $\lambda^{\mathsf{Gtz}}\cap$ uses two auxiliary propositions: (i) normal forms are typeable; (ii) types are preserved during expansion (provided that the expansion happened on the surface of an expression, what we call "head expansion" or "expansion at the root position").

The last section of this chapter, Section 4.4, brings attention back to the untyped λ^{Gtz}-calculus and proposes a way to fix the problem of non-confluence, by restricting one of the two reduction rules forming the critical pair. In that way, two sub-calculi are obtained, namely λ_V^{Gtz} and λ_L^{Gtz} and the proof of the Church-Rosser property which uses Likavec's modification of Takahashi's parallel reductions technique is given.

The contributions presented in the Chapter 4 are obtained in collaboration with José Espírito Santo, Silvia Ghilezan and Silvia Likavec, and are published in [24, 31, 41, 42, 25].

The main subject of Chapter 5 is the $\lambda_{®}^{\mathsf{Gtz}}$-calculus, an intuitionistic sequent term calculus with contraction (duplication) and weakening (erasure) operators. Section 5.1 introduces untyped $\lambda_{®}^{\mathsf{Gtz}}$-calculus. Pre-expressions of the $\lambda_{®}^{\mathsf{Gtz}}$-calculus are defined by extending the syntax of the λ_V^{Gtz}-calculus with explicit operators that perform contraction and weakening in both terms and contexts. $\lambda_{®}^{\mathsf{Gtz}}$-expressions are defined by the inference rules, as a subset of well-formed pre-expressions (we say that a pre-expression is well-formed if in every sub-expression every free variable occurs exactly once, and every binder binds exactly one occurrence of a free variable). The operational semantics is then introduced, including four groups of reduction rules, equivalencies, and extended notions of two meta-operators from λ^{Gtz}-calculus. A notion of parallel substitution is also defined, which is enabled by the clear distribution of free variables in a $\lambda_{®}^{\mathsf{Gtz}}$-expression. At the end of this section, we prove that equivalencies and reductions preserve the set of free variables of an expression.

Section 5.2 is devoted to the simply typed $\lambda_{®}^{\mathsf{Gtz}}$-calculus. The system $\lambda_{®}^{\mathsf{Gtz}} \to$ is obtained by extending the system $\lambda^{\mathsf{Gtz}} \to$ with four new rules for assigning types to the newly introduced resource operators. The rules of the $\lambda_{®}^{\mathsf{Gtz}} \to$ are presented in the context-splitting (multiplicative) style, unlike the context-sharing (additive) style of $\lambda^{\mathsf{Gtz}} \to$, due to the explicit structural rules. Some basic features of the proposed system, such as domain correspondence, subject reduction and equivalence, are stated, and the strong normalisation of the system $\lambda_{®}^{\mathsf{Gtz}} \to$ is proved. The proof is based on the interpretation of the $\lambda_{®}^{\mathsf{Gtz}}$-calculus into the $\lambda\mathsf{lxr}$-calculus of Kesner and Lengrand, which is proved to be strongly normalising.

In Section 5.3 we turn our attention to intersection types for resource control sequent calculus. We define a restricted form of intersection types, namely strict types, that will be assigned to $\lambda_{®}^{\mathsf{Gtz}}$-expressions, and a specific type constant $\top$ (neutral element for intersection) which will be used to type variables introduced by the weakening operator. We propose $\lambda_{®}^{\mathsf{Gtz}}\cap$, syntax-directed and presented in context-splitting style system that assigns strict types to $\lambda_{®}^{\mathsf{Gtz}}$-expressions. We give a detailed analysis of the type assignment rules, with particular attention to three different roles of variables in $\lambda^{\mathsf{Gtz}}_{®}$-calculus and their corresponding types. We prove some basic features of the system $\lambda^{\mathsf{Gtz}}_{®}\cap$: domain correspondence, generation lemma, substitution and append lemma for typing meta-operators and finally preservation of types during computation.

The most important result regarding the system $\lambda^{\mathsf{Gtz}}_{®}\cap$ - a theorem stating that a $\lambda^{\mathsf{Gtz}}_{®}$-term is strongly normalising if and only if it is typeable in the system $\lambda^{\mathsf{Gtz}}_{®}\cap$ is further proved. First, we prove that all typeable terms are terminating. The proof is based on an interpretation of the $\lambda^{\mathsf{Gtz}}_{®}$-calculus into the $\lambda_{®}$-calculus (which is proved to be strongly normalising if typeable with intersection types). Then, we prove the opposite direction of the SN characterisation theorem, i.e. we show that all strongly normalising terms are typeable in the system $\lambda^{\mathsf{Gtz}}_{®}\cap$. Here, we follow the procedure already used for accomplishing the same task in the λ^{Gtz}-calculus.

The calculus presented in Chapter 5 and corresponding type systems, both with simple and intersection types, including all its properties, represent the original results of the thesis. Contributions are the result of the joint work with Pierre Lescanne, Silvia Ghilezan, Silvia Likavec and Dragiša Žunić and are published in [35, 32].

In Chapter 6, we generalise the systems proposed in the previous two chapters, by proposing the resource control cube, a system consisting of eight intuitionistic lambda calculi with either implicit or explicit control of resources and either natural deduction or sequent calculus. Two sides with different underlying logics are referred to as *ND*-base and *LJ*-base of the cube. Within each base, we consider four $\lambda_{\mathcal{R}}$ (respectively $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$), where $\mathcal{R} \subseteq \{\mathsf{c},\mathsf{w}\}$ and c denotes explicit contraction whereas w denotes explicit weakening. The operator which is not denoted by index in the name of the calculi is assumed to be implicit in it.

In Section 6.1 we present the untyped cube - its syntax and operational semantics. We start by defining the notions of pre-terms (pre-expressions) and free variables, followed by definition of terms (expressions). We move then to the operational semantics: reduction and equivalence rules and the definition of implicit substitution. Finally, by instantiating the set $\mathcal{R}$ we specify each of the eight calculi of the cube.

Section 6.2 revisits and summarizes basic type assignment systems with simple types, that are already known for the most of the calculi of the resource control cube (for all but the new calculi $\lambda^{\mathsf{Gtz}}_{\mathsf{c}}$ and $\lambda^{\mathsf{Gtz}}_{\mathsf{w}}$). Systems of each base, $\lambda_{\mathcal{R}}\to$ and $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}\to$, are presented in the uniform way.

In Section 6.3, we introduce intersection type assignment systems, $\lambda_{\mathcal{R}}\cap$ and $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}\cap$. These systems are based on the system $\lambda^{\mathsf{Gtz}}_{®}\cap$, introduced in Subsection

5.2.2, which is incorporated here as the system $\lambda_{\mathsf{cw}}^{\mathsf{Gtz}}\cap$. All systems are syntax-directed and assign strict types to resource control expressions. We conclude this section with examples of type assignments in all eight systems with intersection types.

All three sections of this chapter are divided into two subsections - one dealing with the natural deduction part of the resource control cube, and other dealing with the cube's sequent side.

Finally, in Chapter 7 we recapitulate the work done in the previous chapters with the emphasize on the thesis' contributions, discuss possible fields of application as well as possible directions for future work and conclude.

Chapter 2

Sequent calculus

The sequent calculus has been introduced by Gerhard Gentzen [29] as a formalism more suitable for the proof search than natural deduction. Gentzen introduced the sequent calculi for both intuitionistic and classical logic, denoted by *LJ* and *LK*, respectively. We will now recall some basic concepts of the sequent calculus *LJ*, the system for intuitionistic logic.

The central notion in the sequent calculus is that of a *sequent*. A sequent in *LK* is a formal expression of the form

$$\Gamma \vdash \Delta$$

where Γ and Δ are the sets (or multisets or lists) of formulae, called contexts. The formulae on the left hand side of the turnstyle are called *antecedents* and the ones on the right-hand side of the turnstyle are called *succedents*. The *LJ* system is the restriction of the *LK*, in that succedent can be only one formula. The intuitionistic sequents are therefore written in the form

$$A_1, A_2, ..., A_n \vdash B, \quad n \geq 0$$

where A_i, $i \in \{1, 2, ..., n\}$ and B are logical formulae. The formulae in the general case belong to the first-order predicate logic, but we will here focus only on the implicative fragment of the propositional intuitionistic logic. Therefore, the formulae are made of atomic formulae p belonging to a denumerable set of letters and the binary implication operator $\rightarrow$:

$$A, B \ ::= \ p \mid A \rightarrow B.$$

Inference rules consist of sequents forming *premises* (above the horizontal line) and a *conclusion* (below the line). There can be zero, one, or two premises and there is always a single conclusion. There are three kinds of inference rules: *logical*, *structural* and *a cut-rule*. Logical rules introduce logical connectives and quantifiers to formulae, structural rules perform context transformation and the cut-rule is a particular rule that makes proofs simpler. We also distinguish *left* and *right* rules, depending on the side they act on in the concluding sequent. An example of a logical inference rule is arrow introduction on the right:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ (R\to)$$

It has a single premise and a single conclusion. It introduces the formula $A \to B$ on the right-hand side of the conclusion, and this formula is called *the principal formula* of that inference rule. Formulae that are shown explicitly in the premise, namely A and B, are called *active formulae* and they are involved in forming the principal formula. The other formulae in the context are called *side formulae*.

It is possible to define many variants of Gentzen sequent systems. The basic Gentzen systems for classical and intuitionistic logic denoted as $G1$, $G2$ and $G3$ are formalized in Kleene [48] and later revisited in Troelstra and Schwichtenberg [69]. Briefly, the essential difference between $G1$ and $G3$ is the presence/absence of the explicit structural rules. The distinguishing point in the case of $G2$ is the use of the so-called mix-rule instead of the more common cut-rule. There are also sequent systems in which some structural rules are forbidden. They define various *substructural logics*, which are out of the scope of this thesis.

Originally, Gentzen presented a two-sided variant with explicitly given structural rules, namely *weakening, contraction* and *exchange*. The explicit exchange rule requires the contexts to be treated as *lists*. Structural rules can be also treated implicitly (or hidden) by reformulating the other rules in the system and changing the treatment of contexts. For example, if we wish to hide the exchange rule while keeping other structural rules explicit, then the contexts should be treated as *multisets*. And if we treat contexts as *sets*, all structural rules become implicit in the obtained system.

We will present the implicative fragments of intuitionistic sequent systems $G1$ and $G3$, since they represent the logical framework of the formal calculi elaborated in this thesis.

Figure 2.1 presents the system $G1$. Symbols A, B, C are used to denote formulae and Γ, Γ' denote contexts, which are in this case lists of formulae. Structural rules of weakening, contraction and exchange are explicitly given. All structural rules act only on the left-hand side of the sequents, due to the intuitionistic setting (while the classical system also contains right-hand side counterparts of the structural rules). The axiom (Ax) does not involve a premise. Inference rules with two

$$\frac{}{A \vdash A}\ (Ax)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ (R\to)$$

$$\frac{\Gamma \vdash A \quad \Gamma', B \vdash C}{\Gamma, \Gamma', A \to B \vdash C}\ (L\to)$$

$$\frac{\Gamma \vdash A \quad \Gamma', A \vdash B}{\Gamma, \Gamma' \vdash B}\ (Cut)$$

$$\frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ (Weakening)$$

$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\ (Contraction)$$

$$\frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C}\ (Exchange)$$

Figure 2.1: System $G1$

premises are given in the context-splitting (i.e. multiplicative) style, where Γ, Γ' represents a concatenated list made of the lists Γ and Γ'.

In the rest of the thesis, when speaking about the sequent calculus with explicit structural rules, we will use the modification of the system $G1$, in which the $(Exchange)$ rule is incorporated into other inference rules by treating the contexts as multisets of formulae, instead of as lists of formulae. In that setting, Γ, Γ' will represent the multi-union of contexts Γ and Γ'.

Figure 2.2 presents the sequent system $G3$, obtained from $G1$ by absorbing all structural rules into the remaining rules. Structural rules are hidden in the form of the logical rules and cut-rule, and thus performed automatically, without the possibility to control them. In this system, contexts are treated as finite sets of formulae. Inference rules with two premises are given in the context-sharing (i.e. additive) style. The definition of the axiom rule involves a context, thus allowing arbitrary formulae to be introduced at that level, i.e. weakening rule is hidden in the form of the axiom. In the rest of the thesis, when speaking about the sequent calculus with implicit structural rules, we will use the $G3$ system.

$$\frac{}{\Gamma, A \vdash A}\ (Ax)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ (R\to)$$

$$\frac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C}\ (L\to)$$

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B}\ (Cut)$$

Figure 2.2: System $G3$

Gentzen proved that the sequent calculus satisfies *the cut-elimination theorem*, which states that any sequent provable using the cut-rule, can be proved without the use of this rule. The cut-elimination theorem, also known as *Hauptsatz*, is the central result establishing the significance of the sequent calculus[1]. Moreover, the cut-free proofs in the sequent calculus satisfy *the subformula property*: the premises of the inference rules contain only subformulas of their respective conclusions.

Due to the cut-elimination theorem, the modification of the $G3$ system without (Cut) rule is also valid, and usually denoted by LJ^{cf}.

$$\frac{}{\Gamma, A \vdash A}\ (Ax)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ (R\to)$$

$$\frac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C}\ (L\to)$$

Figure 2.3: System LJ^{cf}

[1]A short and simple proof of Gentzen's Hauptsatz theorem, obtained via a variant of the simply typed λ-calculus, is given by Barendregt and Ghilezan in [7].

Chapter 3

Related term calculi

In this chapter, we briefly recall basic notions of four intuitionistic term calculi that are the most relevant for the calculi investigated in this thesis: λ-calculus, $\bar{\lambda}$-calculus, $\lambda\mathsf{lxr}$-calculus and $\lambda_{®}$-calculus.

The inevitable starting point for any research in the area of computational interpretations of logics is of course the λ-calculus. Since our interest is directed towards intersection types and sequent calculus, we present two type assignment systems for the λ-calculus - the system $\mathcal{D}$, proposed by Krivine in [49] and the system λLJ, proposed by Barendregt and Ghilezan in [7].

The following calculus, probably of the greatest relevance for the development of computational interpretations of LJ, is Herbelin's $\bar{\lambda}$-calculus, proposed in [38], the first intuitionistic term calculus that caught the essence of the sequent-style computation, and therefore served as an inspiration for the design of all other intuitionistic sequent term calculi, the λ^{Gtz}-calculus being one of them.

In the part of the research that explicitly treats operators of duplication and erasure of variables (the so-called resource control) our work was mostly inspired by the $\lambda\mathsf{lxr}$-calculus of Kesner and Lengrand [45], where a simply typed calculus with explicit substitution, weakening and contraction was proposed and its properties investigated.

Finally, some technical results regarding the strong normalisation of the system $\lambda_{®}^{\mathsf{Gtz}}\cap$ (Section 5.3) rely on the analogous results obtained for $\lambda_{®}\cap$ - the $\lambda_{®}$-calculus with intersection types. This system, proposed by Ghilezan et al. in [32], was initially proposed as an auxiliary system with the goal of creating a simpler resource control setting in which $\lambda_{®}^{\mathsf{Gtz}}\cap$ could be translated.

This chapter is divided into four sections, each of them briefly recalling one of the previously mentioned calculi.

3.1 The λ-calculus

The abstract syntax of the λ-calculus is given by:

$$\text{Terms} \qquad M \ ::= \ x \,|\, \lambda x.M \,|\, MM$$

A λ-term, denoted by $M, N, P, M_1, ...$, is either a variable from a denumerable set of term variables $\mathbb{T} = \{x, y, z, ..., x_1, ...x', ...\}$, an abstraction $\lambda x.M$, or an application MN. The set of all λ-terms is denoted by Λ.

In the term $\lambda x.M$ the variable x is bound. The scope of the binder extends to the right as much as possible. Free variables of a term are the ones that are not bound in the term. We denote the set of free variables of a term M by $Fv(M)$.

Definition 3.1 *The set of free variables of a term M is inductively defined as follows:*

$$\begin{array}{lcl} Fv(x) & = & \{x\} \\ Fv(\lambda x.M) & = & Fv(M) \setminus \{x\}; \\ Fv(MN) & = & Fv(M) \cup Fv(N). \end{array}$$

Definition 3.2 *The substitution of a term for a free variable in a λ-term is inductively defined as follows:*

$$\begin{array}{lcl} x[N/x] & = & N; \\ y[N/x] & = & y; \\ (MP)[N/x] & = & M[N/x]P[N/x]; \\ (\lambda x.M)[N/x] & = & \lambda x.M; \\ (\lambda y.M)[N/x] & = & \lambda y.(M[N/x]), \textit{ if } y \notin Fv(N); \\ (\lambda y.M)[N/x] & = & (\lambda y'.M[y'/y])[N/x], \textit{ if } y \in Fv(N), \textit{ and } y' \notin Fv(N). \end{array}$$

When writing λ-terms, we assume the following associativity conventions:

$$\lambda x_1 x_2 ... x_n.M \ = \ \lambda x_1.(\lambda x_2.(...(\lambda x_n.M)...));$$

$$M_1 M_2 M_3 ... M_n \ = \ (...((M_1 M_2) M_3)...M_n)$$

We also assume that a variable cannot be both free and bound in the same term, the so called Barendregt's convention, and thus, when necessary, we perform the renaming of bound variables which is known as α-conversion:

$$\lambda x.M \ \leftrightarrow_\alpha \ \lambda y.M[y/x] \quad \text{if } y \notin Fv(M).$$

The basic computational step in the λ-calculus is β-reduction:

$$(\lambda x.M)N \ \rightarrow_\beta \ M[N/x]$$

Definition 3.3 *(i) A term of the form $(\lambda x.M)N$ is called a β-redex, and the term of the form $M[N/x]$ is its contractum.*

(ii) A term is in β-normal form if it does not have a β-redex as a subterm.

(iii) A λ-term M is strongly normalising, if all reduction sequences starting from M terminate. The set of all strongly normalising λ-terms is denoted by $\mathcal{SN}$.

The characterisation of the set $\mathcal{SN}$ was one of the key problems of theoretical computer science, because only computing with strongly normalising terms would ensure the termination of a program. A solution to this problem was found in the lambda calculus with intersection types.

The system $\mathcal{D}$

The lambda calculus with simple types was introduced already by Church [11]. It is known that if a λ-term is typeable with simple types, then it is strongly normalising, but the converse of this statement turned to be false. As the matter of fact, there are already normal forms that cannot be typed in the simply typed lambda calculus. A typical example is the term $\lambda x.xx$.

Thus, the solution to the problem of describing the set of strongly normalising lambda terms by means of typeability was found in a type system capable of assigning types to more terms than the system with simple types could. The system, sometimes referred to as Torino system, was proposed independently by Coppo and Dezani [13], Sallé [63] and Pottinger [57]. The basic idea is that a term could have more than one type assigned to it, and in that case an intersection of these types is also a type assigned to the term. That simple and natural idea turned out to be successful in overcoming some limitations of the simple type discipline.

Here we present only one among many versions of type systems for the lambda calculi with intersection types, chosen because of its usage in Section 4.2.2.

Definition 3.4 *The syntax of types in the system $\mathcal{D}$ is defined as follows:*

$$\alpha \ ::= \ p \mid \alpha \to \alpha \mid \alpha \cap \alpha$$

where p ranges over a denumerable set of type atoms.

Types are denoted by $\alpha, \beta, \gamma, \alpha_1, \ldots$[1]

[1]We understand that this choice of names could cause clashes with reduction rule names, but we assume that the right meaning will always be clear from the context.

Definition 3.5

(i) A basic type assignment *is an expression of the form $x : \alpha$, where x is a term variable and α is a type.*

(ii) A basis *(or a context) Γ is a set $\{x_1 : \alpha_1, \dots, x_n : \alpha_n\}$ of basic type assignments, where all term variables are different.*

(iii) A domain of the basis Γ *is the set $Dom(\Gamma) = \{x_1, \dots, x_n\}$.*

(iv) A basis extension $\Gamma, x : \alpha$ *denotes the set $\Gamma \cup \{x : \alpha\}$, where $x \notin Dom(\Gamma)$.*

(v) A type assignment *is an expression $\Gamma \vdash M : \alpha$, denoting that a type α can be assigned to a term M in a context Γ.*

The type assignment system for the λ-calculus with intersection types, named $\mathcal{D}$, is presented in Figure 3.1. It is obtained from the system with simple types by adding the rules for introduction $(\cap I)$ and elimination $(\cap E)$ of intersection.

$$\frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ (Ax)$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta}\ (\to I) \qquad \frac{\Gamma \vdash M : \alpha \to \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta}\ (\to E)$$

$$\frac{\Gamma \vdash M : \alpha \quad \Gamma \vdash M : \beta}{\Gamma \vdash M : \alpha \cap \beta}\ (\cap I) \qquad \frac{\Gamma \vdash M : \alpha_1 \cap \alpha_2}{\Gamma \vdash M : \alpha_1}\ (\cap E)$$

Figure 3.1: System $\mathcal{D}$: intersection types for the λ-calculus

The intersection introduced here is assumed to be commutative, associative and idempotent. Therefore, the rule $(\cap E)$ can be applied as follows:
If $\Gamma \vdash M : \alpha_1 \cap \alpha_2 ... \cap \alpha_n$, then $\Gamma \vdash M : \alpha_i$, for any $i \in \{1, ..., n\}$.

Lemma 3.6 *The following two rules are admissible in the system $\mathcal{D}$:*

$$\frac{\Gamma \vdash M : \alpha \quad \Gamma \subseteq \Gamma'}{\Gamma' \vdash M : \alpha}\ (Weak) \qquad \frac{\Gamma \vdash N : \alpha \quad \Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash M[N/x] : \beta}\ (Subst)$$

The most important property of the system $\mathcal{D}$ is formulated in the following proposition, and proved in [60, 49, 30].

Theorem 3.7 ([60]) *A λ-term M is typeable in the system $\mathcal{D}$ if and only if $M \in \mathcal{SN}$.*

The system λLJ

The very first attempts of connecting a term calculus and the system *LJ* consisted of keeping the original syntax and reduction rules of the λ-calculus and expressing the sequent nature of the term calculus by modifying only the type assignment rules. This approach was used by Barendregt and Ghilezan [7]. In their paper, they offered a simple, short and elegant proof of the Gentzen's *Hauptsatz* theorem.

Type assignment rules of the system λLJ are given in Figure 3.2.

$$\frac{}{\Gamma, x:\alpha \vdash x:\alpha}\ (Ax)$$

$$\frac{\Gamma \vdash N:\alpha \quad \Gamma, x:\beta \vdash M:\gamma}{\Gamma, y:\alpha \to \beta \vdash M[yN/x]:\gamma}\ (\to_{left})$$

$$\frac{\Gamma, x:\alpha \vdash M:\beta}{\Gamma \vdash \lambda x.M:\alpha \to \beta}\ (\to_{right})$$

$$\frac{\Gamma \vdash N:\alpha \quad \Gamma, x:\alpha \vdash M:\beta}{\Gamma \vdash M[N/x]:\beta}\ (cut)$$

Figure 3.2: λLJ: simply typed λ-calculus

The subsystem of the λLJ from which the rule (cut) is excluded is denoted by λLJ^{cf}.

Although the construction of the typing rules of the λLJ-calculus was done in complete accordance with Gentzen's sequent calculus, this calculus does not exhibit the Curry-Howard correspondence with the system LJ. The underlying problem will be illustrated by the following example.

Example 3.8 ([7]) *Consider the term $\lambda x.yz$. The type $\gamma \to \beta$ can be assigned to it in the context $z:\alpha,\ y:\alpha \to \beta$. In the standard (= natural deduction style) simple type assignment system this type can be derived in the following way:*

$$\frac{\dfrac{\dfrac{}{x:\gamma,\ y:\alpha \to \beta,\ z:\alpha \vdash y:\alpha \to \beta}\ (Ax) \qquad \dfrac{}{x:\gamma,\ y:\alpha \to \beta,\ z:\alpha \vdash z:\alpha}\ (Ax)}{x:\gamma,\ y:\alpha \to \beta,\ z:\alpha \vdash yz:\beta}\ (\to E)}{y:\alpha \to \beta,\ z:\alpha \vdash \lambda x.yz:\gamma \to \beta.}\ (\to I)$$

while in the system λLJ^{cf} we can do the same assignment in two different ways:

I way:

$$\dfrac{\dfrac{\dfrac{}{x:\gamma,\, z:\alpha \vdash z:\alpha}\,(Ax) \qquad \dfrac{}{x:\gamma,\, z:\alpha,\, u:\beta \vdash u:\beta}\,(Ax)}{x:\gamma,\, z:\alpha,\, y:\alpha\to\beta \vdash yz:\beta}\,(\to_{left})}{y:\alpha\to\beta,\, z:\alpha \vdash \lambda x.yz:\gamma\to\beta.}\,(\to_{right})$$

II way:

$$\dfrac{\dfrac{}{z:\alpha \vdash z:\alpha}\,(Ax) \qquad \dfrac{\dfrac{}{x:\gamma,\, z:\alpha,\, u:\beta \vdash u:\beta}\,(Ax)}{z:\alpha,\, u:\beta \vdash \lambda x.u:\gamma\to\beta}\,(\to_{right})}{y:\alpha\to\beta,\, z:\alpha \vdash \lambda x.yz:\gamma\to\beta.}\,(\to_{left})$$

These two derivations correspond to two ways of building the term

$$\begin{array}{lllll} u & \mapsto & yz & \mapsto & \lambda x.yz, \\ u & \mapsto & \lambda x.u & \mapsto & \lambda x.yz. \end{array}$$

From the previous example we can observe that the correspondence is not one-to-one, since more derivations correspond to one λ-term.

3.2 The $\bar{\lambda}$-calculus

The first solution to the problem presented in the previous subsection was proposed by Hugo Herbelin, in his 1995 paper [38]. He solved the problem by considering the restricted sequent calculus *LJT*, whose cut-free fragment LJT^{cf} is in one-to-one correspondence with normal forms of the λ-calculus. The construction of the system was inspired by the sequent system *LKT*, proposed for the classical logic by Danos, Joinet and Schellinx in [15]. The key point of these restricted systems is the introduction of the so-called stoup - a special place on the left-hand side of the sequent, that may contain a formula. If the stoup is filled in, the formula in it is selected, meaning that it behaves differently than the others. Therefore, we distinguish two sorts of sequents in *LJT*:

- sequents with empty stoup - $(\Gamma;\ \vdash A)$;
- sequents with filled stoup - $(\Gamma; B \vdash A)$.

The rules of *LJT* system are presented in Figure 3.3. This system contains explicit rule for contraction and two cut-rules, which differ in the position of the cut formula. If the cut formula is in the stoup, we use H-cut (head-cut), and if the cut formula is outside the stoup, M-cut (mid-cut) is used. In the cut-rules, the symbol Π in the stoup denotes that the stoup could be both empty or filled. The first four rules constitute the cut-free fragment of the system, called LJT^{cf}.

$$\frac{}{\Gamma; A \vdash A}\ (axiom) \qquad \frac{\Gamma, A; A \vdash B}{\Gamma, A; \vdash B}\ (cont)$$

$$\frac{\Gamma; \vdash A \quad \Gamma; B \vdash C}{\Gamma; A \to B \vdash C}\ (\to_{left}) \qquad \frac{\Gamma, A; \vdash B}{\Gamma; \vdash A \to B}\ (\to_{right})$$

$$\frac{\Gamma; \Pi \vdash A \quad \Gamma; A \vdash B}{\Gamma; \Pi \vdash B}\ (H-cut) \qquad \frac{\Gamma; \vdash A \quad \Gamma, A; \Pi \vdash B}{\Gamma; \Pi \vdash B}\ (M-cut)$$

Figure 3.3: The sequent system LJT

The type assignment system λLJT^{cf} that corresponds to the system LJT^{cf} is presented in Figure 3.4.

$$\frac{}{\Gamma; x : \alpha \vdash x : \alpha}\ (axiom) \qquad \frac{\Gamma, x : \alpha; x : \alpha \vdash M : \beta}{\Gamma, x : \alpha; \vdash M : \beta}\ (cont)$$

$$\frac{\Gamma; \vdash N : \alpha \quad \Gamma; x : \beta \vdash M : \gamma}{\Gamma; y : \alpha \to \beta \vdash M[yN/x] : \gamma}\ (\to_{left}) \qquad \frac{\Gamma, x : \alpha; \vdash M : \beta}{\Gamma; \vdash \lambda x.M : \alpha \to \beta}\ (\to_{right})$$

Figure 3.4: λLJT^{cf}

This system is in one-to-one correspondence with normal forms of the λ-calculus, which will be illustrated by following example.

Example 3.9 *In the system λLJT^{cf} the only possible way of typing the λ-term $\lambda x.yz$ in the context $\Gamma = \{x : \gamma, y : \alpha \to \beta, z : \alpha\}$ is the following:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : \gamma, y : \alpha \to \beta, z : \alpha; z : \alpha \vdash z : \alpha}\ (axiom)}{x : \gamma, y : \alpha \to \beta, z : \alpha; \vdash z : \alpha}\ (cont) \qquad \dfrac{}{x : \gamma, y : \alpha \to \beta, z : \alpha; u : \beta \vdash u : \beta}\ (axiom)}{x : \gamma, y : \alpha \to \beta, z : \alpha; y : \alpha \to \beta \vdash yz : \beta}\ (\to_{left})}{x : \gamma, y : \alpha \to \beta, z : \alpha; \vdash yz : \beta}\ (cont)}{y : \alpha \to \beta, z : \alpha; \vdash \lambda x.yz : \gamma \to \beta.}\ (\to_{right})$$

Notice that yz appeared in the $(\rightarrow_{left})$ *rule as the result of the substitution* $u[yz/u]$.

The other type derivation from Example 3.8 is no longer possible, because one cannot apply the rule $(\rightarrow_{right})$ unless the stoup is empty, and when empty, the stoup can be refilled only by using the axiom, which, on the other hand, cannot be applied at that point. Therefore, $(\rightarrow_{left})$ cannot follow after $(\rightarrow_{right})$ and the problem of many-to-one correspondence that existed in the λLJ calculus is fixed. There is a unique and clear way to build a term - one first has to build the applicative part of a term, and then to add abstractions.

However, there is another cause of mismatch between the λ-calculus and the sequent calculus, which appears in the presence of the cut-rule. Computationally, cut can be interpreted as substitution operator (as already seen in Figure 3.2), thus Herbelin chose to consider λ-calculus with explicit substitution, introduced in [8]. The explicit substitution is denoted by $M\langle x = N\rangle$, read as "replace all free occurrences of x in M by N"[2], and propagated in the term by reductions.

In the λ-calculus (either with implicit or explicit substitution) applications are left associated while building of the applicative part is done in the opposite direction. This mismatch becomes even more transparent during the process of computation. According to the third level of Curry-Howard correspondence, a proof simplification in logic is supposed to correspond to the term reduction in calculus. Applied to this case, cut-elimination should correspond to explicit substitution propagation. But as will be illustrated by the following example, this does not hold. In the following example, simple types are assigned to the λ-terms with explicit substitution using a system obtained from the system λLJT^{cf}, by replacing implicit substitution by explicit one in the rule $(\rightarrow_{left})$ and adding the following cut-rule:

$$\frac{\Gamma;\vdash N:\alpha \quad \Gamma,x:\alpha;\Pi\vdash M:\beta}{\Gamma;\Pi\vdash M\langle x=N\rangle:\beta}\ (M-cut)$$

Example 3.10 ([38]) *The following two proofs represent two consequent steps of the cut-elimination, therefore, they should assign the same type to two terms (the first one before and the second one after the substitution propagation).*
The abbreviation $\gamma \equiv \alpha_2 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \beta$ *is used.*
Proof I:

$$\frac{\Gamma;\vdash N:\alpha \qquad \dfrac{\Gamma,x:\alpha;\vdash M_1:\alpha_1 \quad \Gamma,x:\alpha;z:\gamma\vdash (zM_2...M_n):\beta}{\Gamma,x:\alpha;y:\alpha_1\rightarrow\gamma\vdash (yM_1M_2...M_n):\beta}\ (\rightarrow_{left})}{\Gamma;y:\alpha_1\rightarrow\gamma\vdash (yM_1M_2...M_n)\langle x=N\rangle:\beta,}\ (M-cut)$$

[2]Of course, one has to make sure that the free variables in N do not get bound in M after substitution, which is easily regulated using the Barendregt's convention.

Proof II:

$$\dfrac{\dfrac{\Gamma;\vdash N:\alpha \quad \Gamma,x:\alpha;\vdash M_1:\alpha_1}{\Gamma;\vdash M_1\langle x=N\rangle:\alpha_1}(M-cut) \quad \dfrac{\Gamma;\vdash N:\alpha \quad \Gamma,x:\alpha;z:\gamma\vdash(zM_2...M_n):\beta}{\Gamma;z:\gamma\vdash(zM_2...M_n)\langle x=N\rangle:\beta}(M-cut)}{\Gamma;y:\alpha_1\to\gamma\vdash(yM_1\langle x=N\rangle M_2...M_n)\langle x=N\rangle:\beta.}(\to_{left})$$

The resulting term in the second proof is not what we expect to obtain by substitution propagation. Instead, we would like to obtain $(yM_1...M_{n-1})\langle x=N\rangle M_n\langle x=N\rangle)$. Therefore, the processes of cut-elimination and substitution propagation are not harmonized.

Herbelin fixed this mismatch by introducing a new structure - *a list of arguments*, which exhibits right associativity. In this way, he obtained a calculus in which the applicative expression is no longer of the form $((yM_1)...M_n)$, but of the form $y[M_1;...;M_n]$ which is in accordance with the sequent calculus derivations.

As a result of implementing these ideas, the first formal calculus corresponding to some intuitionistic sequent calculus is proposed, which showed that it is possible to extend the Curry-Howard paradigm also to the third formal logical system, besides Hilbert's axiomatic system and natural deduction.

The syntax of the $\bar{\lambda}$-calculus is given by:

$$\begin{array}{llll} \text{(Terms)} & t & ::= & xl\,|\,\lambda x.t\,|\,tl\,|\,t\langle x=t\rangle \\ \text{(Lists)} & l & ::= & [\,]\,|\,t::l\,|\,l@l\,|\,l\langle x=t\rangle \end{array}$$

where $[\,]$ denotes an empty list, $t::l$ denotes adding a new term into a list and $l@l'$ denotes concatenation of two lists into one.

Terms are denoted by $t,u,v...$[3] and lists are denoted by $l,l',...$ The complex syntax requires a more complex operational semantics. The reduction rules of the $\bar{\lambda}$ calculus are presented in Figure 3.5.

Reductions eliminate explicit substitution and explicit concatenation, thus normal forms of the $\bar{\lambda}$ calculus are of the form:

$$\begin{array}{lll} t_{nf} & ::= & xl_{nf}\,|\,\lambda x.t_{nf} \\ l_{nf} & ::= & [\,]\,|\,t_{nf}::l_{nf}. \end{array}$$

Type assignment rules of the simply typed $\bar{\lambda}$-calculus, that provide one-to-one correspondence with the sequent calculus *LJT* are given in Figure 3.6.

The expression $(\ .l)$, appearing in rules $(Cont)$, $(\to_L)$, (C_{H1}), (C_{H2}) and (C_{M2}) is called an applicative context. It contains a notation dot that saves a place for

[3]In the rest of the thesis, we will use small letters to denote expressions of the sequent term calculi and capital letters to denote terms of the natural deduction term calculi.

(β_{cons})	$(\lambda x.u)(v :: l)$	$\to$	$u\langle x = v\rangle l$
(β_{nil})	$(\lambda x.u)[\,]$	$\to$	$\lambda x.u$
(C_{var})	$(tl)l'$	$\to$	$t(l@l')$
(C_{cons})	$(t :: l)@l'$	$\to$	$t :: (l@l')$
(C_{nil})	$[\,]@l$	$\to$	l
(S_{yes})	$(xl)\langle x = v\rangle$	$\to$	$vl\langle x = v\rangle$
(S_{no})	$(yl)\langle x = v\rangle$	$\to$	$yl\langle x = v\rangle$
(S_{λ})	$(\lambda y.u)\langle x = v\rangle$	$\to$	$\lambda y.(u\langle x = v\rangle)$
(S_{nil})	$[\,]\langle x = v\rangle$	$\to$	$[\,]$
(S_{cons})	$(u :: l)\langle x = v\rangle$	$\to$	$u\langle x = v\rangle :: l\langle x = v\rangle.$

Figure 3.5: Reduction rules of the $\bar{\lambda}$-calculus

eventual term plugging. Formula $. : \alpha$ is called a hole declaration. One can observe that there are two kinds of sequents according to the content of the stoup: sequents with empty stoup are used for typing terms, while sequents containing hole declarations are used for typing applicative contexts. The strong normalisation of simply typed $\bar{\lambda}$-calculus is proved in [38].

The $\bar{\lambda}$-calculus has been the first extension of the λ-calculus that obtained one-to-one correspondence with some intuitionistic sequent calculus, therefore it was an inspiration for other sequent-style lambda calculi, including the λ^{Gtz}-calculus, that will be presented in details in Chapter 4.

3.3 The $\lambda\mathsf{lxr}$-calculus

The $\lambda\mathsf{lxr}$-calculus, proposed by Kesner and Lengrand in [45] is an intuitionistic term calculus with explicit operators for substitution, contraction and weakening. It was designed to provide a correspondence with the intuitionistic fragment of linear logic proof-nets, a formalism that gives an insight into the geometry of proof transformations. More precisely, both the syntax and operational semantics of $\lambda\mathsf{lxr}$ were extracted from the proof-nets' rules.

The main motivation for introducing operators for controlling duplication and erasure of terms, besides obtaining a computational interpretation of the MELL [4] proof-nets, was to enable more efficient control of the explicit substitution propagation. For example, if we want to explicitly perform the substitution $M\langle x = N\rangle$, it is very convenient to have a mechanism that indicates whether x appears free in

[4]MELL is an abbreviation for the multiplicative exponential fragment of linear logic.

$$\frac{}{\Gamma; . : \alpha \vdash (\,.[\,]) : \alpha}\ (Ax)$$

$$\frac{\Gamma, x : \alpha; . : \alpha \vdash (\,.l) : \beta}{\Gamma, x : \alpha; \vdash xl : \beta}\ (Cont)$$

$$\frac{\Gamma, x : \alpha; \vdash t : \beta}{\Gamma; \vdash \lambda x.t : \alpha \to \beta}\ (\to_R)$$

$$\frac{\Gamma; \vdash t : \alpha \quad \Gamma; . : \beta \vdash (\,.l) : \gamma}{\Gamma; . : \alpha \to \beta \vdash (\,.(t :: l)) : \gamma}\ (\to_L)$$

$$\frac{\Gamma; \vdash t : \alpha \quad \Gamma; . : \alpha \vdash (\,.l) : \beta}{\Gamma; \vdash tl : \beta}\ (C_{H1})$$

$$\frac{\Gamma; . : \gamma \vdash (\,.l) : \alpha \quad \Gamma; . : \alpha \vdash (\,.l') : \beta}{\Gamma; . : \gamma \vdash (\,.l@l') : \beta}\ (C_{H2})$$

$$\frac{\Gamma; \vdash t : \alpha \quad \Gamma, x : \alpha; \vdash u : \beta}{\Gamma; \vdash u\langle x = t\rangle : \beta}\ (C_{M1})$$

$$\frac{\Gamma; \vdash t : \alpha \quad \Gamma, x : \alpha; . : \gamma \vdash (\,.l) : \beta}{\Gamma; . : \gamma \vdash (\,.l\langle x = t\rangle) : \beta}\ (C_{M2})$$

Figure 3.6: Simply typed $\bar{\lambda}$-calculus

M, or in some subterm of M. Exactly that information is captured by the explicit weakening (i.e. erasure) operator - an expression $\mathcal{W}_x(M)$ denotes that M is weakened by a variable x, meaning that x has not already appeared free in M. So, in an expression like $(\mathcal{W}_x(M))\langle x = N\rangle$, we know that we can stop propagating the substitution into M, and thus the efficiency of the explicit substitution execution has been significantly improved. Although this connection between explicit weakening and substitution was already investigated by David and Guillaume and incorporated into the design of their calculus with labels λ_{ws} [16], Kesner and Lengrand also added explicit control of contraction (i.e. duplication) and thus obtained a sound and complete correspondence with proof-nets.

The abstract syntax of the λlxr-calculus is the following:

$$\text{Terms} \qquad t \quad ::= \quad x \,|\, \lambda x.t \,|\, tt \,|\, t\langle x = t\rangle \,|\, \mathcal{W}_x(t) \,|\, \mathcal{C}_x^{y|z}\,(t)$$

where x ranges over a denumerable set of term variables. Two new constructs are a weakening $\mathcal{W}_x(t)$, which introduces a free variable x, and a contraction $\mathcal{C}_x^{y|z}\,(t)$, which binds variables y and z and introduces a free variable x. Terms are denoted by $t, u, v, \ldots$

The notion of linearity of terms is assumed, meaning that only terms satisfying the following two conditions are considered:

- in every subterm, every variable has at most one free occurrence;
- every binder does bind a free occurrence of a variable.

Formally, linear terms are introduced by a system of inference rules, and it is showed that each non-linear term can be translated into a linear one.

As a consequence of already mentioned goal (full correspondence with the MELL proof-nets) the operational semantics consists of two segments:

- reduction relations;
- congruence equations.

Reduction rules for the λlxr-calculus are presented in Figure 3.7. They are divided into three groups: B-reduction, system x and system r. B is the standard β-reduction with explicit substitution, the system x is in charge of the substitution propagation, whereas the system r treats the interaction of newly introduced operators of contraction and weakening with the other term constructors. The operators of weakening and contraction are moving in opposite directions, which are imported from the proof-nets simplifications: the weakenings are pulled out to the top level while the contractions are propagated into a term. When two of them meet, they can either cross or merge. A notation $\mathcal{R}$ in the rule Merge denotes a meta operator of capture-free variable renaming. Free variables $Fv(t)$ are formally defined in [45].

It is proved that the reduction rules preserve the set of free variables and linearity constraint, and that the system enjoys confluence and the preservation of β-reduction.

A congruence relation is defined on terms, denoted by $\equiv$. It is the smallest reflexive, symmetric, transitive and context-closed relation induced by the rules given in Figure 3.8. It is proved that each congruence rule induces a congruence class containing finitely many terms. Preservation properties (of free variables and of linearity of terms) are also proved.

The simple types are introduced into the λlxr-calculus. The simply typed λlxr-calculus is defined by rules for assigning simple types to λlxr-terms that are given in Figure 3.9.

The properties concerning the typed λlxr-calculus are: subject reduction and subject equivalence, preservation of strong normalisation and strong normalisation.

B	$(\lambda x.t)u$	$\longrightarrow$	$t\langle x = u\rangle$	
System x				
Abs	$(\lambda y.t)\langle x = u\rangle$	$\longrightarrow$	$\lambda y.(t\langle x = u\rangle)$	
App$_1$	$(tv)\langle x = u\rangle$	$\longrightarrow$	$t\langle x = u\rangle v$	$x \in Fv(t)$
App$_2$	$(tv)\langle x = u\rangle$	$\longrightarrow$	$tv\langle x = u\rangle$	$x \in Fv(u)$
Var	$x\langle x = u\rangle$	$\longrightarrow$	u	
Weak$_1$	$\mathcal{W}_x(t)\langle x = u\rangle$	$\longrightarrow$	$\mathcal{W}_{Fv(u)}(t)$	
Weak$_2$	$\mathcal{W}_y(t)\langle x = u\rangle$	$\longrightarrow$	$\mathcal{W}_y(t\langle x = u\rangle)$	$x \neq y$
Cont	$\mathcal{C}_x^{y,z}(t)\langle x = u\rangle$	$\longrightarrow$	$\mathcal{C}_{Fv(u)}^{\Psi,\Upsilon}(t\langle y = u_1\rangle\langle z = u_2\rangle)$	
		where Ψ, Υ are fresh,	$u_1 = \mathcal{R}_{\Psi}^{Fv(u)}$,	$u_2 = \mathcal{R}_{\Upsilon}^{Fv(u)}$
Comp	$t\langle y = v\rangle\langle x = u\rangle$	$\longrightarrow$	$t\langle y = v\langle x = u\rangle\rangle$	$x \in Fv(v)$
System r				
WAbs	$\lambda x.\mathcal{W}_y(t)$	$\longrightarrow$	$\mathcal{W}_y(\lambda x.t)$	$x \neq y$
WApp1	$\mathcal{W}_y(u)v$	$\longrightarrow$	$\mathcal{W}_y(uv)$	
WApp2	$u\mathcal{W}_y(v)$	$\longrightarrow$	$\mathcal{W}_y(uv)$	
WSubs	$t\langle x = \mathcal{W}_y(u)\rangle$	$\longrightarrow$	$\mathcal{W}_y(t\langle x = u\rangle)$	
Merge	$\mathcal{C}_w^{y,z}(\mathcal{W}_y(t))$	$\longrightarrow$	$\mathcal{R}_w^z(t)$	
Cross	$\mathcal{C}_w^{y,z}(\mathcal{W}_x(t))$	$\longrightarrow$	$\mathcal{W}_x(\mathcal{C}_w^{y,z}(t))$	$x \neq y,\ x \neq z$
CAbs	$\mathcal{C}_w^{y,z}(\lambda x.t)$	$\longrightarrow$	$\lambda x.\mathcal{C}_w^{y,z}(t)$	
CApp1	$\mathcal{C}_w^{y,z}(tu)$	$\longrightarrow$	$\mathcal{C}_w^{y,z}(t)\,u$	$y,z \in Fv(t)$
CApp2	$\mathcal{C}_w^{y,z}(tu)$	$\longrightarrow$	$t\,\mathcal{C}_w^{y,z}(u)$	$y,z \in Fv(u)$
CSubs	$\mathcal{C}_w^{y,z}(t\langle x = u\rangle)$	$\longrightarrow$	$t\langle x = \mathcal{C}_w^{y,z}(u)\rangle$	$y,z \in Fv(u)$

Figure 3.7: Reduction rules for the λlxr-calculus

$\mathcal{C}_w^{x,v}((\mathcal{C}_x^{y,z}(t)))$	$\equiv$	$\mathcal{C}_w^{x,y}(\mathcal{C}_x^{z,v}(t))$	
$\mathcal{C}_x^{y,z}(t)$	$\equiv$	$\mathcal{C}_x^{z,y}(t)$	
$\mathcal{C}_{x'}^{y',z'}(\mathcal{C}_x^{y,z}(t))$	$\equiv$	$\mathcal{C}_x^{y,z}(\mathcal{C}_{x'}^{y',z'}(t))$	if $x \neq y', z'$ and $x' \neq y, z$
$\mathcal{W}_x(\mathcal{W}_y(t))$	$\equiv$	$\mathcal{W}_y(\mathcal{W}_x(t))$	
$t\langle x = u\rangle\langle y = v\rangle$	$\equiv$	$t\langle y = v\rangle\langle x = u\rangle$	if $y \notin Fv(u)$ and $x \notin Fv(v)$
$\mathcal{C}_w^{y,z}(t)\langle x = u\rangle$	$\equiv$	$\mathcal{C}_w^{y,z}(t\langle x = u\rangle)$	if $x \neq w$ and $y, z \notin Fv(t)$

Figure 3.8: Congruence equations for the λlxr-calculus

$$\frac{}{x:\alpha \vdash x:\alpha}\,(Axiom) \qquad \frac{\Gamma, x:\beta \vdash t:\alpha \quad \Gamma' \vdash v:\beta}{\Gamma,\Gamma' \vdash t\langle x=v\rangle : \alpha}\,(Subs)$$

$$\frac{\Gamma, x:\alpha \vdash t:\beta}{\Gamma \vdash \lambda x.t : \alpha \to \beta}\,(Lambda) \qquad \frac{\Gamma \vdash t:\alpha\to\beta \quad \Gamma' \vdash v:\alpha}{\Gamma,\Gamma' \vdash tv : \beta}\,(App)$$

$$\frac{\Gamma, x:\alpha, y:\alpha \vdash t:\beta}{\Gamma, z:\alpha \vdash \mathcal{C}_z^{x|y}(t) : \beta}\,(Cont) \qquad \frac{\Gamma \vdash t:\beta}{\Gamma, x:\alpha \vdash \mathcal{W}_x(t) : \beta}\,(Weak)$$

Figure 3.9: Simply typed λlxr-calculus

The proof of strong normalisation relies on the translation of simply typed λlxr-terms into MELL proof-nets, which are known to be strongly normalising.

Theorem 3.11 ([45], Strong normalisation) *The relation $\to_{\lambda\mathsf{lxr}}$ is strongly normalising on the set of typed* λlxr*-terms.*

3.4 The $\lambda_{®}$-calculus

Finally, we briefly present the $\lambda_{®}$-calculus, an extension of the standard λ-calculus with explicit weakening and contraction and implicit substitution. The $\lambda_{®}$ calculus with intersection types was initially proposed by Ghilezan et al. in [32] as an auxiliary system in which $\lambda_{®}^{\mathsf{Gtz}}\cap$ (the subject of Chapter 5) could be translated in order to prove the strong normalisation. The $\lambda_{®}$-calculus is simpler than both the λlxr-calculus (due to the implicit rather than explicit substitution) and the $\lambda_{®}^{\mathsf{Gtz}}$-calculus (because it is constructed in a natural deduction style instead of the sequent calculus one). It operationally corresponds to the λ_{cw}-calculus, one of the calculi of the Kesner and Renaud's Prismoid of resources [46, 47]. Together with λ, λ^{Gtz}, $\lambda_{®}^{\mathsf{Gtz}}$ and four other calculi, it constitutes a structure called Resource control cube, presented in Chapter 6.

This section contains a brief presentation of the system $\lambda_{®}\cap$, resource control lambda calculus with intersection types, as well as the properties of the system relevant for the content of Chapter 5. Motivation, detailed explanations and examples for most of the features of the system coincide with its sequent counterpart $\lambda_{®}^{\mathsf{Gtz}}\cap$, so they are included in Chapter 5 and therefore omitted here. Propositions are stated here without proofs, which are given in [33].

Untyped $\lambda_{®}$

The abstract syntax of $\lambda_{®}$ pre-terms is the following:

$$\text{Pre-terms} \qquad t \quad ::= \quad x \,|\, \lambda x.t \,|\, tt \,|\, x \odot t \,|\, x <^{x_1}_{x_2} t$$

where x ranges over a denumerable set of term variables, $\lambda x.t$ and tt are standard lambda abstraction and application, $x \odot t$ is a weakening and $x <^{x_1}_{x_2} t$ is a contraction.

The list of free variables of a pre-term t, denoted by $Fv[t]$, is defined as follows (where l,m denotes appending two lists and $l \setminus x$ denotes removing all occurrences of an element x from a list):

$$\begin{array}{lcl}
Fv[x] & = & x; \\
Fv[\lambda x.t] & = & Fv[t] \setminus x; \\
Fv[tu] & = & Fv[t], Fv[u]; \\
Fv[x \odot t] & = & x, Fv[t]; \\
Fv[x <^{x_1}_{x_2} t] & = & x, ((Fv[t] \setminus x_1) \setminus x_2).
\end{array}$$

The set of free variables of a pre-term t, denoted by $Fv(t)$, is extracted out of the list $Fv[t]$, by un-ordering the list and by removing the multiple occurrences of each variable, if any. The set of bound variables of a pre-term t, denoted by $Bv(t)$, contains all variables that exist in t, but are not free in it.

Now, using the notion of free variables, we can extract the set of $\lambda_{®}$-terms (denoted by $\Lambda_{®}$) out of the set of $\lambda_{®}$ pre-terms. The definition of $\lambda_{®}$-terms is given in Figure 3.10. Terms will from now on be denoted by $M, N, P, \ldots$.

$$\frac{}{x \in \Lambda_{®}} \qquad \frac{t \in \Lambda_{®} \quad x \in Fv(t)}{\lambda x.t \in \Lambda_{®}}$$

$$\frac{t \in \Lambda_{®} \quad u \in \Lambda_{®} \quad Fv(t) \cap Fv(u) = \emptyset}{tu \in \Lambda_{®}}$$

$$\frac{t \in \Lambda_{®} \quad x \notin Fv(t)}{x \odot t \in \Lambda_{®}}$$

$$\frac{t \in \Lambda_{®} \quad x_1 \neq x_2 \quad x_1, x_2 \in Fv(t) \quad x \notin Fv(t) \setminus \{x_1, x_2\}}{x <^{x_1}_{x_2} t \in \Lambda_{®}}$$

Figure 3.10: $\lambda_{®}$-terms

The operational semantics of the $\lambda_{®}$-calculus consists of reduction rules given in Figure 3.11 and equivalencies given in Figure 3.12. Like in the case of the λlxr-calculus, computation is directed toward propagation of contraction and extraction of weakening.

$$\begin{array}{rrcl}
(\beta) & (\lambda x.M)N & \to & M[N/x] \\
(\gamma_1) & x <^{x_1}_{x_2} (\lambda y.M) & \to & \lambda y.x <^{x_1}_{x_2} M \\
(\gamma_2) & x <^{x_1}_{x_2} (MN) & \to & (x <^{x_1}_{x_2} M)N, \text{ if } x_1, x_2 \notin Fv(N) \\
(\gamma_3) & x <^{x_1}_{x_2} (MN) & \to & M(x <^{x_1}_{x_2} N), \text{ if } x_1, x_2 \notin Fv(M) \\
(\omega_1) & \lambda x.(y \odot M) & \to & y \odot (\lambda x.M),\ x \neq y \\
(\omega_2) & (x \odot M)N & \to & x \odot (MN) \\
(\omega_3) & M(x \odot N) & \to & x \odot (MN) \\
(\gamma\omega_1) & x <^{x_1}_{x_2} (y \odot M) & \to & y \odot (x <^{x_1}_{x_2} M),\ y \neq x_1, x_2 \\
(\gamma\omega_2) & x <^{x_1}_{x_2} (x_1 \odot M) & \to & M[x/x_2]
\end{array}$$

Figure 3.11: Reduction rules of the $\lambda_{®}$-calculus

$$\begin{array}{lrcl}
(\varepsilon_1) & x \odot (y \odot M) & \equiv & y \odot (x \odot M) \\
(\varepsilon_2) & x <^{x_1}_{x_2} M & \equiv & x <^{x_2}_{x_1} M \\
(\varepsilon_3) & x <^{y}_{z} (y <^{u}_{v} M) & \equiv & x <^{y}_{u} (y <^{z}_{v} M) \\
(\varepsilon_4) & x <^{x_1}_{x_2} (y <^{y_1}_{y_2} M) & \equiv & y <^{y_1}_{y_2} (x <^{x_1}_{x_2} M),\ x \neq y_1, y_2,\ y \neq x_1, x_2
\end{array}$$

Figure 3.12: Equivalences in the $\lambda_{®}$-calculus

The inductive definition of the meta operator [/], representing the implicit substitution of free variables, is given by:

$$\begin{array}{rcl}
x[N/x] & \triangleq & N \\
(\lambda y.M)[N/x] & \triangleq & \lambda y.M[N/x],\ x \neq y \\
(MP)[N/x] & \triangleq & M[N/x]P,\ x \notin Fv(P) \\
(MP)[N/x] & \triangleq & MP[N/x],\ x \notin Fv(M) \\
(y \odot M)[N/x] & \triangleq & y \odot M[N/x],\ x \neq y \\
(x \odot M)[N/x] & \triangleq & Fv(N) \odot M \\
(y <^{y_1}_{y_2} M)[N/x] & \triangleq & y <^{y_1}_{y_2} M[N/x],\ x \neq y \\
(x <^{x_1}_{x_2} M)[N/x] & \triangleq & Fv[N] <^{Fv[N_1]}_{Fv[N_2]} M[N_1/x_1][N_2/x_2]
\end{array}$$

By inspecting reductions, equivalencies and substitution definition, one can observe that an interface preservation of $\lambda_{\circledR}$-terms is satisfied during computation, meaning that the the set of free variables does not change (contrary to the λ-calculus). For that reason, the $\lambda_{\circledR}$-calculus is said to be *a resource aware calculus*.

The system $\lambda_{\circledR}\cap$

In order to obtain a type assignment system flexible enough to capture all strongly normalising $\lambda_{\circledR}$-terms, intersection types were introduced into the $\lambda_{\circledR}$-calculus. As already said, this system was developed after the intersection type assignment system for the $\lambda_{\circledR}^{\mathsf{Gtz}}$-calculus, thus all essential ideas were borrowed from the system $\lambda_{\circledR}^{\mathsf{Gtz}}\cap$ and are explained in Chapter 5. Here, we provide the reader only with a minimal content inevitable for stating necessary propositions. Observe that simple types can be assigned to the $\lambda_{\circledR}$-terms as in Figure 3.9, without the rule $(Subs)$.

In the system $\lambda_{\circledR}\cap$, presented in Figure 3.13, we assign strict types to $\lambda_{\circledR}$-terms. The syntax of strict types and all corresponding notions are defined in Subsection 5.3.1.

$$\frac{}{x:\sigma \vdash x:\sigma}\ (Ax) \qquad \frac{\Gamma, x:\alpha \vdash M:\sigma}{\Gamma \vdash \lambda x.M : \alpha \to \sigma}\ (\to_I)$$

$$\frac{\Gamma \vdash M : \cap_i^n \tau_i \to \sigma \quad \Delta_0 \vdash N:\tau_0 \quad \Delta_1 \vdash N:\tau_1 \quad \ldots \quad \Delta_n \vdash N:\tau_n}{\Gamma, \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n \vdash MN:\sigma}\ (\to_E)$$

$$\frac{\Gamma, x:\alpha, y:\beta \vdash M:\sigma}{\Gamma, z:\alpha\cap\beta \vdash z <^x_y M:\sigma}\ (Cont) \qquad \frac{\Gamma \vdash M:\sigma}{\Gamma, x:\top \vdash x \odot M:\sigma}\ (Weak)$$

Figure 3.13: $\lambda_{\circledR}\cap$: $\lambda_{\circledR}$-calculus with intersection types

Strong normalisation for the presented system is proved in [33], by using an adaptation of the reducibility method.

Proposition 3.12 ([33], Typeability $\Rightarrow$ SN in $\lambda_{\circledR}\cap$) *If $\Gamma \vdash M:\sigma$, then M is strongly normalising, i.e. $M \in \mathcal{SN}$.*

The following proposition claims the other direction, i.e. the characterisation of strong normalisation. The proof relies on typeability of normal forms and head subject expansion.

Proposition 3.13 ([33], SN $\Rightarrow$ Typeability in $\lambda_{\circledR}\cap$) *All strongly normalising $\lambda_{\circledR}$-terms are typeable in the $\lambda_{\circledR}\cap$ system.*

As an immediate consequence of the previous two propositions, one obtains the characterisation theorem.

Theorem 3.14 ([33]) *In the $\lambda_{\circledR}$-calculus, a term M is strongly normalising if and only if it is typeable in $\lambda_{\circledR}\cap$.*

Apart from the four presented calculi, there are other term calculi that also involve some kind of resource control. In the classical setting, Lescanne and Žunić expanded the classical sequent $\mathcal{X}$-calculus of van Bakel et al. [71] and proposed the $*\mathcal{X}$-calculus, a calculus that computationally corresponds to the classical sequent calculus with the explicit structural rules of weakening and contraction [52, 75]. Due to the classical nature of this calculus, its syntax contains both left and right eraser and duplicator operators. The notation for the resource operators used in the rest of this thesis ($x \odot M$ for contraction and $x <^{y}_{z} M$ for contraction), is taken from the $*\mathcal{X}$-calculus.

A different approach to the resource aware lambda calculus, motivated mostly by the development of the process calculi, was investigated by Boudol in [9]. Instead of extending the syntax of the λ-calculus with explicit resource operators, Boudol proposed a non-deterministic calculus with a generalised notion of application. In his work, a function is applied to a structure called a bag, having the form $(N_1^{m_1}|...|N_k^{m_k})$ in which N_i, $i = 1,...,k$ are resources and $m_i \in \mathbb{N} \cup \{\infty\}$, $i = 1,...,k$ are multiplicities, representing the maximum possible number of the resource usage. In this framework, the usual application is written as MN^{∞}. A variant of this calculus was typed with non-idempotent intersection types by Pagani and Ronchi della Rocca in [55].

Chapter 4

λ^{Gtz}-calculus

Following Herbelin's ideas from [38] and trying to generalise his results to the full sequent calculus, José Espírito Santo and Luis Pinto first proposed the lambda calculus with generalised multiary application λ^{Jm}. Later, a modified term calculus named the λ^{Gtz}-calculus (pronounced *lambda Gentzen* calculus) was proposed by Espírito Santo in [27]. Its simply-typed version provides, in the context of the implicative intuitionistic logic, the Curry-Howard correspondence for the full sequent calculus. This was an improvement with respect to the $\bar{\lambda}$-calculus, where the correspondence was obtained between simply typed $\bar{\lambda}$-terms and the restricted sequent calculus *LJT*. The main difference between the two calculi, on the syntactical level, is the generalised form of the list from the $\bar{\lambda}$-calculus, called a context in λ^{Gtz}. Using λ^{Gtz} notation, lists are of the form $t_1 :: t_2 :: ... :: t_n :: \widehat{x}.x$ (corresponding to $t_1 :: t_2 :: ... :: t_n :: [\,]$ in $\bar{\lambda}$), whereas the contexts have more general form $t_1 :: t_2 :: ... :: t_n :: \widehat{x}.t$. This modification, consequently, enabled a construction of a syntax directed type assignment system, in which all cut rules are merged into one.

This chapter is entirely devoted to the λ^{Gtz}-calculus. Section 4.1 and Subsection 4.2.1 revisit Espirito Santo's results on the untyped and the simply typed λ^{Gtz}-calculus, while the results involving intersection type systems and the characterisation of strong normalisation represent the original contribution of the thesis. These results are developed by José Espírito Santo, Silvia Ghilezan, Silvia Likavec and myself, and are published in [24, 31, 41, 42, 25].

4.1 Type-free λ^{Gtz}-calculus

We start by presenting the syntax of the λ^{Gtz}-calculus, its operational semantics, and some properties of the untyped version of this calculus.

The abstract syntax of the λ^{Gtz}-calculus is given by:

$$\begin{array}{lrcl} \text{Terms} & t & ::= & x \,|\, \lambda x.t \,|\, tk \\ \text{Contexts} & k & ::= & \widehat{x}.t \,|\, t :: k \end{array}$$

The main characteristic of the λ^{Gtz}-calculus is the existence of two syntactic categories - terms and contexts. A term is either a variable from a denumerable set of term variables $\mathbb{T} = \{x, y, z, ..., x_1, ...x', ...\}$, an abstraction $\lambda x.t$, or an application tk, usually called a *cut*. A context is either a *selection* $\widehat{x}.t$ or a *context constructor* (*linear left introduction*) $t :: k$ (the operator :: is read *cons*). Terms and contexts are together referred to as *expressions*. We use the notation $t, u, v, t_1..$ for the terms, $k, k', ...$ for the contexts and $e, e', ...$ for the expressions.

As pointed out in [28], computationally, the contexts represent a prescription of what to do next with an expression which is plugged into it. A selection $\widehat{x}.t$ means "substitute for x in t" whereas a cons $t :: k$ means "apply to t and proceed according to k".

If one uses the usual analogy with the function theory, contexts could be understood as lists of arguments. They are constructed from a term by selecting a variable in the term (which corresponds to choosing an active formula in Gentzen's sequent calculus). A new element could be added to the list using concatenation, performed via the *cons* operator. There are no context variables - the trivial context is $\widehat{x}.x$, which corresponds to an empty list $[\,]$.

Notice that a cut tk represents a plugging of a term t into a context k, which is a key difference with respect to the ordinary λ-calculus application tt. We distinguish two interpretations of a cut, according to the form of the context k. If k is a selection, cut represents an explicit substitution $t(\widehat{x}.v)$. If k is a cons, cut represents *a multiary generalised application* $t(u_1 :: \cdots :: u_m :: \widehat{x}.v)$, for some $m \geq 1$. In the last case, if $m = 1$, the cut is reduced to a notion of generalised application $t(u :: \widehat{x}.v)$ introduced in [44]; whereas if $v = x$, we get a multiary application $t(u_1 :: \cdots :: u_m :: [])$ (usually written as $t[u_1; \cdots; u_m]$) introduced in [21]. Finally, if both $m = 1$ and $v = x$, the cut is reduced to the form of an ordinary application.

In expressions $\lambda x.t$ and $\widehat{x}.t$, the variable x is bound. The scope of the binders extends to the right as much as possible. For example, we write $\widehat{x}.tu$ instead of $\widehat{x}.(tu)$. Free variables are the ones that are neither bound by abstraction nor by selection operator. Barendregt's convention, stating that the bound variables in an expression should be denoted differently from the free ones, applies in both cases. We denote the free variables of an expression e by $Fv(e)$.

Definition 4.1 *The set of free variables of an expression e is inductively defined as follows:*

$$\begin{array}{lcl} Fv(x) & = & \{x\}; \\ Fv(\lambda x.t) & = & Fv(t) \setminus \{x\}; \\ Fv(tk) & = & Fv(t) \cup Fv(k); \\ Fv(\widehat{x}.t) & = & Fv(t) \setminus \{x\}; \\ Fv(t :: k) & = & Fv(t) \cup Fv(k). \end{array}$$

The computation with the λ^{Gtz}-expressions is carried out by the reduction rules of the λ^{Gtz}-calculus, presented in Figure 4.1.

$$\boxed{\begin{array}{llcl} (\beta) & (\lambda x.t)(u :: k) & \to & u\widehat{x}.tk \\ (\pi) & (tk)k' & \to & t(k@k') \\ (\sigma) & t\widehat{x}.v & \to & v[t/x] \\ (\mu) & \widehat{x}.xk & \to & k, \text{ if } x \notin k \end{array}}$$

Figure 4.1: Reduction rules of the λ^{Gtz}-calculus

Reduction rules are executed via two meta-operators, namely *the substitution* $[\ /\ ]$ and *the append* @. $[\ /\ ]$ denotes the regular implicit substitution defined in Figure 4.2. In what follows, we use a relation symbol $\triangleq$ to denote an equality by definition. The meta-operator @, called append, is used for joining two contexts. It is defined by the rules presented in Figure 4.3.

$$\boxed{\begin{array}{rcl} x[u/x] & \triangleq & u \\ y[u/x] & \triangleq & y \\ (\lambda y.t)[u/x] & \triangleq & \lambda y.t[u/x] \\ (tk)[u/x] & \triangleq & t[u/x]k[u/x] \\ (\widehat{y}.t)[u/x] & \triangleq & \widehat{y}.t[u/x] \\ (t :: k)[u/x] & \triangleq & t[u/x] :: k[u/x] \end{array}}$$

Figure 4.2: Substitution in the λ^{Gtz}-calculus

$$\boxed{\begin{array}{lcl} (t :: k)@k' & \triangleq & t :: (k@k'); \\ (\widehat{x}.t)@k' & \triangleq & \widehat{x}.tk'. \end{array}}$$

Figure 4.3: Meta-operator @ in the λ^{Gtz}-calculus

The rule (β) generates a substitution but it is the rule (σ) that executes it, on the meta-level. The rule (π) simplifies the head of a cut (t is the *head* of tk). The rule μ (whose origin can be found in [66]) has a structural character and it either performs a trivial substitution in the reduction $t(\widehat{x}.xk) \to tk$, or it minimizes the use of the generality feature in the reduction $t(u_1 \cdots u_m :: \widehat{x}.xk) \to t(u_1 \cdots u_m :: k)$.

As already mentioned, the substitution in the λ^{Gtz}-calculus is defined via a meta-operator, i.e. it is not part of the syntax, hence the λ^{Gtz}-calculus does not belong to the group of calculi with explicit substitution. However, this calculus supports the possibility of delayed substitutions (due to the definition of the β-reduction), which is one of the main properties of the explicit substitution calculi.

The rules (β), (π) and (σ) aim at eliminating all cuts but those of the trivial[1] form $y(u_1 :: \cdots u_m :: \widehat{x}.v)$ (for some $m \geq 1$). In that way, reductions in the λ^{Gtz}-calculus correspond to the cut-elimination process in the sequent calculus, providing one level of the Curry-Howard correspondence.

The set of $\beta\pi\sigma$-normal forms is given by the following abstract syntax:

$$\begin{array}{lrcl} \text{(Terms)} & t_{nf} & = & x \mid \lambda x.t_{nf} \mid x(t_{nf} :: k_{nf}) \\ \text{(Contexts)} & k_{nf} & = & \widehat{x}.t_{nf} \mid t_{nf} :: k_{nf}. \end{array}$$

Now we illustrate the operational semantics of λ^{Gtz} with the following examples.

Example 4.2 *Let us consider the term $t \equiv (\lambda x.y)(y(\widehat{z}.z) :: \widehat{x}.\lambda y.x)$. To avoid the clash of free and bound variables, before the computation we apply Barendregt's convention and rename bound occurrences of the variables x and y yielding $t \equiv (\lambda x.y)(y(\widehat{z}.z) :: \widehat{x'}.\lambda y'.x')$. Now, we can reduce t as follows:*
first way:

$$\begin{array}{rcl} t & \to_\beta & (y\widehat{z}.z)\widehat{x}.(y\widehat{x'}.\lambda y'.x') \\ & \to_\pi & y\,((\widehat{z}.z)@\widehat{x}.(y\widehat{x'}.\lambda y'.x')) \\ & \triangleq & y\widehat{z}.(z\widehat{x}.(y\widehat{x'}.\lambda y'.x') \\ & \to_\sigma & y\widehat{z}.\, y\widehat{x'}.(\lambda y'.x')[z/x] \\ & \triangleq & y\widehat{z}.(y\widehat{x'}.\lambda y'.x') \\ & \to_\sigma & y\widehat{z}.\,(\lambda y'.x')[y/x'] \\ & \triangleq & y\widehat{z}.\lambda y'.y \\ & \to_\sigma & (\lambda y'.y)[y/z] \\ & \triangleq & \lambda y'.y. \end{array}$$

Second way:

[1]A cut is called trivial if its head is a variable.

$$\begin{array}{rcl} t & \rightarrow_\beta & (y\widehat{z}.z)\widehat{x}.(y\widehat{x'}.\lambda y'.x') \\ & \rightarrow_\sigma & (y\widehat{x'}.\lambda y'.x')[y\widehat{z}.z/x] \\ & \triangleq & y\widehat{x'}.\lambda y'.x' \\ & \rightarrow_\sigma & (\lambda y'.x')[y/x'] \\ & \triangleq & \lambda y'.y. \end{array}$$

Although in the previous example both ways of reducing led to the same normal form, the λ^{Gtz}-calculus is not confluent. The reason is a critical pair that exists between reductions π and σ [2]. Thus, the problem analogous to the "call-by-name/call-by-value dilemma" of Curien-Herbelin's $\bar{\lambda}\mu\tilde{\mu}$-calculus [14] exists also in the λ^{Gtz}-calculus, and will be illustrated by the following example.

Example 4.3 *Let $t_0 \equiv (tk)(\widehat{x}.v)$. This term is both a π-redex and a σ-redex. Contracting it as a π-redex (the call-by-value option) we get $t_1 \equiv t(k@(\widehat{x}.v))$. Contracting it as a σ redex (the call-by-name option) we get $t_2 \equiv v[tk/x]$. In a number of cases, these two terms cannot be reduced to the same normal form.*
Consider, for example, this particular case: $t = z$, $v = y$, and $k = u :: (\widehat{w}.w)$, where z and y are variables, $y \neq x$, and u is a normal form. Then
the call-by-value option:

$$\begin{array}{rcl} t_0 & \equiv & (z(u :: \widehat{w}.w))(\widehat{x}.y) \\ & \rightarrow_\pi & z((u :: \widehat{w}.w)@(\widehat{x}.y)) \\ & \triangleq & z(u :: (\widehat{w}.w@(\widehat{x}.y))) \\ & \triangleq & z(u :: (\widehat{w}.w(\widehat{x}.y))) \\ & \rightarrow_\mu & z(u :: \widehat{x}.y). \end{array}$$

the call-by-name option:

$$\begin{array}{rcl} t_0 & \equiv & (z(u :: \widehat{w}.w))(\widehat{x}.y) \\ & \rightarrow_\sigma & y[z(u :: \widehat{w}.w)/x] \\ & \triangleq & y. \end{array}$$

Obviously, obtained normal forms differ.

However, the confluence can be regained by quite simple modifications on the syntax and the operational semantics. Two confluent λ^{Gtz} sub-calculi will be presented in the Section 4.4.

[2]Non-confluent expressions are those λ^{Gtz}-expressions in which both π and σ reductions can be performed at the same time.

4.2 Typed λ^{Gtz}-calculus

4.2.1 Simply typed λ^{Gtz}-calculus

The basic type assignment system for the λ^{Gtz}-calculus is the one with simple types, introduced by Espírito Santo in [27] and denoted by $\lambda^{\mathsf{Gtz}} \to$.

Definition 4.4 *The syntax of simple types is defined as follows:*

$$\alpha \ ::= \ p \mid \alpha \to \alpha$$

where p ranges over a denumerable set of type atoms.

Types will be denoted by $\alpha, \beta, \gamma, \alpha_1, ...$ and the set of all simple types will be denoted by $\mathrm{T}_{\to}$.

Definition 4.5

(i) A basic type assignment *is an expression of the form $x : \alpha$, where x is a term variable and α is a type.*

(ii) A basis Γ *is a set $\{x_1 : \alpha_1, \ldots, x_n : \alpha_n\}$ of basic type assignments, where all term variables are different.*

(iii) A domain of the basis Γ *is the set $Dom(\Gamma) = \{x_1, \ldots, x_n\}$.*

(iv) A basis extension $\Gamma, x : \alpha$ *denotes the set $\Gamma \cup \{x : \alpha\}$, where $x \notin Dom(\Gamma)$.*

(v) There are two kinds of type assignments*:*

- $\Gamma \vdash t : \alpha$ - *a type assignment for terms;*
- $\Gamma; \beta \vdash k : \alpha$ - *a type assignment for contexts.*

Like in Herbelin's $\bar{\lambda}$-calculus, the system $\lambda^{\mathsf{Gtz}} \to$ contains the special place between the symbols ; and $\vdash$ on the left-hand side of the sequents in the type assignments for contexts. This place is called *the stoup* and it contains a selected formula with which we continue the computation.

Definition 4.6 *The type assignment system $\lambda^{\mathsf{Gtz}} \to$ is given by the rules presented in Figure 4.4.*

$$\frac{}{\Gamma, x:\alpha \vdash x:\alpha}\ (Ax)$$

$$\frac{\Gamma, x:\alpha \vdash t:\beta}{\Gamma \vdash \lambda x.t:\alpha \to \beta}\ (\to_R) \qquad \frac{\Gamma \vdash t:\alpha \quad \Gamma;\beta \vdash k:\gamma}{\Gamma;\alpha \to \beta \vdash t::k:\gamma}\ (\to_L)$$

$$\frac{\Gamma \vdash t:\alpha \quad \Gamma;\alpha \vdash k:\beta}{\Gamma \vdash tk:\beta}\ (Cut) \qquad \frac{\Gamma, x:\alpha \vdash t:\beta}{\Gamma;\alpha \vdash \widehat{x}.t:\beta}\ (Sel)$$

Figure 4.4: $\lambda^{\mathsf{Gtz}} \to$: the simply typed λ^{Gtz}-calculus

The system $\lambda^{\mathsf{Gtz}} \to$ consists of five type-assignment rules (one for each kind of λ^{Gtz}-expression) given in the context-sharing style. There are two rules for typing contexts, namely (Sel) and $(\to_L)$, that contain the stoup and three rules for typing terms, namely (Ax), $(\to_R)$ and (Cut). Apart from the stoup, it is clear that the system $\lambda^{\mathsf{Gtz}} \to$ represents exactly the sequent calculus system with implicit structural rules (system $G3$), decorated with λ^{Gtz}-expressions. Thus, the formulae-as-types side of the Curry-Howard correspondence between the simply-typed λ^{Gtz}-calculus and sequent calculus LJ is satisfied.

The system $\lambda^{\mathsf{Gtz}} \to$ satisfies the following properties, stated here without proofs[3].

Proposition 4.7 (Generation lemma for $\lambda^{\mathsf{Gtz}} \to$)

(i) $\Gamma \vdash x:\alpha$ *iff* $x:\alpha \in \Gamma$.

(ii) $\Gamma \vdash \lambda x.t:\alpha$ *iff* $\alpha \equiv \beta \to \gamma$ *and* $\Gamma, x:\beta \vdash t:\gamma$.

(iii) $\Gamma;\alpha \vdash \widehat{x}.t:\beta$ *iff* $\Gamma, x:\alpha \vdash t:\beta$.

(iv) $\Gamma \vdash tk:\alpha$ *iff there is a type* β *such that* $\Gamma \vdash t:\beta$, *and* $\Gamma;\beta \vdash k:\alpha$.

(v) $\Gamma;\alpha \vdash t::k:\beta$ *iff* $\alpha \equiv \gamma \to \delta,,\ \Gamma \vdash t:\gamma$ *and* $\Gamma;\delta \vdash k:\beta$.

Proposition 4.8 (Substitution lemma for $\lambda^{\mathsf{Gtz}} \to$)

(i) *If* $\Gamma, x:\alpha \vdash t:\beta$ *and* $\Gamma \vdash u:\alpha$, *then* $\Gamma \vdash t[u/x]:\beta$.

(ii) *If* $\Gamma, x:\alpha;\gamma \vdash k:\beta$ *and* $\Gamma \vdash u:\alpha$, *then* $\Gamma;C \vdash k[u/x]:\beta$.

[3]The corresponding properties of the system with intersection types, which is the focus of this research, will be elaborated in the following subsection in details, hence we preferred to avoid unnecessary repetition in this subsection.

Proposition 4.9 (Append lemma for $\lambda^{\mathsf{Gtz}} \to$) *If* $\Gamma;\gamma \vdash k:\beta$ *and* $\Gamma;\beta \vdash k':\alpha$, *then* $\Gamma;\gamma \vdash k@k':\alpha$.

Theorem 4.10 (Subject reduction for $\lambda^{\mathsf{Gtz}} \to$)

(i) If $\Gamma \vdash t:\alpha$ *and* $t \to t'$, *then* $\Gamma \vdash t':\alpha$.

(ii) If $\Gamma;\beta \vdash k:\alpha$ *and* $k \to k'$, *then* $\Gamma;\beta \vdash k':\alpha$.

The preservation of type under reductions (i.e. subject reduction) shows which proof transformation of the sequent calculus corresponds to each reduction rule of the λ^{Gtz}-calculus. The rule (β) corresponds to the key-step in cut-elimination, whereas the rules (σ) and (π) correspond to right and left permutation of cuts, respectively. The rule (μ) undoes the sequence of two inference steps consisting of deselecting the stoup formula, without contraction, and, immediately after, selecting the same formula. In that way, we can see that the simplification (i.e. reduction) of a term corresponds to the simplification (i.e. cut-elimination) of its type derivation, which is exactly the third level of Curry-Howard correspondence.

Espírito Santo proved strong normalisation for the system $\lambda^{\mathsf{Gtz}} \to$ in [27], by translating it into λ-calculus with "delayed" substitutions [26]. But, as in the case of the simply typed λ-calculus, the basic type assignment system $\lambda^{\mathsf{Gtz}} \to$ cannot characterise all strongly normalising λ^{Gtz}-terms. For example, the term $\lambda x.x(x :: \widehat{y}.y)$ (which corresponds to the term $\lambda x.xx$ in the λ-calculus) is a λ^{Gtz} normal form, yet it does not have a type in $\lambda^{\mathsf{Gtz}} \to$.

Example 4.11 *In the λ-calculus, the term $\lambda xy.xy$ is typed with $(\alpha \to \beta) \to (\alpha \to \beta)$. The corresponding term in the λ^{Gtz}-calculus is $\lambda x.\lambda y.x(y :: \widehat{z}.z)$, and here we give its typing in the system $\lambda^{\mathsf{Gtz}} \to$:*

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$x:\alpha\to\beta, y:\alpha \vdash x:\alpha\to\beta$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$x:\alpha\to\beta, y:\alpha \vdash y:\alpha$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$x:\alpha\to\beta, z:\beta \vdash z:\beta$}
\RightLabel{(Sel)}
\UnaryInfC{$x:\alpha\to\beta;\beta \vdash \widehat{z}.z:\beta$}
\RightLabel{$(\to_L)$}
\BinaryInfC{$x:\alpha\to\beta, y:\alpha;\alpha\to\beta \vdash y :: \widehat{z}.z:\beta$}
\RightLabel{(Cut)}
\BinaryInfC{$x:\alpha\to\beta, y:\alpha \vdash x(y :: \widehat{z}.z):\beta$}
\RightLabel{$(\to_R)$}
\UnaryInfC{$x:\alpha\to\beta \vdash \lambda y.x(y :: \widehat{z}.z):\alpha\to\beta$}
\RightLabel{$(\to_R)$}
\UnaryInfC{$\vdash \lambda x.\lambda y.x(y :: \widehat{z}.z):(\alpha\to\beta)\to(\alpha\to\beta)$}
\end{prooftree}
$$

In the previous example, we have seen the very simple way to translate a λ-term into corresponding λ^{Gtz}-term. This translation is, however, valid only for the set of normal forms. The general mapping from the λ-calculus to the λ^{Gtz}-calculus will be defined later.

4.2.2 Intersection types for the λ^{Gtz}-calculus

The intersection types were introduced into the λ^{Gtz}-calculus by Espírito Santo, Ghilezan and Ivetić in [24] in order to obtain a type assignment system in which all strongly normalising terms would be typeable. The detailed account on this system is given in [25].

Definition 4.12 *The syntax of intersection types is defined as follows:*

$$\alpha \ ::= \ p \mid \alpha \to \alpha \mid \alpha \cap \alpha$$

where p ranges over a denumerable set of type atoms.

Types will be denoted by $\alpha, \beta, \gamma, \alpha_1, \ldots$ and the set of all intersection types will be denoted by $\mathrm{T}_\cap$.

As usual, the set $\mathrm{T}_\cap$ is partitioned into equivalence classes induced by the following equivalence relation.

Definition 4.13

(i) Pre-order $\leq$ over the set of types is the smallest relation that satisfies the following rules:

1. $\alpha \leq \alpha$;
2. $\alpha \cap \beta \leq \alpha$ *and* $\alpha \cap \beta \leq \beta$;
3. $(\alpha \to \beta) \cap (\alpha \to \gamma) \leq \alpha \to (\beta \cap \gamma)$;
4. $\alpha \leq \beta$ *and* $\beta \leq \gamma$ *implies* $\alpha \leq \gamma$;
5. $\alpha \leq \beta$ *and* $\alpha \leq \gamma$ *implies* $\alpha \leq \beta \cap \gamma$[4];
6. $\alpha' \leq \alpha$ *and* $\beta \leq \beta'$ *implies* $\alpha \to \beta \leq \alpha' \to \beta'$.

(ii) Two types are equivalent, $\alpha \sim \beta$, *if and only if* $\alpha \leq \beta$ *and* $\beta \leq \alpha$.

In the rest of this chapter, we will consider types modulo the introduced equivalence relation. We will assume that $\cap$ operator has the priority over the $\to$ operator, therefore $\alpha \to \beta \cap \gamma$ stands for $\alpha \to (\beta \cap \gamma)$. Also, the abbreviation

$$\cap_i^n \alpha_i \equiv \alpha_1 \cap \ \ldots \ \cap \alpha_n \text{ for some } n \geq 1$$

will be used throughout the chapter.

The following equivalencies follow from the Definition 4.13, therefore they will be used in the rest of this chapter without explicit mentioning.

[4]It is possible to replace this rule with the following two rules: 5.1 $\alpha \leq \alpha \cap \alpha$; 5.2 $\alpha' \leq \alpha$ and $\beta' \leq \beta$ implies $\alpha' \cap \beta' \leq \alpha \cap \beta$.

Lemma 4.14

(i) $(\alpha \to \beta) \cap (\alpha \to \gamma) \sim \alpha \to \beta \cap \gamma$;

(ii) $\cap_i^n(\alpha \to \beta_i) \sim \alpha \to \cap_i^n \beta_i$;

(iii) $\alpha \cap \alpha \sim \alpha$;

(iv) $\alpha \cap \beta \sim \beta \cap \alpha$;

(v) $(\alpha \cap \beta) \cap \gamma \sim \alpha \cap (\beta \cap \gamma)$.

Proof:

(i) The direction $(\alpha \to \beta) \cap (\alpha \to \gamma) \leq \alpha \to \beta \cap \gamma$ holds directly from the rule 3 of Definition 4.13. We prove that $\alpha \to \beta \cap \gamma \leq (\alpha \to \beta) \cap (\alpha \to \gamma)$. From $\alpha \leq \alpha$ (rule 1 of Definition 4.13) and $\beta \cap \gamma \leq \beta$ (rule 2 of Definition 4.13), applying rule 6 of Definition 4.13, we have $\alpha \to \beta \cap \gamma \leq \alpha \to \beta$. Analogously, $\alpha \to \beta \cap \gamma \leq \alpha \to \gamma$ holds since $\alpha \leq \alpha$ and $\beta \cap \gamma \leq \gamma$. Now, from $\alpha \to \beta \cap \gamma \leq \alpha \to \beta$ and $\alpha \to \beta \cap \gamma \leq \alpha \to \gamma$ we get $\alpha \to \beta \cap \gamma \leq (\alpha \to \beta) \cap (\alpha \to \gamma)$ using the rule 5 of Definition 4.13.

(ii) The proof goes by induction on i, with the previous case (i) of this proof being the base case for $i = 2$.

(iii) The direction $\alpha \cap \alpha \leq \alpha$ follows from rule 2 of Definition 4.13. The direction $\alpha \leq \alpha \cap \alpha$ follows from rule 5 of Definition 4.13, applied to $\alpha \leq \alpha$ and $\alpha \leq \alpha$.

(iv) Both inequalities follow from rules 2 and 5 of Definition 4.13, one time starting from $\alpha \cap \beta \leq \beta$ and $\alpha \cap \beta \leq \alpha$, and another time starting from $\beta \cap \alpha \leq \alpha$ and $\beta \cap \alpha \leq \beta$.

(v) By applying rules 2, 4 and 5 of Definition 4.13. $\square$

Definitions of a type assignment, a basis and related notions are analogous to the ones from Definition 4.5 and therefore omitted.

Definition 4.15 *The type assignment system $\lambda^{\mathsf{Gtz}}\cap$ is given by the rules presented in Figure 4.5.*

It is easy to observe that the system $\lambda^{\mathsf{Gtz}}\cap$ is an extension of the system $\lambda^{\mathsf{Gtz}} \to$, given in Figure 4.4. The latter can be obtained from the former by taking a restriction $n = 1$ in the rules (Ax), $(\to_L)$ and (Cut).

$$\frac{}{\Gamma, x : \alpha_1 \cap \dots \cap \alpha_n \vdash x : \alpha_1}\ (Ax)$$

$$\frac{\Gamma, x : \alpha \vdash t : \beta}{\Gamma \vdash \lambda x.t : \alpha \to \beta}\ (\to_R) \qquad \frac{\Gamma \vdash t : \alpha_1\ \cdots\ \Gamma \vdash t : \alpha_n \qquad \Gamma; \beta \vdash k : \gamma}{\Gamma; \cap_i^n \alpha_i \to \beta \vdash t :: k : \gamma}\ (\to_L)$$

$$\frac{\Gamma, x : \alpha \vdash v : \beta}{\Gamma; \alpha \vdash \widehat{x}.v : \beta}\ (Sel) \qquad \frac{\Gamma \vdash t : \alpha_1\ \cdots\ \Gamma \vdash t : \alpha_n \qquad \Gamma; \cap_i^n \alpha_i \vdash k : \beta}{\Gamma \vdash tk : \beta}\ (Cut)$$

Figure 4.5: $\lambda^{\mathsf{Gtz}}\cap$: intersection types for the λ^{Gtz}-calculus

Notice that in the system $\lambda^{\mathsf{Gtz}}\cap$ there are no separate rules for the intersection introduction and for intersection elimination, contrary to the usual way of introducing intersection types in the λ-calculus, proposed by Coppo and Dezani-Ciancaglini in [13]. Also, the typing rule for $\leq$ is not included in our system. The management of intersection is built into the other rules, where necessary, and thus the proposed type assignment system exhibits the important feature of syntax-directness.

Also, notice that due to Lemma 4.14 where commutativity and associativity of intersection are proved, we can also use the weaker form of the axiom rule:

$$\frac{}{\Gamma, x : \cap_i^n \alpha_i \vdash x : \alpha_j}\ (Ax),\ j \in \{1, ..., n\}.$$

The following proposition, called Generation lemma, explains the way of unfolding the type assignment derivations. It represents one of the crucial technical lemmas, since it is used in the proofs of most of the other propositions. In the case of the system $\lambda^{\mathsf{Gtz}}\cap$, this lemma trivially holds due to the fact that the system is *syntax-directed*, meaning that there is exactly one type assignment rule corresponding to each sort of λ^{Gtz}-expression.

Proposition 4.16 (Generation lemma for $\lambda^{\mathsf{Gtz}}\cap$)

(i) $\Gamma \vdash x : \alpha$ *iff* $x : \cap_i^n \alpha_i \in \Gamma$ *and* $\alpha \equiv \alpha_j$, *for some* $j \leq n$.

(ii) $\Gamma \vdash \lambda x.t : \alpha$ *iff* $\alpha \equiv \beta \to \gamma$ *and* $\Gamma, x : \beta \vdash t : \gamma$.

(iii) $\Gamma; \alpha \vdash \widehat{x}.t : \beta$ *iff* $\Gamma, x : \alpha \vdash t : \beta$.

(iv) $\Gamma \vdash tk : \alpha$ *iff there is a type* $\beta \equiv \cap_i^n \beta_i$ *such that* $\Gamma \vdash t : \beta_i$, *for all* $i \in \{1, \dots, n\}$, *and* $\Gamma; \cap_i^n \beta_i \vdash k : \alpha$.

(v) $\Gamma;\alpha \vdash t :: k : \beta$ *iff* $\alpha \equiv \cap_i^n \gamma_i \to \delta$, $\Gamma;\delta \vdash k : \beta$, *and* $\Gamma \vdash t : \gamma_i$ *for all* $i \in \{1,\dots,n\}$.

Proof: The proof is straightforward since all the rules are syntax-directed. □

The left introduction of intersection is an admissible rule in this system, which is the subject of the following proposition.

Proposition 4.17 (Admissible rule - $(\cap_L)$)

(i) *If* $\Gamma, x : \alpha_j \vdash t : \beta$, *for some* $j \leq n$, *then* $\Gamma, x : \cap_i^n \alpha_i \vdash t : \beta$.

(ii) *If* $\Gamma, x : \alpha_j;\gamma \vdash k : \beta$, *for some* $j \leq n$, *then* $\Gamma, x : \cap_i^n \alpha_i;\gamma \vdash k : \beta$.

Proof: By simultaneous induction on the structure of terms and contexts.

- Case $t \equiv x$. Let $\Gamma, x : \alpha_j \vdash x : \beta$, then from the rule (Ax) we conclude that $\alpha_j = \cap_l^m \beta_{jl}$, $l \in L = \{1,\dots,m\}$ and $\beta \equiv \beta_{jl_0}$ for some $l_0 \in L$. Then $\cap_i^n \alpha_i = \cap_i^n(\cap_l^m \beta_{il})$, $l \in L, i \in \{1,\dots,n\}$ so applying the rule (Ax) we get $\Gamma, x : \cap_i^n \alpha_i \vdash x : \beta$.

- Case $t \equiv y$. This case is trivially satisfied because the type of one variable does not affect the type of the other one.

- Case $t \equiv \lambda y.t'$. Let $\Gamma, x : \alpha_j \vdash \lambda y.t' : \beta$. Then by Generation lemma 4.16(ii) $\beta \equiv \gamma \to \delta$ and $\Gamma, x : \alpha_j, y : \gamma \vdash t' : \delta$. Applying the IH on t' we get $\Gamma, x : \cap_i^n \alpha_i, y : \gamma \vdash t' : \delta$, and conclude with $\Gamma, x : \cap_i^n \alpha_i \vdash \lambda y.t' : \beta$ using the rule $(\to_R)$.

- Case $k \equiv \widehat{y}.t'$. Let $\Gamma, x : \alpha_j;\delta \vdash \widehat{y}.t' : \beta$. This derivation could only be generated from $\Gamma, x : \alpha_j, y : \delta \vdash t' : \beta$, according to Generation lemma 4.16(iii). Applying the IH on t' we get $\Gamma, x : \cap_i^n \alpha_i, y : \delta \vdash t' : \beta$, and then by the rule (Sel) we get $\Gamma, x : \cap_i^n \alpha_i;\delta \vdash \widehat{y}.t' : \beta$.

- Case $t \equiv t'k$. Let $\Gamma, x : \alpha_j \vdash t'k : \beta$. Then, according to Generation lemma 4.16 (iv), there is a type $\cap_l^m \gamma_l$ such that $\Gamma, x : \alpha_j \vdash t' : \gamma_l$, for all $l \in \{1,\dots,m\}$ and $\Gamma, x : \alpha_j;\cap_l^m \gamma_l \vdash k : \beta$. Applying the IH on both t' and k we get $\Gamma, x : \cap_i^n \alpha_i \vdash t' : \gamma_l$ for all $l \in \{1,\dots,m\}$ and $\Gamma, x : \cap_i^n \alpha_i;\cap_l^m \gamma_l \vdash k : \beta$, and then by the rule (Cut) we get $\Gamma, x : \cap_i^n \alpha_i \vdash t'k : \beta$.

- Case $k \equiv t' :: k'$. Let $\Gamma, x : \alpha_j;\cap_l^m \gamma_l \to \delta \vdash t' :: k' : \beta$. According to Generation lemma 4.16(v), the corresponding premises are $\Gamma, x : \alpha_j \vdash t' : \gamma_l$, for all $l \in \{1,\dots,m\}$ and $\Gamma, x : \alpha_j;\delta \vdash k' : \beta$. Applying the IH on both t' and k' we get $\Gamma, x : \cap_i^n \alpha_i \vdash t' : \gamma_l$ for all $l \in \{1, \dots, m\}$, and $\Gamma, x : \cap_i^n \alpha_i ; \delta \vdash k' : \beta$, and then by the rule $(\rightarrow_L)$ we get $\Gamma, x : \cap_i^n \alpha_i ; \cap_l^m \gamma_l \rightarrow \delta \vdash t' :: k' : \beta$. $\square$

The type assignment system $\lambda^{\mathsf{Gtz}}\cap$ satisfies the following standard properties.

Lemma 4.18

(i) If $\Gamma \vdash t : \alpha$*, then* $Fv(t) \subseteq Dom(\Gamma)$.

(ii) If $\Gamma ; \beta \vdash k : \alpha$*, then* $Fv(k) \subseteq Dom(\Gamma)$.

Proof: By case analysis on the type assignment rules. Free variables are introduced only by the rule (Ax). In the rules $(\rightarrow_R)$ and (Sel) declaration $x : A$ is removed from the basis, which corresponds to the binding of x by abstraction or selection operator. In the remaining two rules neither the size of the basis nor the number of free variables changes. $\square$

Proposition 4.19 (Basis expansion)

(i) $\Gamma \vdash t : \alpha \;\Leftrightarrow\; \Gamma, x : \beta \vdash t : \alpha$ *and* $x \notin Fv(t)$.

(ii) $\Gamma ; \gamma \vdash k : \alpha \;\Leftrightarrow\; \Gamma, x : \beta ; \gamma \vdash k : \alpha$ *and* $x \notin Fv(k)$.

Proof: The proof follows from the definition of a basis and Lemma 4.18. $\square$

Definition 4.20 (Bases intersection)

$$\begin{array}{rcl} \Gamma_1 \cap \Gamma_2 & = & \{x : \alpha \mid x : \alpha \in \Gamma_1 \;\&\; x \notin Dom(\Gamma_2)\} \\ & \cup & \{x : \alpha \mid x : \alpha \in \Gamma_2 \;\&\; x \notin Dom(\Gamma_1)\} \\ & \cup & \{x : \alpha \cap \beta \mid x : \alpha \in \Gamma_1 \;\&\; x : \beta \in \Gamma_2\}. \end{array}$$

Proposition 4.21 (Bases intersection)

(i) $\Gamma_1 \vdash t : \alpha \;\Rightarrow\; \Gamma_1 \cap \Gamma_2 \vdash t : A$.

(ii) $\Gamma_1 ; \beta \vdash k : \alpha \;\Rightarrow\; \Gamma_1 \cap \Gamma_2 ; \beta \vdash k : A$.

Proof: By induction on the number of elements in $Dom(\Gamma_2)$, using Proposition 4.19 and Proposition 4.17. $\square$

As already stated, the substitution and the append are two meta-operators of the λ^{Gtz}-calculus. The following two lemmas, namely Substitution lemma and Append lemma, explain the behaviour of these meta-operators in the presence of intersection types.

Lemma 4.22 (Substitution lemma for $\lambda^{\mathsf{Gtz}}\cap$)

(i) If $\Gamma, x : \cap_i^n \alpha_i \vdash t : \beta$ *and* $\Gamma \vdash u : \alpha_j$, *for all* $j \in \{1, \dots, n\}$, *then* $\Gamma \vdash t[u/x] : \beta$.

(ii) If $\Gamma, x : \cap_i^n \alpha_i ; \gamma \vdash k : \beta$ *and* $\Gamma \vdash u : \alpha_j$, *for all* $j \in \{1, \dots, n\}$, *then* $\Gamma; \gamma \vdash k[u/x] : \beta$.

Proof: (i) and (ii) are proved by simultaneous induction on the structure of t and k.

- t is a variable:
  - $t \equiv x$:
    From $\Gamma, x : \cap_i^n \alpha_i \vdash x : \beta$, using Generation lemma 4.16(i) we derive $\beta \equiv \alpha_j$, for some $j \in \{1, \dots, n\}$. Since $x[u/x] \triangleq u$ the proof is contained in the second premise.
  - $t \equiv y$:
    From $\Gamma, x : \cap_i^n \alpha_i \vdash y : \beta$ and Proposition 4.19 we derive that $\Gamma \vdash y : \beta$. Since $y[u/x] \triangleq y$ the proof is complete.
- $t \equiv \lambda y.v$:
  From $\Gamma, x : \cap_i^n \alpha_i \vdash \lambda y.v : \beta$, using Generation lemma 4.16(ii) we get $\beta \equiv \gamma \to \delta$ and $\Gamma, x : \cap_i^n \alpha_i, y : \gamma \vdash v : \delta$. Applying the induction hypothesis to v we get $\Gamma, y : \gamma \vdash v[u/x] : \delta$. Since $(\lambda y.v)[u/x] \triangleq \lambda y.v[u/x]$, the proof is complete using the rule $(\to_R)$.
- $t \equiv vk$:
  From $\Gamma, x : \cap_i^n \alpha_i \vdash vk : \beta$, using Generation lemma 4.16(iv), we derive that there exists a type $\cap_j^m \gamma_j$, $j = 1, ..., m$, $m \geq 1$, such that $\Gamma, x : \cap_i^n \alpha_i \vdash v : \gamma_j, \forall j \in \{1, \dots, m\}$ and $\Gamma, x : \cap_i^n \alpha_i ; \cap_j^m \gamma_j \vdash k : \beta$. Applying the induction hypothesis to v and k we get:

  $$\frac{\Gamma \vdash v[u/x] : \gamma_1 \ \cdots \ \Gamma \vdash v[u/x] : \gamma_m \quad \Gamma; \cap_j^m \gamma_j \vdash k[u/x] : \beta}{\Gamma \vdash v[u/x]k[u/x] : \beta} \ (Cut)$$

  This is exactly what we need since $(vk)[u/x] \triangleq v[u/x]k[u/x]$.
- $k \equiv \widehat{y}.v$:
  From $\Gamma, x : \cap_i^n \alpha_i ; \gamma \vdash \widehat{y}.v : \beta$, using Generation lemma 4.16(iii), we get $\Gamma, x : \cap_i^n \alpha_i, y : \gamma \vdash v : \beta$. Applying the induction hypothesis to v we get

  $$\frac{\Gamma, y : \gamma \vdash v[u/x] : \beta}{\Gamma; \gamma \vdash \widehat{y}.v[u/x] : \beta} \ (Sel)$$

  This ends the proof since $(\widehat{y}.v)[u/x] \triangleq \widehat{y}.v[u/x]$.

- $k \equiv t :: k'$:
  From $\Gamma, x : \cap_i^n \alpha_i; \gamma \vdash t :: k' : \beta$, using Generation lemma 4.16(v), we derive $\gamma \equiv \cap_j^m \delta_j \to \varepsilon$, $\Gamma, x : \cap_i^n \alpha_i; \varepsilon \vdash k' : \beta$, and $\Gamma, x : \cap_i^n \alpha_i \vdash t : \delta_j, \forall j \in \{1, \dots, m\}$. Applying the induction hypothesis to t and k' we get

$$\frac{\Gamma \vdash t[u/x] : \delta_1 \;\cdots\; \Gamma \vdash t[u/x] : \delta_m \quad \Gamma; \varepsilon \vdash k'[u/x] : \beta}{\Gamma; \cap_j^m \delta_j \to \varepsilon \vdash t[u/x] :: k'[u/x] : \beta}\,(\to_L)$$

  Since $\cap_j^m \delta_j \to \varepsilon \equiv \gamma$ and $(t :: k')[u/x] \triangleq t[u/x] :: k'[u/x]$, the proof is complete. □

Lemma 4.23 (Append lemma for $\lambda^{\mathsf{Gtz}}\cap$) *If for all $i \in \{1, \dots, n\}$, $\Gamma; \gamma \vdash k : \beta_i$ and $\Gamma; \cap_i^n \beta_i \vdash k' : \alpha$, then $\Gamma; \gamma \vdash k@k' : \alpha$.*

Proof: By induction on the structure of k.

- $k \equiv \widehat{x}.v$:
  From $\Gamma; \gamma \vdash \widehat{x}.v : \beta_i, \forall i \in \{1, \dots, n\}$, using Generation lemma 4.16(iii) it follows that $\Gamma, x : \gamma \vdash v : \beta_i, \forall i \in \{1, \dots, n\}$. Without losing generality we can assume that $x \notin Fv(k')$ (if the variable x was free in k' we would have to rename it in k where it is bound; then we would not have the variable x, but some other variable). According to Proposition 4.19 we can extend the basis in the second premise to $\Gamma, x : \gamma; \cap_i^n \beta_i \vdash k' : \alpha$. Then,

$$\frac{\dfrac{\Gamma, x : \gamma \vdash v : \beta_1 \;\cdots\; \Gamma, x : \gamma \vdash v : \beta_n \quad \Gamma, x : \gamma; \cap_i^n \beta_i \vdash k' : \alpha}{\Gamma, x : \gamma \vdash vk' : \alpha}\,(Cut)}{\Gamma; \gamma \vdash \widehat{x}.vk' : \alpha}\,(Sel)$$

  Since $(\widehat{x}.v)@k' \triangleq \widehat{x}.vk'$, the proof is complete.

- $k \equiv v :: k''$:
  From $\Gamma; C \vdash v :: k'' : \beta_i, \forall i \in \{1, \dots, n\}$, using Generation lemma 4.16(v), it follows that $\gamma \equiv \cap_j^m \delta_j \to \varepsilon$, $\Gamma; \varepsilon \vdash k'' : \beta_i, \forall i \in \{1, \dots, n\}$, and $\Gamma \vdash v : \delta_j, \forall j \in \{1, \dots, m\}$. Applying the induction hypothesis to k'' and k' we get $\Gamma; \varepsilon \vdash k''@k' : \alpha$. Now we can build the following derivation:

$$\frac{\Gamma \vdash v : \delta_1 \;\cdots\; \Gamma \vdash v : \delta_m \quad \Gamma; \varepsilon \vdash k''@k' : \alpha}{\Gamma; \cap_j^m \delta_j \to \varepsilon \vdash v :: (k''@k') : \alpha.}\,(\to_L)$$

  Since $\cap_j^m \delta_j \to \varepsilon \equiv \gamma$ and $(v :: k'')@k' \triangleq v :: (k''@k')$, the proof is complete. □

Now, we can prove that the type of a λ^{Gtz}-expression in the system $\lambda^{\mathsf{Gtz}}\cap$ does not change during reduction.

Theorem 4.24 (Subject Reduction for $\lambda^{\mathsf{Gtz}}\cap$)

(i) If $\Gamma \vdash t : \alpha$ *and* $t \to t'$, *then* $\Gamma \vdash t' : \alpha$.

(ii) If $\Gamma;\beta \vdash k : \alpha$ *and* $k \to k'$, *then* $\Gamma;\beta \vdash k' : \alpha$.

Proof: Both cases are proved simultaneously by induction on the last applied reduction. We distinguish four cases:

- Case (β):
  Suppose that $\Gamma \vdash (\lambda x.t)(u :: k) : \alpha$. We need to show that $\Gamma \vdash u(\widehat{x}.tk) : \alpha$.
  From $\Gamma \vdash (\lambda x.t)(u :: k) : \alpha$, using Generation lemma 4.16(iv), it follows that there exists a type $\cap_i^n \beta_i$ such that $\Gamma \vdash \lambda x.t : \beta_i, \forall i \in \{1,\dots,n\}$ and $\Gamma;\cap_i^n \beta_i \vdash u :: k : \alpha$. Using Lemma 4.14 and Generation lemma 4.16(v) for the second premise we deduce that $\cap_i^n \beta_i \equiv \cap_i^n(\cap_j^m \gamma_j \to \delta_i) \sim \cap_j^m \gamma_j \to \cap_i^n \delta_i$, $\Gamma;\cap_i^n \delta_i \vdash k : \alpha$, and $\Gamma \vdash u : \gamma_j, \forall j \in \{1,\dots,m\}$. On the other hand, from $\Gamma \vdash \lambda x.t : \cap_j^m \gamma_j \to \delta_i, \forall i \in \{1,\dots,n\}$, using Generation lemma 4.16(ii), it follows that $\Gamma, x : \cap_j^m \gamma_j \vdash t : \delta_i, \forall i \in \{1,\dots,n\}$. From here we conclude that $x \notin Dom(\Gamma)$. Now we can write a type derivation for the term $u(\widehat{x}.tk)$:

$$\dfrac{\Gamma \vdash u : \gamma_1 \;\cdots\; \Gamma \vdash u : \gamma_m \qquad \dfrac{\dfrac{\Gamma, x : \cap_j^m \gamma_j \vdash t : \delta_1 \;\cdots\; \Gamma, x : \cap_j^m \gamma_j \vdash t : \delta_n \quad \Gamma, x : \cap_j^m \gamma_j;\cap_i^n \delta_i \vdash k : \alpha}{\Gamma, x : \cap_j^m \gamma_j \vdash tk : \alpha}\,(Cut)}{\Gamma;\cap_j^m \gamma_j \vdash \widehat{x}.tk : \alpha}\,(Sel)}{\Gamma \vdash u(\widehat{x}.tk) : \alpha.}\,(Cut)$$

- Case (π):
  Suppose that $\Gamma \vdash (tk)k' : \alpha$. We have to show that $\Gamma \vdash t(k@k') : \alpha$.
  From $\Gamma \vdash (tk)k' : \alpha$, using Generation lemma 4.16(iv), it follows that there exists a type $\cap_i^n \beta_i$ such that $\Gamma \vdash tk : \beta_i$, for all $i \in \{1,\dots,n\}$ and $\Gamma;\cap_i^n \beta_i \vdash k' : \alpha$. Next, using Generation lemma 4.16(iv) for the first premise we conclude that for each $i \in \{1,\dots,n\}$ there exists a type $\cap_j^m \gamma_j$ such that $\Gamma \vdash t : \gamma_j$, for all $j \in \{1,\dots,m\}$ and $\Gamma;\cap_j^m \gamma_j \vdash k : \beta_i$, for all $i \in \{1,\dots,n\}$. From $\Gamma;\cap_j^m \gamma_j \vdash k : \beta_i$, for all $i \in \{1,\dots,n\}$ and $\Gamma;\cap_i^n \beta_i \vdash k' : \alpha$, applying Proposition 4.23 we get $\Gamma;\cap_j^m \gamma_j \vdash k@k' : \alpha$. So we can conclude the following:

$$\dfrac{\Gamma \vdash t : \gamma_1 \cdots \Gamma \vdash t : \gamma_m \quad \Gamma;\cap_j^m \gamma_j \vdash k@k' : \alpha}{\Gamma \vdash t(k@k') : \alpha}\,(Cut)$$

- Case (σ):
  Suppose that $\Gamma \vdash t(\widehat{x}.v) : \alpha$. We have to show that $\Gamma \vdash v[t/x] : \alpha$.
  From $\Gamma \vdash t(\widehat{x}.v) : \alpha$, using Generation lemma 4.16(iv), it follows that there exists a type $\cap_i^n \beta_i$ such that $\Gamma \vdash t : \beta_i$, for all $i \in \{1,\dots,n\}$ and $\Gamma; \cap_i^n \beta_i \vdash \widehat{x}.v : \alpha$. Next, using Generation lemma 4.16(iii) for the second premise we derive that $\Gamma, x : \cap_i^n \beta_i \vdash v : \alpha$. Now we can apply Substitution lemma 4.22 and get $\Gamma \vdash v[t/x] : \alpha$.

- Case (μ):
  Suppose that $\Gamma; \beta \vdash \widehat{x}.xk : \alpha$. We have to show that $\Gamma; \beta \vdash k : \alpha$. Using Generation lemma 4.16(iii) it follows that $\Gamma, x : \beta \vdash xk : \alpha$. Next, using Generation lemma 4.16(iv) there exists a type $\cap_i^n \gamma_i$ such that $\Gamma, x : \beta \vdash x : \gamma_i$, for all $i \in \{1,\dots,n\}$ and $\Gamma, x : \beta; \cap_i^n \gamma_i \vdash k : \alpha$. From the first sequent, using Generation lemma 4.16(i), it follows that $\beta \sim \cap_i^n \gamma_i$. Since $x \notin Fv(k)$, the proof is complete using Proposition 4.19. $\Box$

Example 4.25 *In the λ-calculus, the term $\lambda x.xx$ has the type $(\alpha \cap (\alpha \to \beta)) \to \beta$. The corresponding term in the λ^{Gtz}-calculus is $\lambda x.x(x :: \widehat{y}.y)$. Although being a normal form this term is not typeable in the simply typed λ^{Gtz}-calculus. It is typeable in $\lambda^{\mathsf{Gtz}}\cap$ in the following way:*

$$
\dfrac{
\dfrac{}{x : \alpha \cap (\alpha \to \beta) \vdash x : \alpha \to \beta}\,(Ax)
\qquad
\dfrac{
\dfrac{}{x : \alpha \cap (\alpha \to \beta) \vdash x : \alpha}\,(Ax)
\quad
\dfrac{\dfrac{}{x : \alpha \cap (\alpha \to \beta), y : \beta \vdash y : \beta}\,(Ax)}{x : \alpha \cap (\alpha \to \beta); \beta \vdash \widehat{y}.y : \beta}\,(Sel)
}{x : \alpha \cap (\alpha \to \beta); \alpha \to \beta \vdash x :: \widehat{y}.y : \beta}\,(\to_L)
}{
\dfrac{x : \alpha \cap (\alpha \to \beta) \vdash x(x :: \widehat{y}.y) : \beta}{\vdash \lambda x.x(x :: \widehat{y}.y) : (\alpha \cap (\alpha \to \beta)) \to \beta}\,(\to_R).
}\,(Cut)
$$

4.2.3 The systems leading to $\lambda^{\mathsf{Gtz}}\cap$

The construction of the appropriate intersection type assignment system for the λ^{Gtz}-calculus was not a straightforward process. Two unsuccessful (but useful) attempts, presented in the following subsections, led to the system $\lambda^{\mathsf{Gtz}}\cap$.

First attempt: Intuitive system $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{I}}$

Our first (and the most natural) attempt consisted of simply adding standard typing rules for intersection operator to the existing Espírito Santo's basic type assignment system $\lambda^{\mathsf{Gtz}} \to$ following the characteristic symmetry of the sequent calculus.

The type assignment system $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{I}}$ is given in Figure 4.6.

$$\frac{}{\Gamma, x:\alpha \vdash x:\alpha}\ (Ax)$$

$$\frac{\Gamma, x:\alpha \vdash t:\beta}{\Gamma \vdash \lambda x.t:\alpha \to \beta}\ (\to_R) \qquad \frac{\Gamma \vdash t:\alpha \quad \Gamma;\beta \vdash k:\gamma}{\Gamma;\alpha \to \beta \vdash t::k:\gamma}\ (\to_L)$$

$$\frac{\Gamma \vdash t:\alpha \quad \Gamma;\alpha \vdash k:\beta}{\Gamma \vdash tk:\beta}\ (Cut) \qquad \frac{\Gamma, x:\alpha \vdash t:\beta}{\Gamma;\alpha \vdash \widehat{x}.t:\beta}\ (Sel)$$

$$\frac{\Gamma \vdash t:\alpha \quad \Gamma \vdash t:\beta}{\Gamma \vdash t:\alpha \cap \beta}\ (\cap_R) \qquad \frac{\Gamma, x:\alpha_1 \vdash t:\beta}{\Gamma, x:\alpha_1 \cap \alpha_2 \vdash t:\beta}\ (\cap_L)$$

$$\frac{\Gamma \vdash t:\alpha, \ \ \alpha \leq \beta}{\Gamma \vdash t:\beta}\ (\leq_R)$$

Figure 4.6: First attempt: intuitive system $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{I}}$

Basis expansion and Bases intersection lemmas can be easily proved for the proposed system, where bases intersection is defined as usual.

The following rules are admissible in $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{I}}$.

Proposition 4.26 ($\leq$ rules)

(i) If $\Gamma, x:\alpha \vdash t:\gamma$ *and* $\beta \leq \alpha$*, then* $\Gamma, x:\beta \vdash t:\gamma$.
If $\Gamma, x:\alpha;\gamma \vdash k:\delta$ *and* $\beta \leq \alpha$*, then* $\Gamma, x:\beta;\gamma \vdash k:\delta$.

(ii) If $\Gamma;\gamma \vdash k:\alpha$ *and* $\alpha \leq \beta$*, then* $\Gamma;\gamma \vdash k:\beta$.

Proposition 4.27 ($\cap$ rules)

(i) If $\Gamma \vdash t:\alpha_1 \cap \alpha_2$*, then* $\Gamma \vdash t:\alpha_i$*, for each* $i \in \{1,2\}$.

(ii) If $\Gamma;\alpha_1 \vdash \widehat{x}.t:\beta$*, then* $\Gamma;\alpha_1 \cap \alpha_2 \vdash \widehat{x}.t:\beta$.

(iii) If $\Gamma;\gamma \vdash k:\alpha$ *and* $\Gamma;\gamma \vdash k:\beta$*, then* $\Gamma;\gamma \vdash k:\alpha \cap \beta$.

This system has two problems. The first one is that the second statement from Proposition 4.27 holds only for the selection, while it is not possible to prove a similar statement for the context of the form $k \equiv t :: k_1$ (since type changes are not allowed in the stoup in any of the typing rules). The second one is that in the presence of $(\cap_R)$ rule all terms could have intersection types. These problems make it impossible to formulate the Generation lemma which would enable us to "reverse" the rules of the type assignment system and which is usually necessary for proving the Subject reduction and Subject expansion properties. Hence, the main tool for further proofs was missing and forced us to search for a new intersection type assignment system for the λ^{Gtz}-calculus.

Second attempt: Restrictive system $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{R}}$

Having realized that the above presented system is inappropriate, mainly because it allows too much due to the overly permissive typing rules, we turned to a more restrictive approach and designed a system inspired by the type assignment system for classical sequent $\lambda\mu\tilde{\mu}$-calculus proposed by Dougherty et al. in [19]. In this system pre-order $\leq$ on types is completely omitted, as well as RHS introduction of intersection, which turned out to be the problematic rule in the previous system $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{I}}$ (indeed only LHS intersection introduction is important in the system, whereas RHS intersection introduction was only added for symmetry reasons). To regain the broken symmetry of the system, we replaced LHS intersection introduction with upgraded rules (Ax) and $(\to_L)$, in which intersection is implicitly introduced. This system assigns types to the same set of terms as the previous one, but it is more restrictive since the set of types that can be assigned to a certain term is smaller. For example, in the previous system the type of the abstraction could be both $\alpha \cap \beta$ and $\alpha \to \beta$, whereas in this one it can only be $\alpha \to \beta$. The system, denoted by $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{R}}$, is given in Figure 4.7, where $\cap\alpha_i$ abbreviates $\cap_{i=1}^{n}\alpha_i$, for some $n \geq 1$.

$$\frac{}{\Gamma, x : \cap_i^n \alpha_i \vdash x : \alpha_j}\ (Ax)$$

$$\frac{\Gamma, x : \alpha \vdash t : \beta}{\Gamma \vdash \lambda x.t : \alpha \to \beta}\ (\to_R) \qquad \frac{\Gamma \vdash t : \alpha_1 \cdots \Gamma \vdash t : \alpha_n \quad \Gamma; \beta \vdash k : \gamma}{\Gamma; \cap_i^n \alpha_i \to \beta \vdash t :: k : \gamma}\ (\to_L)$$

$$\frac{\Gamma \vdash t : \alpha \quad \Gamma; \alpha \vdash k : \beta}{\Gamma \vdash tk : \beta}\ (Cut) \qquad \frac{\Gamma, x : \alpha \vdash t : \beta}{\Gamma; \alpha \vdash \widehat{x}.t : \beta}\ (Sel)$$

Figure 4.7: Second attempt: restrictive system $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{R}}$

Basis expansion and Bases intersection lemmas hold for this system as well and the rule $(\cap_L)$ from the previous system is now admissible. Since there is exactly one rule for deriving each sequent the system is syntax-directed so it is trivial to formulate and prove the following Generation lemma.

Proposition 4.28 (Generation lemma for $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{R}}$)

(i) $\Gamma \vdash x : \alpha$ *iff* $x : \alpha \cap \alpha_1 \ldots \cap \alpha_n \in \Gamma$ *for some* $n \geq 0$.

(ii) $\Gamma \vdash \lambda x.t : \alpha$ *iff* $\alpha \equiv \beta \to \gamma$ *and* $\Gamma, x : \beta \vdash t : \gamma$.

(iii) $\Gamma; \alpha \vdash \widehat{x}.t : \beta$ *iff* $\Gamma, x : \alpha \vdash t : \beta$.

(iv) $\Gamma \vdash tk : \alpha$ *iff there exists a type* β *such that* $\Gamma \vdash t : \beta$ *and* $\Gamma; \beta \vdash k : \alpha$.

(v) $\Gamma; \alpha \vdash t :: k : \beta$ *iff* $\alpha \equiv \cap_i^n \gamma_i \to \delta$ *and* $\Gamma; \delta \vdash k : \beta$ *and* $\Gamma \vdash t : \gamma_i$, *for all* $i \in \{1, \ldots, n\}$.

The basic properties we wanted to prove were Subject reduction and Subject expansion. We proved Substitution lemma, analogous to the one from λ-calculus and the following Append lemma.

Lemma 4.29 (Append lemma for $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{R}}$) *If* $\Gamma; \gamma \vdash k : \beta$ *and* $\Gamma; \beta \vdash k' : \alpha$*, then* $\Gamma; \gamma \vdash k@k' : \alpha$.

However, when trying to prove Subject reduction for $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{R}}$, we were stuck already with the first reduction rule (β). Supposing that $\Gamma \vdash (\lambda x.t)(u :: k) : \alpha$, we wanted to prove that $\Gamma \vdash u\widehat{x}.tk : \alpha$. From $\Gamma \vdash (\lambda x.t)(u :: k) : \alpha$ and using Generation lemma 4.28 (iv) it follows that there exists a type β such that $\Gamma \vdash \lambda x.t : \beta$ and $\Gamma; \beta \vdash u :: k : \alpha$. From the last sequent, using Generation lemma 4.28 (v) it follows that $\beta = \cap_i^n \gamma_i \to \delta$, $\Gamma; \delta \vdash k : \alpha$ and $\Gamma \vdash u : \gamma_i$ for all $i \in \{1, \ldots, n\}$. From $\Gamma \vdash \lambda x.t : \beta$, using Generation lemma 4.28 (ii) it follows that $\Gamma, x : \cap_i^n \gamma_i \vdash t : \delta$. Now we had to assign a type to term $u\widehat{x}.tk$:

$$\dfrac{\Gamma \vdash u : \gamma_1 \cdots \Gamma \vdash u : \gamma_n \qquad \dfrac{\dfrac{\Gamma, x : \cap_i^n \gamma_i \vdash t : \delta \quad \Gamma, x : \cap_i^n \gamma_i; \delta \vdash k : \alpha}{\Gamma, x : \cap_i^n \gamma_i \vdash tk : \alpha}\,(Cut)}{\Gamma; \cap_i^n \gamma_i \vdash \widehat{x}.tk : \alpha}\,(Sel)}{???}\,(Cut)$$

The last (Cut) rule is impossible to apply, since the types γ_i and $\cap \gamma_i$ do not match.

There are two solutions to this problem: we can either change the (β) reduction rule or we can again change the type system. The first solution can be achieved by replacing the (β) reduction rule with a larger computational step - $(\beta + \sigma)$ reduction rule as follows:

$$(\lambda x.t)(u :: k) \ \to_\beta \ t[u/x]k.$$

With this reduction rule, it is possible to prove Subject reduction for the rules $(\beta), (\sigma)$ and (π) without changing the type system. The (μ) reduction is of a different nature and for this reduction it is possible to prove the following proposition.

Proposition 4.30 *If* $\Gamma; \cap_i^n \beta_i \vdash \widehat{x}.xk : \alpha$, *then* $\Gamma; \beta_j \vdash k : \alpha$, *for some* $j \in \{1, ..., n\}$.

But such a modification implies losing the possibility to delay substitution and the call-by-value computational side of λ^{Gtz}. Also, Subject expansion property (needed for characterisation of strong normalisation) does not hold.

Hence, in order to obtain type assignment system which characterises all strongly normalising terms, we had to change the type assignment system again. For more details about the system $\lambda^{\mathsf{Gtz}}\cap_{\mathsf{R}}$ see [31].

The appropriate type assignment system, presented in Subsection 4.2.2, in which Subject reduction and Subject expansion at the root position hold for the original reductions of the λ^{Gtz}-calculus, was introduced in Espírito Santo et al. [24]. In order to assign the same type to β-redex $(\lambda x.t)(u :: k)$ and its contractum $u\widehat{x}.tk$ (as required by Subject reduction) we needed to implicitly introduce intersection in the (Cut) rule. The necessity for certain equivalencies among types showed up, so we returned $\leq$ relation. But $\leq$ relation is not explicitly introduced into typing rules, its only role is in defining equivalence, so that the equivalent types can be interchangeable in derivations. The important role belongs to the following equivalence: $\cap_i^n(\alpha \to \beta_i) \sim \alpha \to \cap_i^n \beta_i$.

With this system we finally succeeded in characterising strong normalisation in the λ^{Gtz}-calculus, which will be elaborated in the following section.

4.3 Characterisation of SN in the λ^{Gtz}-calculus

4.3.1 Typeability $\Rightarrow$ SN

In this subsection, we will prove that all expressions typeable in the $\lambda^{\mathsf{Gtz}}\cap$ system satisfy the strong normalisation. The basic idea of the proof is to establish a connection between the λ^{Gtz}-calculus with intersection types and the typed λ-calculus with intersection types via an appropriate mapping that preserves types, and then to use the strong normalisation result from [60] for λ-terms typeable in the system $\mathcal{D}$, presented in Section 3.1.

The target calculus for the embedding is the λ-calculus enriched with two more reduction rules besides the regular (β) reduction. These two reductions, namely (π_1) and (π_2), are called permutations:

$$\begin{array}{lrcl} (\pi_1) & (\lambda x.M)NP & \to & (\lambda x.MP)N \\ (\pi_2) & M((\lambda x.P)N) & \to & (\lambda x.MP)N, \end{array}$$

We will also use the notation $\pi = \pi_1 \cup \pi_2$.

Espírito Santo proved in [26, 22] that these new reductions do not essentially change the calculus, more precisely they do not change the set of the strongly normalising λ-terms. The proof partly relies on Regnier's result from [59] since the permutation (π_1) is actually proposed by Regnier under the name σ.

Proposition 4.31 ([26]) *π-reduction is terminating (finite) in λ-calculus.*

Proposition 4.32 ([22]) *If λ-term M is β-SN, then it is also $\beta\pi$-SN.*

Now we define a mapping $\lfloor\ \rfloor$ from the λ^{Gtz}-calculus to the λ-calculus. We use Λ to denote the set of λ-terms, Λ^{Gtz} for the set of λ^{Gtz}-terms and Λ_C^{Gtz} for the set of λ^{Gtz}-contexts. In order to make a clear distinction between λ^{Gtz}-terms and λ-terms, we denote the latter ones with the capital letters $M, N, \ldots$

Definition 4.33 *Mapping $\lfloor\ \rfloor : \Lambda^{\mathsf{Gtz}} \to \Lambda$ is defined together with the auxiliary mapping $\lfloor\ \rfloor_{\mathrm{k}} : \Lambda_C^{\mathsf{Gtz}} \to (\Lambda \to \Lambda)$ in the following way:*

$$\begin{array}{rcl}
\lfloor x \rfloor & = & x \\
\lfloor \lambda x.t \rfloor & = & \lambda x.\lfloor t \rfloor \\
\lfloor tk \rfloor & = & \lfloor k \rfloor_{\mathrm{k}}(\lfloor t \rfloor) \\
\\
\lfloor \widehat{x}.t \rfloor_{\mathrm{k}}(M) & = & (\lambda x.\lfloor t \rfloor)M \\
\lfloor t :: k \rfloor_{\mathrm{k}}(M) & = & \lfloor k \rfloor_{\mathrm{k}}(M\lfloor t \rfloor)
\end{array}$$

Example 4.34 *Let $t \equiv x(y :: z :: \widehat{u}.u)$. Then*

$$\begin{array}{rcl}
\lfloor t \rfloor & = & \lfloor x(y :: z :: \widehat{u}.u) \rfloor \\
& = & \lfloor y :: z :: \widehat{u}.u \rfloor_{\mathrm{k}}(x) \\
& = & \lfloor z :: \widehat{u}.u \rfloor_{\mathrm{k}}(xy) \\
& = & \lfloor \widehat{u}.u \rfloor_{\mathrm{k}}(xyz) \\
& = & (\lambda u.u)(xyz).
\end{array}$$

The following proposition shows that the introduced mapping preserves the type of a term. The plain notation $\Gamma \vdash t : \alpha$ will be used for type assignments in the system $\lambda^{\mathsf{Gtz}}\cap$, while $\Gamma \vdash_{\mathcal{D}} M : \alpha$ will be used for type assignments in the system $\mathcal{D}$.

Proposition 4.35

(i) If $\Gamma \vdash t : \alpha$, then $\Gamma \vdash_{\mathcal{D}} \lfloor t \rfloor : \alpha$.

(ii) If $\Gamma; \alpha \vdash k : \beta$ and $\Gamma \vdash_{\mathcal{D}} M : \alpha$, then $\Gamma \vdash_{\mathcal{D}} \lfloor k \rfloor_{\mathrm{k}}(M) : \beta$.

Proof: By simultaneous induction on derivations of $\Gamma \vdash t : \alpha$ and $\Gamma; \alpha \vdash k : \beta$. We distinguish the following cases according to the last typing rule applied:

- Case (Ax) is obtained by the corresponding (Ax) rule in $\mathcal{D}$ together with the rule $(\cap E)$, applied a number of times.

- Case $(\to R)$ is easy, because $\mathcal{D}$ has the corresponding typing rule.

- Case (Cut). Derivation ends with

$$\frac{\Gamma \vdash t : \alpha_1 \cdots \Gamma \vdash t : \alpha_n \quad \Gamma; \cap_i^n \alpha_i \vdash k : \beta}{\Gamma \vdash tk : \beta}\ (Cut).$$

  By applying the IH to the first premise, we have $\Gamma \vdash_{\mathcal{D}} \lfloor t \rfloor : \alpha_i$, for all $i \in \{1, ..., n\}$. By repeated application of $(\cap I)$, we get $\Gamma \vdash_{\mathcal{D}} \lfloor t \rfloor : \cap_i^n \alpha_i$. Now we can apply the IH to the second premise, yielding $\Gamma \vdash_{\mathcal{D}} \lfloor k \rfloor_{\mathrm{k}}(\lfloor t \rfloor) : B$. This is what we want, since $\lfloor k \rfloor_{\mathrm{k}}(\lfloor t \rfloor) = \lfloor tk \rfloor$.

- Case (Sel). Derivation ends with

$$\frac{\Gamma, x : \alpha \vdash t : \beta}{\Gamma; \alpha \vdash \widehat{x}.t : \beta}\ (Sel).$$

  By the IH, we have that $\Gamma, x : \alpha \vdash_{\mathcal{D}} \lfloor t \rfloor : \beta$. Let $M \in \Lambda$ and let $\Gamma \vdash_{\mathcal{D}} M : \alpha$. Then in $\mathcal{D}$ we have

$$\frac{\dfrac{\Gamma, x : \alpha \vdash_{\mathcal{D}} \lfloor t \rfloor : \beta}{\Gamma \vdash_{\mathcal{D}} \lambda x. \lfloor t \rfloor : \alpha \to \beta}\ (\to I) \qquad \Gamma \vdash_{\mathcal{D}} M : \alpha}{\Gamma \vdash_{\mathcal{D}} (\lambda x. \lfloor t \rfloor) M : \beta}\ (\to E).$$

  This is what we want, since $\lfloor \widehat{x}.t \rfloor_{\mathrm{k}}(M) = (\lambda x. \lfloor t \rfloor) M$.

- Case $(\to L)$. Derivation ends with

$$\frac{\Gamma \vdash t : \alpha_1 \cdots \Gamma \vdash t : \alpha_n \qquad \Gamma; \beta \vdash k : \gamma}{\Gamma; \cap_i^n \alpha_i \to \beta \vdash t :: k : \gamma}\ (\to L).$$

  By the IH we have $\Gamma \vdash_{\mathcal{D}} \lfloor t \rfloor : \alpha_i$, for all $i \in \{1, ..., n\}$. Therefore, by repeated application of $(\cap I)$, we get $\Gamma \vdash_{\mathcal{D}} \lfloor t \rfloor : \cap_i^n \alpha_i$. Now, take some $M \in \Lambda$ such that $\Gamma \vdash_{\mathcal{D}} M : \cap_i^n \alpha_i \to \beta$. Then in $\mathcal{D}$ we have

$$\frac{\Gamma \vdash_{\mathcal{D}} M : \cap_i^n \alpha_i \to \beta \quad \Gamma \vdash_{\mathcal{D}} \lfloor t \rfloor : \cap_i^n \alpha_i}{\Gamma \vdash_{\mathcal{D}} M\lfloor t \rfloor : \beta}\ (\to E).$$

Now, from $\Gamma;\beta \vdash k : \gamma$ and $\Gamma \vdash_{\mathcal{D}} M\lfloor t \rfloor : \beta$, by the IH we get $\Gamma \vdash_{\mathcal{D}} \lfloor k \rfloor_{\mathrm{k}}(M\lfloor t \rfloor) : \gamma$. This completes the proof, since $\lfloor k \rfloor_{\mathrm{k}}(M\lfloor t \rfloor) = \lfloor t :: k \rfloor_{\mathrm{k}}(M)$. $\square$

The next step towards the proof of the termination of the system $\lambda^{\mathsf{Gtz}}\cap$ is to prove that the reductions of the λ^{Gtz}-calculus could be simulated by the reductions of the target calculus, namely β, π_1 and π_2 reductions. In order to do so, we need some auxiliary propositions. We use the notation $\to$ for the reductions in the λ^{Gtz}-calculus, and the notation $\to_\lambda$ for the reductions of the λ-calculus enriched with permutation reductions.

Lemma 4.36 $\lfloor k \rfloor_{\mathrm{k}}((\lambda x.P)N) \to_\lambda (\lambda x.\lfloor k \rfloor_{\mathrm{k}}(P))N$.

Proof: By induction on the structure of the context k.

- Base case $k \equiv \widehat{y}.t$.

$$\lfloor k \rfloor_{\mathrm{k}}((\lambda x.P)N) \equiv \lfloor \widehat{y}.t \rfloor_{\mathrm{k}}((\lambda x.P)N) = (\lambda y.\lfloor t \rfloor)((\lambda x.P)N) \to_{\pi_2}$$
$$(\lambda x.(\lambda y.\lfloor t \rfloor)P)N = (\lambda x.\lfloor \widehat{y}.t \rfloor_{\mathrm{k}}(P))N \equiv (\lambda x.\lfloor k \rfloor_{\mathrm{k}}(P))N.$$

- Case $k \equiv t :: k'$.

$$\lfloor k \rfloor_{\mathrm{k}}((\lambda x.P)N) \equiv \lfloor t :: k' \rfloor_{\mathrm{k}}((\lambda x.P)N) = \lfloor k' \rfloor_{\mathrm{k}}((\lambda x.P)N\lfloor t \rfloor) \to_{\pi_1}$$
$$\lfloor k' \rfloor_{\mathrm{k}}((\lambda x.P\lfloor t \rfloor)N) \to_{IH} (\lambda x.\lfloor k' \rfloor_{\mathrm{k}}(P\lfloor t \rfloor))N = (\lambda x.\lfloor t :: k' \rfloor_{\mathrm{k}}(P))N \equiv$$
$$(\lambda x.\lfloor k \rfloor_{\mathrm{k}}(P))N.\square$$

In the following lemma, $\lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor k \rfloor_{\mathrm{k}}$ denotes the composition of two functions.

Lemma 4.37 *If $M \in \Lambda$ and $k, k' \in \Lambda_C^{\mathsf{Gtz}}$, then* $\lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor k \rfloor_{\mathrm{k}}(M) \to_\lambda \lfloor k@k' \rfloor_{\mathrm{k}}(M)$.

Proof: By induction on the structure of the context k.

- Base case $k \equiv \widehat{y}.t$.

$$\lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor k \rfloor_{\mathrm{k}}(M) \equiv \lfloor k' \rfloor_{\mathrm{k}}(\lfloor \widehat{y}.t \rfloor_{\mathrm{k}}(M)) = \lfloor k' \rfloor_{\mathrm{k}}((\lambda y.\lfloor t \rfloor)M).$$

On the other hand

$$\lfloor k@k' \rfloor_{\mathrm{k}}(M) \equiv \lfloor \widehat{y}.t@k' \rfloor_{\mathrm{k}}(M) = \lfloor \widehat{y}.tk' \rfloor_{\mathrm{k}}(M) = (\lambda y.\lfloor tk' \rfloor)M =$$
$$(\lambda y.\lfloor k' \rfloor_{\mathrm{k}}(\lfloor t \rfloor))M.$$

Now, the proposition holds by lemma 4.36.

- Case $k \equiv t :: k''$.

$$\lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor k \rfloor_{\mathrm{k}}(M) \equiv \lfloor k' \rfloor_{\mathrm{k}}(\lfloor t :: k'' \rfloor_{\mathrm{k}}(M)) = \lfloor k' \rfloor_{\mathrm{k}}(\lfloor k'' \rfloor_{\mathrm{k}}(M\lfloor t \rfloor) \to_{IH}$$
$$\lfloor k''@k' \rfloor_{\mathrm{k}}(M\lfloor t \rfloor) = \lfloor t :: k''@k' \rfloor_{\mathrm{k}}(M) = \lfloor (t :: k'')@k' \rfloor_{\mathrm{k}}(M) \equiv \lfloor k@k' \rfloor_{\mathrm{k}}(M).\square$$

Lemma 4.38

a) $\lfloor t[v/x] \rfloor = \lfloor t \rfloor[\lfloor v \rfloor/x]$.

b) $\lfloor k[v/x] \rfloor_{\mathrm{k}}(M) = \lfloor k \rfloor_{\mathrm{k}}[\lfloor v \rfloor/x](M)$.

Proof: By mutual induction on the structure of t and k, using the definition of the substitution.$\square$

Lemma 4.39 *If $x \notin Fv(k)$, then $(\lfloor k \rfloor_{\mathrm{k}}(M))[N/x] = \lfloor k \rfloor_{\mathrm{k}}(M[N/x])$.*

Proof: By induction on the structure of the context k.

- Base case $k \equiv \widehat{y}.t$, where $x \notin Fv(t)$ and $x \neq y$.

$$(\lfloor k \rfloor_{\mathrm{k}}(M))[N/x] \equiv (\lfloor \widehat{y}.t \rfloor_{\mathrm{k}}(M))[N/x] = ((\lambda y.\lfloor t \rfloor)M)[N/x] =$$
$$(\lambda y.\lfloor t \rfloor)M[N/x] = \lfloor \widehat{y}.t \rfloor_{\mathrm{k}}(M[N/x]) \equiv \lfloor k \rfloor_{\mathrm{k}}(M[N/x]).$$

- Case $k \equiv t :: k'$, where $x \notin Fv(t) \cup Fv(k')$.

$$(\lfloor k \rfloor_{\mathrm{k}}(M))[N/x] \equiv (\lfloor t :: k' \rfloor_{\mathrm{k}}(M))[N/x] = (\lfloor k' \rfloor_{\mathrm{k}}(M\lfloor t \rfloor))[N/x] =_{IH}$$
$$\lfloor k' \rfloor_{\mathrm{k}}((M\lfloor t \rfloor)[N/x]) = \lfloor k' \rfloor_{\mathrm{k}}((M[N/x]\lfloor t \rfloor)) = \lfloor t :: k' \rfloor_{\mathrm{k}}(M[N/x]) \equiv$$
$$\lfloor k \rfloor_{\mathrm{k}}(M[N/x]).\square$$

Now, we can prove the proposition stating that λ^{Gtz} reductions can be simulated by λ-reductions.

Proposition 4.40 (Simulation of reductions)

(i) If a term $t \to t'$, then $\lfloor t \rfloor \to_{\lambda} \lfloor t' \rfloor$.

(ii) If a context $k \to k'$, then $\lfloor k \rfloor_{\mathrm{k}}(M) \to_{\lambda} \lfloor k' \rfloor_{\mathrm{k}}(M)$, for any $M \in \Lambda$.

Proof: By case analysis on the applied reduction. Without losing generality, we prove the statement only for the outermost reductions.

(β) $t \equiv (\lambda x.v)(u :: k) \to u(\widehat{x}.vk) \equiv t'$.

On the one hand we have

$\lfloor t \rfloor \equiv \lfloor (\lambda x.v)(u :: k) \rfloor = \lfloor u :: k \rfloor_{\mathrm{k}}(\lfloor \lambda x.v \rfloor) = \lfloor k \rfloor_{\mathrm{k}}((\lambda x.\lfloor v \rfloor)\lfloor u \rfloor)$

On the other hand,

$\lfloor t' \rfloor \equiv \lfloor u(\widehat{x}.vk) \rfloor = \lfloor \widehat{x}.vk \rfloor_{\mathrm{k}}(\lfloor u \rfloor) = (\lambda x.\lfloor vk \rfloor)\lfloor u \rfloor = (\lambda x.\lfloor k \rfloor_{\mathrm{k}}(\lfloor v \rfloor))\lfloor u \rfloor$.

Thus $\lfloor t \rfloor \to_{\lambda} \lfloor t' \rfloor$ by Lemma 4.36.

(π) $t \equiv (uk)k' \to u(k@k') \equiv t'$.

On the one hand we have

$\lfloor t \rfloor \equiv \lfloor (uk)k' \rfloor = \lfloor k' \rfloor_{\mathrm{k}}(\lfloor uk \rfloor) = \lfloor k' \rfloor_{\mathrm{k}}(\lfloor k \rfloor_{\mathrm{k}}(\lfloor u \rfloor))$

On the other hand,

$\lfloor t' \rfloor \equiv \lfloor u(k@k') \rfloor = \lfloor k@k' \rfloor_{\mathrm{k}}(\lfloor u \rfloor)$.

Applying Lemma 4.37 we get that $\lfloor t \rfloor \to_{\lambda} \lfloor t' \rfloor$.

(σ) $t \equiv u(\widehat{x}.v) \to v[u/x] \equiv t'$.

On the one hand we have

$\lfloor t \rfloor \equiv \lfloor u(\widehat{x}.v) \rfloor = \lfloor \widehat{x}.v \rfloor_{\mathrm{k}}(\lfloor u \rfloor) = (\lambda x.\lfloor v \rfloor)\lfloor u \rfloor$.

On the other hand, applying Lemma 4.38, we have that

$\lfloor t' \rfloor \equiv \lfloor v[u/x] \rfloor = \lfloor v \rfloor[\lfloor u \rfloor/x]$.

Therefore, $t \to_{\lambda} t'$ by (β) reduction in the λ-calculus.

(μ) $\widehat{x}.xk \to k$.

This reduction reduces context to context, so for an arbitrary $M \in \Lambda$:

$\lfloor \widehat{x}.xk \rfloor_{\mathrm{k}}(M) = (\lambda x.\lfloor xk \rfloor)M = (\lambda x.\lfloor k \rfloor_{\mathrm{k}}(x))M$.

Now, this reduces by (β) reduction to $(\lfloor k \rfloor_{\mathrm{k}}(x))[M/x]$ in λ. Since we know that $x \notin Fv(k)$, which is a side condition for the (μ) reduction, we can apply Lemma 4.39 and get $(\lfloor k \rfloor_{\mathrm{k}}(x))[M/x] = \lfloor k \rfloor_{\mathrm{k}}(x[M/x]) = \lfloor k \rfloor_{\mathrm{k}}(M)$. Thus $\lfloor \widehat{x}.xk \rfloor_{\mathrm{k}}(M) \to_{\lambda} \lfloor k \rfloor_{\mathrm{k}}(M)$ by (β) reduction followed by Lemma 4.39. □

Theorem 4.41 (Typeability ⇒ SN in $\lambda^{\mathsf{Gtz}}\cap$) *If a λ^{Gtz}-term t is typeable in the system $\lambda^{\mathsf{Gtz}}\cap$, then t is SN.*

Proof: Suppose that t is typeable in $\lambda^{\mathsf{Gtz}}\cap$, but not SN. By Proposition 4.35, $\lfloor t \rfloor$ is typeable in $\mathcal{D}$. By Proposition 4.40, each λ^{Gtz}-reduction of an infinite chain starting with t can be simulated by a λ-reduction, meaning that there exists an infinite chain of reductions starting with $\lfloor t \rfloor$. Since $\lfloor t \rfloor$ is typeable, this is in contradiction with the SN of the system $\mathcal{D}$ (Propositions 3.7 and 4.32). Thus, t must be SN. □

4.3.2 SN $\Rightarrow$ Typeability

In order to prove that all strongly normalising λ^{Gtz}-expressions are typeable, we start with proving the typeability of normal forms. Since in the λ^{Gtz}-calculus β, π and σ reductions eliminate cuts, $\beta\pi\sigma$-normal forms are those expression that do not contain any application, i.e. cut, but the trivial one. The set of these normal forms can be described in the following way:

$$\begin{array}{lrcl} \text{(Terms)} & t_{nf} & = & x \mid \lambda x.t_{nf} \mid x(t_{nf} :: k_{nf}) \\ \text{(Contexts)} & k_{nf} & = & \widehat{x}.t_{nf} \mid t_{nf} :: k_{nf} \end{array}$$

Notice that the previous taxonomy of normal forms does not exactly correspond to the set of $\beta\pi\sigma\mu$-normal forms. For example, $\widehat{x}.x(t_{nf} :: k_{nf})$ belongs to the set of $\beta\pi\sigma$-normal forms, but can be reduced by the reduction μ to k_{nf}, if $x \notin k_{nf}$. However, since the set of $\beta\pi\sigma\mu$-normal forms is a subset of the set of $\beta\pi\sigma$-normal forms, we can prove the following proposition.

Proposition 4.42 $\beta\pi\sigma$*-normal forms of the* λ^{Gtz}*-calculus are typeable in the* $\lambda^{\mathsf{Gtz}}\cap$ *system. Hence so are* $\beta\pi\sigma\mu$*-normal forms.*

Proof: By simultaneous induction on the structure of $\beta\pi\sigma$-normal terms and contexts.

- Basic case: Every variable is typeable.
- $\lambda x.t_{nf}$ is typeable.
  By the IH, t_{nf} is typeable, so $\Gamma \vdash t_{nf} : \beta$. We examine two cases:

Case 1. If $x : \alpha \in \Gamma$, then $\Gamma = \Gamma', x : \alpha$ and we can assign the following type to $\lambda x.t_{nf}$:

$$\frac{\Gamma', x : \alpha \vdash t_{nf} : \beta}{\Gamma' \vdash \lambda x.t_{nf} : \alpha \to \beta.}\ (\to_R)$$

Case 2. If $x : \alpha \notin \Gamma$, then by Proposition 4.19 we get $\Gamma, x : \alpha \vdash t_{nf} : \beta$ thus concluding

$$\dfrac{\Gamma, x : \alpha \vdash t_{nf} : \beta}{\Gamma \vdash \lambda x.t_{nf} : \alpha \to \beta.}\ (\to_R)$$

- $\widehat{x}.t_{nf}$ is typeable.
  Proof is similar to the previous one.

- $t_{nf} :: k_{nf}$ is typeable.
  By the IH t_{nf} and k_{nf} are typeable, i.e. $\Gamma_1 \vdash t_{nf} : \alpha$ and $\Gamma_2; \beta \vdash k_{nf} : \gamma$. Then, by Proposition 4.21 we get $\Gamma_1 \cap \Gamma_2 \vdash t_{nf} : \alpha$ and $\Gamma_1 \cap \Gamma_2; \beta \vdash k_{nf} : \gamma$, so we assign the following type to $t_{nf} :: k_{nf}$:

$$\dfrac{\Gamma_1 \cap \Gamma_2 \vdash t_{nf} : \alpha \quad \Gamma_1 \cap \Gamma_2; \beta \vdash k_{nf} : \gamma}{\Gamma_1 \cap \Gamma_2; \alpha \to \beta \vdash t_{nf} :: k_{nf} : \gamma.}\ (\to_L)$$

- $x(t_{nf} :: k_{nf})$ is typeable.
  By the IH and the previous case, the context $t_{nf} :: k_{nf}$ is typeable, i.e. $\Gamma; \alpha \to \beta \vdash t_{nf} :: k_{nf} : \gamma$. We examine three cases:

Case 1. If $x : \alpha \to \beta \in \Gamma$, then:

$$\dfrac{\dfrac{}{\Gamma \vdash x : \alpha \to \beta}\ (Ax) \qquad \Gamma; \alpha \to \beta \vdash t_{nf} :: k_{nf} : \gamma}{\Gamma \vdash x(t_{nf} :: k_{nf}) : \gamma.}\ (Cut)$$

Case 2. If $x : \delta \in \Gamma$, then $\Gamma = \Gamma', x : \delta$ and we can expand the basis of $x : \alpha \to \beta \vdash x : \alpha \to \beta$ to $\Gamma', x : \delta \cap (\alpha \to \beta) \vdash x : \alpha \to \beta$ using Propositions 4.17 and 4.19. Also, by Proposition 4.17, we have $\Gamma', x : \delta \cap (\alpha \to \beta); \alpha \to \beta \vdash t_{nf} :: k_{nf} : \gamma$. Now, the corresponding type assignment is:

$$\dfrac{\Gamma', x : \delta \cap (\alpha \to \beta) \vdash x : \alpha \to \beta \qquad \Gamma', x : \delta \cap (\alpha \to \beta); \alpha \to \beta \vdash t_{nf} :: k_{nf} : \gamma}{\Gamma', x : \delta \cap (\alpha \to \beta) \vdash x(t_{nf} :: k_{nf}) : \gamma.}\ (Cut)$$

Case 3. If $x \notin Dom(\Gamma)$, by Proposition 4.19 we get $\Gamma, x : \alpha \to \beta; \alpha \to \beta \vdash t_{nf} :: k_{nf} : \gamma$ from $\Gamma; \alpha \to \beta \vdash t_{nf} :: k_{nf} : \gamma$, and then conclude:

$$\dfrac{\dfrac{}{\Gamma, x : \alpha \to \beta \vdash x : \alpha \to \beta}\ (Ax) \qquad \Gamma, x : \alpha \to \beta; \alpha \to \beta \vdash t_{nf} :: k_{nf} : \gamma}{\Gamma, x : \alpha \to \beta \vdash x(t_{nf} :: k_{nf}) : \gamma.}\ (Cut)$$

□

The next goal is to prove that the type of a λ^{Gtz}-expression does not change during the expansion - a process reverse to reduction. In order to achieve that goal, we first treat the cases of particular expansions.

Lemma 4.43 (β-expansion for $\lambda^{\mathsf{Gtz}}\cap$) *If* $\Gamma \vdash u(\widehat{x}.tk) : \alpha$ *and* $x \notin Fv(u) \cup Fv(k)$, *then* $\Gamma \vdash (\lambda x.t)(u :: k) : \alpha$.

Proof: $\Gamma \vdash u(\widehat{x}.tk) : \alpha$ implies, by Generation lemma 4.16(iv), that there is a type $\beta \equiv \cap_i^n \beta_i$, such that $\Gamma \vdash u : \beta_i$, for all $i \in \{1,...,n\}$ and $\Gamma; \cap_i^n \beta_i \vdash \widehat{x}.(tk) : \alpha$. Further, this implies, by Generation lemma 4.16(iii), that $\Gamma, x : \cap_i^n \beta_i \vdash tk : \alpha$ so applying again Generation lemma 4.16(iv) we obtain that there is a type $\gamma \equiv \cap_j^m \gamma_j$ such that $\Gamma, x : \cap_i^n \beta_i \vdash t : \gamma_j$ for all $j \in \{1,...,m\}$ and $\Gamma, x : \cap_i^n \beta_i; \cap_j^m \gamma_j \vdash k : \alpha$. By assumption, the variable x is not free in k, so using Proposition 4.19 we can write the previous sequent as $\Gamma; \cap_j^m \gamma_j \vdash k : \alpha$. Now, because of the equivalence $\cap_j^m (\cap_i^n \beta_i \to \gamma_j) \sim \cap_i^n \beta_i \to \cap_j^m \gamma_j$, we have[5]:

$$\dfrac{\dfrac{\Gamma, x : \cap_i^n \beta_i \vdash t : \gamma_j,\ \forall j}{\Gamma \vdash \lambda x.t : \cap_i^n \beta_i \to \gamma_j,\ \forall j}\,(\to_R) \qquad \dfrac{\Gamma \vdash u : \beta_i,\ \forall i \quad \Gamma; \cap_j^m \gamma_j \vdash k : \alpha}{\Gamma; \cap_i^n \beta_i \to \cap_j^m \gamma_j \vdash u :: k : \alpha}\,(\to_L)}{\Gamma \vdash (\lambda x.t)(u :: k) : \alpha.}\,(Cut)$$

□

The following two lemmas treat the expansions containing the meta-operators of the λ^{Gtz}-calculus.

Lemma 4.44 (Inverse substitution lemma for $\lambda^{\mathsf{Gtz}}\cap$)

(i) Let $\Gamma \vdash v[t/x] : \alpha$, *and let t be typeable. Then there is a basis* Γ' *and a type* $\beta \equiv \cap_i^n \beta_i$, *such that* $\Gamma', x : \cap_i^n \beta_i \vdash v : \alpha$ *and* $\Gamma' \vdash t : \beta_i$, *for all* $i \in \{1,...,n\}$.

(ii) Let $\Gamma; \gamma \vdash k[t/x] : \alpha$, *and let t be typeable. Then there is a basis* Γ' *and a type* $\beta \equiv \cap_i^n \beta_i$, *such that* $\Gamma', x : \cap_i^n \beta_i; \gamma \vdash k : \alpha$ *and* $\Gamma' \vdash t : \beta_i$, *for all* $i \in \{1,...,n\}$.

Proof: By simultaneous induction on the structure of the term v and the context k.

- The base cases:

[5]We are aware that the usage of quantifiers is not completely correct in the type derivations, but due to the lack of space we were forced to choose a compact form over the completely correct one.

– $v \equiv x$.
Then $v[t/x] = x[t/x] \triangleq t$. From the first premise we have $\Gamma \vdash t : \alpha$, whereas from the assumption that t is typeable we have $\Gamma^* \vdash t : \gamma$. Since $x \notin Fv(t)$, by the Proposition 4.19 it follows that $x \notin \Gamma$ and $x \notin \Gamma^*$. Now, for $\Gamma' \equiv \Gamma \cap \Gamma^*$ and $\beta = \alpha \cap \gamma$ by (Ax) we have $\Gamma', x : \alpha \cap \gamma \vdash x : \alpha$, and by the bases intersection (Proposition 4.21) we get $\Gamma' \vdash t : \alpha$ and $\Gamma' \vdash t : \gamma$.

– $v \equiv y$.
Then $v[t/x] = y[t/x] \triangleq y$, therefore $\Gamma \vdash y : \alpha$. From the assumption that t is typeable we have $\Gamma^* \vdash t : \beta$. Since $x \notin Fv(t)$, $x \notin \Gamma^*$ and $x \notin \Gamma$. Now, for $\Gamma' \equiv \Gamma \cap \Gamma^*$ and by the Propositions 4.19 and 4.21 we have $\Gamma', x : \beta \vdash y : \alpha$ and $\Gamma' \vdash t : \beta$.

- $v \equiv \lambda y.v'$.
$v[t/x] = (\lambda y.v')[t/x] \triangleq \lambda y.v'[t/x]$. From $\Gamma \vdash \lambda y.v'[t/x] : \alpha$, by Generation lemma 4.16(ii) it follows that $\alpha \equiv \gamma \to \delta$ and $\Gamma, y : \gamma \vdash v'[t/x] : \delta$. Applying the IH to v' we conclude that there exist Γ' and $\cap_j^m \beta_j$ such that $\Gamma', x : \cap_j^m \beta_j, y : \gamma \vdash v' : \delta$ and $\Gamma, y : \gamma \vdash t : \beta_j$ for all $j \in \{1, ..., m\}$. Now, we conclude:

$$\frac{\Gamma', x : \cap_j^m \beta_j, y : \gamma \vdash v' : \delta}{\Gamma', x : \cap_j^m \beta_j \vdash \lambda y.v' : \gamma \to \delta} \ (\to_R).$$

- $k \equiv \widehat{y}.t'$.
Then $k[t/x] = (\widehat{y}.t')[t/x] \triangleq \widehat{y}.t'[t/x]$. From $\Gamma; \gamma \vdash \widehat{y}.t'[t/x] : \alpha$, by Generation lemma 4.16(iii) it follows that $\Gamma, y : \gamma \vdash t'[t/x] : \alpha$. Applying the IH to t' we obtain Γ' and $\cap_i^n \beta_i$ such that $\Gamma', x : \cap_i^n \beta_i \vdash t' : A$ and $\Gamma' \vdash t : \beta_i$, for all $i \in \{1, ..., n\}$. Since $y \notin Fv(t)$, $y \notin \Gamma'$, by the Proposition 4.19 we get:

$$\frac{\Gamma', x : \cap_i^n \beta_i, y : \gamma \vdash t' : \alpha}{\Gamma', x : \cap_i^n \beta_i; \gamma \vdash \widehat{y}.t' : \alpha.} \ (Sel)$$

- $v \equiv t'k$.
Then $v[t/x] = (t'k)[t/x] \triangleq (t'[t/x])(k[t/x])$. From the premise
$\Gamma \vdash (t'[t/x])(k[t/x]) : \alpha$ and by Generation lemma 4.16(iv) we have that there is a type $\cap_i^n \gamma_i$ such that $\Gamma \vdash t'[t/x] : \gamma_i$ for all $i \in \{1, ..., n\}$, and $\Gamma; \cap_i^n \gamma_i \vdash k[t/x] : \alpha$. Applying the IH to t' we conclude that there exist Γ_1 and $\cap_j^m \beta'_j$ such that $\Gamma_1 \vdash t : \beta'_j$ for all $j \in \{1, ..., m\}$ and $\Gamma_1, x : \cap_j^m B'_j \vdash t' : \gamma_i$ for all $i \in \{1, ..., n\}$. Further, applying the IH to k we conclude that there exist Γ_2 and $\cap_k^p \beta''_k$ such that $\Gamma_2 \vdash t : \beta''_k$ for all $k \in \{1, ..., p\}$, and $\Gamma_2, x : \cap_k^p \beta''_k; \cap_i^n \gamma_i \vdash k : \alpha$. Now, for $\Gamma' \equiv \Gamma_1 \cap \Gamma_2$ and $\cap_l^q \beta_l \equiv (\cap_j^m \beta'_j) \cap (\cap_k^p \beta''_k)$ we have that $\Gamma' \vdash t : \beta_l$

for all $l \in \{1,...,q\}$, and

$$\frac{\Gamma',x:\cap_l^q\beta_l \vdash t':\gamma_1 \cdots \Gamma',x:\cap_l^q\beta_l \vdash t':\gamma_n \quad \Gamma_1,x:\cap_l^q\beta_l;\cap_i^n\gamma_i \vdash k:\alpha}{\Gamma',x:\cap_l^q\beta_l \vdash t'k:\alpha,}\,(Cut)$$

using the Propositions 4.19 and 4.17.

- $k \equiv t' :: k'$.
Then $k[t/x] \triangleq (t'[t/x]) :: (k'[t/x])$, hence from $\Gamma;\gamma \vdash t'[t/x] :: k'[t/x] : \alpha$, by Generation lemma 4.16(v), it follows that $\gamma \equiv \cap_i^n\delta_i \to \varepsilon$, $\Gamma \vdash t'[t/x] : \delta_i$, for all $i \in \{1,...,n\}$ and $\Gamma;\varepsilon \vdash k'[t/x] : \alpha$. Applying the IH to both sequents we conclude that there exist Γ', Γ'', $\cap_j^m\beta'_j$ and $\cap_k^p\beta''_k$ such that $\Gamma' \vdash t : \beta'_j$ for all $j \in \{1,...,m\}$, $\Gamma',x:\cap_j^m\beta'_j \vdash t' : \delta$, $\Gamma'' \vdash t : \beta''_k$ for all $k \in \{1,...,p\}$, and $\Gamma'',x:\cap_k^p\beta''_k;\varepsilon \vdash k' : \alpha$. For $\Gamma_1 \equiv \Gamma' \cap \Gamma''$ and $\cap_l^q\beta_l \equiv \cap_j^m\beta'_j \cap (\cap_k^p\beta''_k)$ we conclude that $\Gamma_1 \vdash t : \beta_l$ for all $l \in \{1,...,q\}$ and

$$\frac{\Gamma_1,x:\cap_l^q\beta_l \vdash t':\delta_1 \cdots \Gamma_1,x:\cap_l^q\beta_l \vdash t':\delta_n \quad \Gamma_1,x:\cap_l^q\beta_l;\varepsilon \vdash k':\alpha}{\Gamma_1,x:\cap_l^q\beta_l;\cap_i^n\delta_i \to \varepsilon \vdash t' :: k' : \alpha}\,(\to_L)$$

which completes the proof since $\cap_i^n\delta_i \to \varepsilon \equiv \gamma$. □

Lemma 4.45 (Inverse append lemma for $\lambda^{\mathsf{Gtz}}\cap$) *If $\Gamma;\beta \vdash k@k' : \alpha$ then there is a type $\gamma \equiv \cap_i^n\gamma_i$ such that $\Gamma;\beta \vdash k : \gamma_i$ for all $i \in \{1,...,n\}$, and $\Gamma;\cap_i^n\gamma_i \vdash k' : \alpha$.*

Proof: By induction on the structure of k.

- Basic case: $k \equiv \widehat{x}.v$.
In this case $k@k' = (\widehat{x}.v)@k' = \widehat{x}.vk'$. From $\Gamma;\beta \vdash \widehat{x}.vk' : \alpha$, by Generation lemma 4.16(iii), we have that $\Gamma,x:\beta \vdash vk' : \alpha$. Then, by Generation lemma 4.16(iv), there is a type $\gamma \equiv \cap_i^n\gamma_i$ such that $\Gamma,x:\beta \vdash v : \gamma_i$, for all $i \in \{1,...,n\}$, and $\Gamma,x:\beta;\cap_i^n\gamma_i \vdash k' : \alpha$. From the first sequent we get $\Gamma;\beta \vdash \widehat{x}.v : \gamma_i$, for all $i \in \{1,...,n\}$. From the second one, considering that x is not free in k', we get $\Gamma;\cap_i^n\gamma_i \vdash k' : \alpha$.

- $k \equiv u :: k''$.
In this case, $k@k' = (u :: k'')@k' = u :: (k''@k')$. From $\Gamma;\beta \vdash u :: (k''@k') : \alpha$, by Generation lemma 4.16(v), $\beta \equiv \cap_i^n\gamma_i \to \delta$, $\Gamma;\delta \vdash k''@k' : \alpha$ and $\Gamma \vdash u : \gamma_i$, for all $i \in \{1,...,n\}$. From the first sequent, by the IH, we get some $\varepsilon \equiv \cap_j^m\varepsilon_j$

such that $\Gamma;\delta \vdash k'' : \varepsilon_j$, for all $j \in \{1,...,m\}$, and $\Gamma;\cap_j^m\varepsilon_j \vdash k' : \alpha$. Finally, for each $j \in \{1,...,m\}$, we have

$$\dfrac{\Gamma \vdash u : \gamma_1 \cdots \Gamma \vdash u : \gamma_n \quad \Gamma;\delta \vdash k'' : \varepsilon_j}{\Gamma;\cap_i^n\gamma_i \to \delta(\equiv \beta) \vdash u :: k'' : \varepsilon_j}\,(\to_L)$$

so the proof is completed. □

Proposition 4.46 (Subject expansion at head position for $\lambda^{\mathsf{Gtz}}\cap$) *Let $t \to t'$, and let t be the contracted redex. If t' is typeable in $\lambda^{\mathsf{Gtz}}\cap$, then t is typeable in $\lambda^{\mathsf{Gtz}}\cap$.*

Proof: We examine four different cases, according to the last applied reduction.

- (β) : Directly follows from Lemma 4.43.
- (σ) : We show that typeability of $t' \equiv v[u/x]$ leads to typeability of $t \equiv u\widehat{x}.v$. Assume that $\Gamma \vdash v[u/x] : \alpha$. By Lemma 4.44 there exist Γ' and $\beta \equiv \cap_i^n\beta_i$ such that $\Gamma' \vdash u : \beta_i$, for all $i \in \{1,...,n\}$ and $\Gamma', x : \cap_i^n\beta_i \vdash v : \alpha$. Now

$$\dfrac{\Gamma' \vdash u : \beta_1 \cdots \Gamma' \vdash u : \beta_n \quad \dfrac{\Gamma', x : \cap_i^n\beta_i \vdash v : \alpha}{\Gamma';\cap_i^n\beta_i \vdash \widehat{x}.v : \alpha}\,(Sel)}{\Gamma' \vdash u\widehat{x}.v : \alpha.}\,(Cut)$$

- (π) : We show that typeability of $t(k@k')$ implies typeability of $(tk)k'$. From $\Gamma \vdash t(k@k') : \alpha$, by Generation lemma 4.16(iv) we have that there exists $\beta \equiv \cap_i^n\beta_i$ such that $\Gamma \vdash t : \beta_i$, for all $i \in \{1,...,n\}$, and $\Gamma;\cap_i^n\beta_i \vdash k@k' : \alpha$. By applying Lemma 4.45 to the previous sequent,
we get $\Gamma;\cap_i^n\beta_i \vdash k : \gamma_j$, for all $j \in \{1,...,m\}$, and $\Gamma;\cap_j^m\gamma_j \vdash k' : \alpha$, for some type $\gamma \equiv \cap_j^m\gamma_j$. Now, for each $j \in \{1,...,m\}$, we have

$$\dfrac{\Gamma \vdash t : \beta_1 \cdots \Gamma \vdash t : \beta_n \quad \Gamma;\cap_i^n\beta_i \vdash k : \gamma_j}{\Gamma \vdash tk : \gamma_j}\,(Cut)$$

So $\Gamma \vdash tk : \gamma_j$, for all $j \in \{1,...,m\}$. We obtain $\Gamma \vdash (tk)k' : \alpha$ with one more application of (Cut) rule.

- (μ) : It should be shown that typeability of k implies typeability of $\widehat{x}.xk$. Assume $\Gamma;\beta \vdash k : \alpha$. Since $x \notin k$ we can suppose that $x \notin Dom(\Gamma)$, and by using Proposition 4.19 write $\Gamma, x : \beta;\beta \vdash k : \alpha$. Now

$$\dfrac{\dfrac{\Gamma, x : \beta \vdash x : \beta \quad \Gamma, x : \beta;\beta \vdash k : \alpha}{\Gamma, x : \beta \vdash xk : \alpha}\,(Cut)}{\Gamma;\beta \vdash \widehat{x}.xk : \alpha.}\,(Sel)$$

□

Theorem 4.47 (SN $\Rightarrow$ typeability) *All strongly normalising λ^{Gtz}-expressions are typeable in the system $\lambda^{\mathsf{Gtz}}\cap$.*

Proof: The proof is by induction on the length of the longest reduction path out of a strongly normalising expression e, with a subinduction on the size of e.

If e is a $\beta\sigma\pi\mu$-normal form, then it is typeable by Proposition 4.42.

If e is itself a redex, let e' be the expression obtained by contracting the redex e. Therefore e' is strongly normalising and by the IH it is typeable. Then e is typeable, by Proposition 4.46.

Next suppose that e is not itself a redex nor a normal form. Then e is of one of the following forms: $\lambda x.u$, $x(u :: k)$, $u :: k$, or $\widehat{x}.u$ (in each case with u or k *not* $\beta\pi\sigma$-normal). Each of the above u and k is strongly normalising, hence typeable by the IH, as the subexpressions of e. It is easy then to build the typing of e, as in the proof of Proposition 4.42. □

Finally, we can give a full characterisation of strong normalisation in the λ^{Gtz}-calculus.

Corollary 4.48 *A λ^{Gtz}-term is strongly normalising if and only if it is typeable in $\lambda^{\mathsf{Gtz}}\cap$.*

Proof: By Theorems 4.41 and 4.47. □

4.4 Regaining confluence in the λ^{Gtz}-calculus

In spite of many desirable features, the λ^{Gtz}-calculus is not confluent, i.e. does not satisfy Church-Rosser property, meaning that the choice of the computational strategy can transform a term into different normal forms. As already stated in Section 4.1, where the problem is illustrated by Example 4.3, the source of non-confluence is the presence of a critical pair of reductions, namely π and σ.

In this section, we propose two calculi derived from the λ^{Gtz}-calculus that do enjoy the confluence property. The goal is to eliminate the critical pair, and we achieve it by restricting some reduction rules of the λ^{Gtz}-calculus. These restrictions are made by modifications of the syntax. We also prove that the proposed calculi are confluent using the modification of the parallel reductions technique.

4.4.1 Two confluent sub-calculi

We start by looking back at Example 4.3. If we forbid the σ reduction to be performed on the term $(tk)(\widehat{x}.v)$, we would obtain a "call-by-value" sub-calculus, denoted by λ_V^{Gtz}. The abstract syntax of the λ_V^{Gtz}-calculus is the following:

$$\begin{array}{lrcl} \text{Values} & T & ::= & x \,|\, \lambda x.t \\ \text{Terms} & t & ::= & T \,|\, tk \\ \text{Contexts} & k & ::= & \widehat{x}.t \,|\, t :: k \end{array}$$

So, as in [14], we introduce *values* as a new syntactic category. Reduction rules of the λ_V^{Gtz}-calculus are β, π, μ of the λ^{Gtz}-calculus and

$$(\sigma_V) \quad T(\widehat{x}.v) \quad \rightarrow \quad v[T/x].$$

This reduction system is forcing us to reduce the head of the cut to the value before substituting it instead of x in v, which is exactly the essence of the call-by-value computational strategy.

On the other hand, if we forbid π reduction to be performed on the term $(tk)(\widehat{x}.v)$, we would obtain another confluent sub-calculus, denoted by λ_L^{Gtz}. The abstract syntax of the λ_L^{Gtz}-calculus is the following:

$$\begin{array}{lrcl} \text{Terms} & t & ::= & x \,|\, \lambda x.t \,|\, tk \\ \text{Lists} & l & ::= & \widehat{x}.x \,|\, t :: l \\ \text{Contexts} & k & ::= & l \,|\, \widehat{x}.t \end{array}$$

Here, we have to introduce the syntactic category of *lists*, that are a subset of the set of contexts and whose form is $t_1 :: t_2 :: ... :: t_k :: \widehat{x}.x$. The trivial selection, $\widehat{x}.x$, actually represents an empty list [ ], so by applying this convention we can write the above list in the form $[t_1, t_2, ..., t_k]$. Reduction rules of the λ_L^{Gtz}-calculus are β, σ, μ of the λ^{Gtz}-calculus and

$$(\pi_L) \quad (tk)l \quad \rightarrow \quad t(k@l).$$

In this reduction system, only the term of the form $(tk)(\widehat{x}.x)$ is at the same time a σ-redex and a (π_L)-redex, but applying each of these two reductions leads to the same result, namely tk, so the confluence is not broken.

4.4.2 The proof of confluence

After the elimination of the critical pair, we obtained two λ^{Gtz}-subcalculi, namely the λ_V^{Gtz}-calculus and the λ_L^{Gtz}-calculus. We will now prove the confluence of the λ_V^{Gtz}-calculus, whereas the proof of confluence for the λ_L^{Gtz}-calculus will be skipped because it is analogous.

Definition 4.49 (Confluence, Church-Rosser property) *A reduction R is said to be confluent if its reflexive and transitive closure $\twoheadrightarrow_R$ satisfies the so-called diamond property, i.e. if for all terms t, t_1, t_2 the following holds:*

$$\textit{if } t_1 \twoheadleftarrow t \twoheadrightarrow t_2, \textit{ then there exists } t' \textit{ such that } t_1 \twoheadrightarrow t' \twoheadleftarrow t_2.$$

The technique we use for proving this property is the parallel reductions technique, developed by Takahashi in [68] and adapted by Likavec in [53] for proving the Church-Rosser property of the $\lambda\mu\tilde{\mu}$ sub-calculi. This approach is based on simultaneous reduction of all existing redexes in a term. In the sequel, $\rightarrow$ will denote the union of all four λ_V^{Gtz} reductions and $\twoheadrightarrow$ will denote its reflexive and transitive closure.

First, we need to introduce the notion of parallel reductions for λ_V^{Gtz}-calculus, denoted by $\Rightarrow$ and defined inductively as follows:

Definition 4.50 (Parallel reductions for λ_V^{Gtz}-calculus)

$$\frac{}{x \Rightarrow x}\ (g1) \qquad \frac{t \Rightarrow t'}{\lambda x.t \Rightarrow \lambda x.t'}\ (g2) \qquad \frac{t \Rightarrow t',\ k \Rightarrow k'}{tk \Rightarrow t'k'}\ (g3)$$

$$\frac{t \Rightarrow t'}{\widehat{x}.t \Rightarrow \widehat{x}.t'}\ (g4) \qquad \frac{t \Rightarrow t',\ k \Rightarrow k'}{t :: k \Rightarrow t' :: k'}\ (g5)$$

$$\frac{t \Rightarrow t',\ u \Rightarrow u',\ k \Rightarrow k'}{(\lambda x.t)(u :: k) \Rightarrow u'\widehat{x}.(t'k')}\ (g6) \qquad \frac{T \Rightarrow T',\ t \Rightarrow t'}{T(\widehat{x}.t) \Rightarrow t'[T'/x]}\ (g7)$$

$$\frac{t \Rightarrow t',\ k \Rightarrow k',\ k_1 \Rightarrow k_1'}{(tk)k_1 \Rightarrow t'(k' @ k_1')}\ (g8) \qquad \frac{k \Rightarrow k'}{\widehat{x}.xk \Rightarrow k'}\ (g9)$$

Lemma 4.51 (Reflexivity) *For every expression e, $e \Rightarrow e$.*

Proof: By induction on the structure of e. The basic case is covered by the rule $(g1)$ from definition 4.50. In all other cases, we apply the IH on subexpressions of e and rules $(g2)$ - $(g5)$. $\square$

Lemma 4.52 (Substitution) *If $x \neq y$ and $x \notin Fv(v_2)$, then*

$$e[v_1/x][v_2/y] = e[v_2/y][(v_1[v_2/y])/x].$$

Proof: By induction on the structure of e. The basic case is when e is a variable. There are three possibilities:

- $e \equiv x$.
  Then, $x[v_1/x][v_2/y] = v_1[v_2/y]$ and $x[v_2/y][(v_1[v_2/y])/x]$
  $= x[(v_1[v_2/y])/x] = v_1[v_2/y]$.

- $e \equiv y$.
  Then, $y[v_1/x][v_2/y] = y[v_2/y] = v_2$ and $y[v_2/y][(v_1[v_2/y])/x]$
  $= v_2[(v_1[v_2/y])/x] = v_2$, because $x \notin Fv(v_2)$.

- $e \equiv z$, such that $z \neq x$ and $z \neq y$.
  Then both sides are equal to z.

In all other cases, we apply the IH to the subexpressions of e. □

Now, we give the definition of *holes*.

Definition 4.53 (Holes)

$$\begin{array}{rcl} \mathcal{H}_t & ::= & [\,] \mid \lambda x.\mathcal{H}_t \mid t\mathcal{H}_c \mid \mathcal{H}_t k \\ \mathcal{H}_c & ::= & \widehat{x}.\mathcal{H}_t \mid t :: \mathcal{H}_c \mid \mathcal{H}_t :: k \end{array}$$

We will write $\mathcal{H}$ instead of $\mathcal{H}_t \cup \mathcal{H}_c$. $\mathcal{H}[e]$ denotes filling the hole in C with an expression e. Now, we can prove the following statement, in which we will also use $h, h', \ldots$ to denote expressions.

Lemma 4.54

(i) If $e \to e'$, then $e \Rightarrow e'$.

(ii) If $e \Rightarrow e'$, then $e \twoheadrightarrow e'$.

(iii) If $e \Rightarrow e'$ and $h \Rightarrow h'$, then $e[h/x] \Rightarrow e'[h'/x]$.

Proof:

(i) By induction on the kind of a hole in a redex. If $e \to e'$, then $e = \mathcal{H}[h]$, $e' = \mathcal{H}[h']$ and $h \to h'$.

- The basic case is $\mathcal{H} \equiv [\,]$.
  There are four possible cases of reductions in $h \to h'$.
  - β-reduction - then $h \equiv (\lambda x.t)(u :: k)$ and $h' \equiv u(\widehat{x}.tk)$. By Lemma 4.51 $t \Rightarrow t$, $u \Rightarrow u$ and $k \Rightarrow k$, so from the rule $(g6)$ of Definition 4.50 $h \Rightarrow h'$, hence $e \Rightarrow e'$.

- σ-reduction - then $h \equiv T\widehat{x}.v$ and $h' \equiv v[T/x]$. By Lemma 4.51 $T \Rightarrow T$ and $v \Rightarrow v$, so from the rule $(g7)$ we have that $h \Rightarrow h'$.
- π-reduction - then $h \equiv (tk)k'$ and $h' \equiv tk@k'$. By Lemma 4.51 $t \Rightarrow t$, $k \Rightarrow k$ and $k' \Rightarrow k'$, so from the rule $(g8)$ $h \Rightarrow h'$.
- μ-reduction - then $h \equiv \widehat{x}.xk$ and $h' \equiv k$. By Lemma 4.51 $k \Rightarrow k$, so from the rule $(g9)$ $h \Rightarrow h'$.

- $\mathcal{H} \equiv \lambda x.\mathcal{H}'$.
  Then $e \equiv \lambda x.\mathcal{H}'[h]$ and $e' \equiv \lambda x.\mathcal{H}'[h']$. By the IH we have $\mathcal{H}'[h] \Rightarrow \mathcal{H}'[h']$, so from the rule $(g2)$ of Definition 4.50 we get $e \Rightarrow e'$.

- $\mathcal{H} \equiv t\ \mathcal{H}'$.
  Then $e \equiv t\ \mathcal{H}'[h]$ and $e' \equiv t\ \mathcal{H}'[h']$. By the IH we have $\mathcal{H}'[h] \Rightarrow \mathcal{H}'[h']$, from Lemma 4.51 we have $t \Rightarrow t$ so from the rule $(g3)$ of Definition 4.50 we get $e \Rightarrow e'$.
  The proof is similar for the remaining hole kinds.

(ii) By induction on the definition of the parallel reduction. We show several cases, the others are proved similarly.

- Basic case is $e \equiv x \Rightarrow x \equiv e'$.
  In that case, $e \equiv x \twoheadrightarrow x \equiv e'$ is trivially satisfied.
- $e \equiv tk \Rightarrow t'k' \equiv e'$.
  This is the direct consequence of the premises $t \Rightarrow t'$ and $k \Rightarrow k'$. By the IH, $t \twoheadrightarrow t'$ and $k \twoheadrightarrow k'$, hence

$$e \equiv tk \twoheadrightarrow t'k' \equiv e'.$$

- $e \equiv (\lambda x.t)(u :: k) \Rightarrow u'(\widehat{x}.t'k') \equiv e'$.
  This is the direct consequence of the premises $t \Rightarrow t'$, $u \Rightarrow u'$ and $k \Rightarrow k'$. By the IH, $t \twoheadrightarrow t'$, $u \twoheadrightarrow u'$ and $k \twoheadrightarrow k'$, hence

$$e \equiv (\lambda x.t)(u :: k) \to u(\widehat{x}.tk) \twoheadrightarrow u'(\widehat{x}.t'k') \equiv e'.$$

(iii) By induction on the definition of the parallel reduction.

- The first basic case is $e \equiv x \Rightarrow x \equiv e'$.
  Then $e[h/x] \equiv x[h/x] = h \Rightarrow h' = x[h/x'] \equiv e'[h/x']$, which is given in assumptions.

- The second basic case is $e \equiv y \Rightarrow y \equiv e'$, $y \neq x$.
  Then $e[h/x] \equiv y[h/x] = y \Rightarrow y = y[h/x'] \equiv e'[h/x']$, by rule $(g1)$ of Definition 4.50.
- $e \equiv \lambda y.t \Rightarrow \lambda y.t' \equiv e'$.
  This follows from the premise $t \Rightarrow t'$, so by applying the IH we get $t[h/x] \Rightarrow t'[h'/x]$. From this, by the rule $(g2)$ and the substitution definition we obtain

$$e[h/x] \equiv \lambda y.t[h/x] \Rightarrow \lambda y.t'[h/x'] \equiv e'[h/x'].$$

- $e \equiv tk \Rightarrow t'k' \equiv e'$.
  Premises of this statement are $t \Rightarrow t'$ and $k \Rightarrow k'$. Applying the IH to both of them we get $t[h/x] \Rightarrow t'[h/x']$ and $k[h/x] \Rightarrow k'[h/x']$. Now we derive

$$\begin{array}{rcl} e[h/x] & \equiv & (tk)[h/x] = t[h/x]k[h/x] \\ & \Rightarrow & t'[h/x']k'[h/x'] = (t'k')[h/x'] \\ & \equiv & e'[h/x'], \end{array}$$

  using the substitution definition and the rule $(g3)$ of Definition 4.50.
- $e \equiv \widehat{y}.t \Rightarrow \widehat{y}.t' \equiv e'$.
  The direct premise of the statement is $t \Rightarrow t'$, so applying the IH to it we get $t[h/x] \Rightarrow t'[h/x']$. From this, by the rule $(g4)$ and the substitution definition it follows that

$$e[h/x] \equiv \widehat{y}.t[h/x] \Rightarrow \widehat{y}.t'[h'/x] \equiv e'[h'/x].$$

- $e \equiv t :: k \Rightarrow t' :: k' \equiv e'$.
  The statement follows from the premises $t \Rightarrow t'$ and $k \Rightarrow k'$. By applying the IH to both of them we get $t[h/x] \Rightarrow t'[h'/x]$ and $k[h/x] \Rightarrow k'[h'/x]$, yielding

$$\begin{array}{rcl} e[h/x] & \equiv & (t :: k)[h/x] \\ & = & t[h/x] :: k[h/x] \\ & \Rightarrow & t'[h'/x] :: k'[h'/x] \\ & = & (t' :: k')[h'/x] \\ & \equiv & e'[h'/x], \end{array}$$

  by the substitution definition and the rule $(g5)$ of Definition 4.50.
- $e \equiv (\lambda y.t)(u :: k) \Rightarrow u'\widehat{y}.(t'k') \equiv e'$.
  Direct premises of this statement are $t \Rightarrow t'$, $u \Rightarrow u'$ and $k \Rightarrow k'$. From the

IH we have $t[h/x] \Rightarrow t'[h'/x]$, $u[h/x] \Rightarrow u'[h'/x]$ and $k[h/x] \Rightarrow k'[h'/x]$. Now,

$$\begin{array}{rcl} e[h/x] & \equiv & ((\lambda y.t)(u :: k))[h/x] \\ & = & (\lambda y.t)[h/x](u[h/x] :: k[h/x]) \\ & \Rightarrow & u'[h'/x]\widehat{y}.(t'[h'/x]k'[h'/x]) \\ & \equiv & e'[h'/x], \end{array}$$

using the rule $(g6)$ of Definition 4.50.

- $e \equiv T\widehat{y}.u \Rightarrow u'[T'/y] \equiv e'$.
This is the consequence of the premises $t \Rightarrow t'$ and $u \Rightarrow u'$. From the IH we get $t[h/x] \Rightarrow t'[h'/x]$ and $u[h/x] \Rightarrow u'[h'/x]$, so applying Lemma 4.52 we derive

$$\begin{array}{rcl} e[h/x] & \equiv & (T\widehat{y}.u)[h/x] \\ & = & T[h/x]\widehat{y}.u[h/x] \\ & \Rightarrow & u'[h'/x][(T'[h'/x])/y] \\ & = & u'[T'/y][h'/x] \\ & \equiv & e'[h'/x], \end{array}$$

using the rule $(g7)$ of Definition 4.50.

- $e \equiv (tk)k_1 \Rightarrow t'(k'@k_1') \equiv e'$.
The statement follows from the premises $t \Rightarrow t'$, $k \Rightarrow k'$ and $k_1 \Rightarrow k_1'$. From the IH we get $t[h/x] \Rightarrow t'[h'/x]$, $k[h/x] \Rightarrow k'[h'/x]$ and $k_1[h/x] \Rightarrow k_1'[h'/x]$. Now, by using the rule $(g8)$ of Definition 4.50, we conclude

$$\begin{array}{rcl} e[h/x] & \equiv & ((tk)k_1)[h/x] \\ & = & (t[h/x]k[h/x])k_1[h/x] \\ & \Rightarrow & t'[h'/x](k'[h'/x]@k_1'[h'/x]) \\ & \equiv & e'[h'/x]. \end{array}$$

- $e \equiv \widehat{x}.xk \Rightarrow k' \equiv e'$.
The statement follows from the premise $k \Rightarrow k'$. From the IH we get $k[x := h] \Rightarrow k'[x := h']$, and then by using the rule $(g9)$ of Definition 4.50 we conclude

$$\begin{array}{rcl} e[h/x] & \equiv & (\widehat{x}.xk)[h/x] \\ & = & \widehat{x}.xk[h/x] \\ & \Rightarrow & k'[h'/x] \\ & \equiv & e'[h'/x], \end{array}$$

so the proof is done.□

The expression e^*, introduced in the following definition, is obtained from e by simultaneously reducing all existing redexes of e.

Definition 4.55 *Let e be an expression. The expression e^* is inductively defined as follows:*

$(*1)$ $x^* \equiv x$

$(*2)$ $(\lambda x.t)^* \equiv \lambda x.t^*$

$(*3)$ $(\widehat{x}.t)^* \equiv \widehat{x}.t^*$

$(*4)$ $(t :: k)^* \equiv t^* :: k^*$

$(*5)$ $(tk)^* \equiv t^*k^*$ *if* $tk \neq (\lambda x.v)(u :: k_1)$ *and* $tk \neq T(\widehat{x}.v)$ *and* $tk \neq (uk_1)k_2$

$(*6)$ $((\lambda x.t)(u :: k))^* \equiv u^*(\widehat{x}.t^*k^*)$

$(*7)$ $(T(\widehat{x}.v))^* \equiv v^*[T^*/x]$

$(*8)$ $((tk)k_1)^* \equiv t^*(k^*@k_1^*)$.

Theorem 4.56 (Star-property of $\Rightarrow$) *If $e \Rightarrow e'$, then $e' \Rightarrow e^*$.*

Proof: By induction of the structure of e. We are proving several cases, the remaining cases are similar.

- $e \equiv x$.
  e can be only reduced in parallel to itself i.e. $e' \equiv x$. On the other hand, $e^* \equiv x$.

- $e \equiv \widehat{x}.t$.
  Then $e' \equiv \widehat{x}.t'$ for some t' such that $t \Rightarrow t'$. From the IH, we get $t' \Rightarrow t^*$, yielding $e' \equiv \widehat{x}.t' \Rightarrow \widehat{x}.t^* \equiv e^*$.

- $e \equiv tk$ and $e \neq (\lambda x.v)(u :: k)$ and $e \neq T(\widehat{x}.v)$ and $e \neq (uk_1)k_2$.
  In that case, applying $(g7)$ we get $e \Rightarrow t'k'$ for some t' and k' such that $t \Rightarrow t'$ and $k \Rightarrow k'$. From the IH, $t' \Rightarrow t^*$ and $k' \Rightarrow k^*$, so we have $e' \equiv t'k' \Rightarrow t^*k^* \equiv e^*$.

- $e \equiv (\lambda x.t)(u :: k)$.
  Now there are two options for the structure of e', because e can be reduced in parallel by rules $(g3)$ and $(g6)$.

  1. $e' \equiv (\lambda x.t')(u' :: k')$ for some t', u' and k' such that $t \Rightarrow t'$, $u \Rightarrow u'$ and $k \Rightarrow k'$. By applying the IH to each of the three subexpressions, we get $t' \Rightarrow t^*$, $u' \Rightarrow u^*$ and $k' \Rightarrow k^*$. Finally, from $(*6)$ we get $e' \equiv (\lambda x.t')(u' :: k') \Rightarrow u^*\widehat{x}.t^*k^* \equiv e^*$.

2. $e' = u'\widehat{x}.t'k'$ for some t', u' and k' such that $t \Rightarrow t'$, $u \Rightarrow u'$ and $k \Rightarrow k'$. By applying the IH to each of the three subexpressions, we again get $t' \Rightarrow t^*$, $u' \Rightarrow u^*$ and $k' \Rightarrow k^*$. The statement now follows from rules $(*3)$ and $(*5)$.

- $G \equiv T\widehat{x}.v$.
  Also in this case there are two options for the structure of e', because e can be reduced in parallel by rules $(g3)$ and $(g7)$.

1. $e' \equiv T'\widehat{x}.v'$ for some T' and v' such that $T \Rightarrow T'$ and $v \Rightarrow v'$. From the IH on both subexpressions we get $T' \Rightarrow T^*$ and $v' \Rightarrow v^*$. Finally, by using $(*7)$ we get $e' \equiv T'\widehat{x}.v' \Rightarrow v^*[T^*/x] \equiv e^*$.
2. $e' = v'[T'/x]$ for some T' and v' such that $T \Rightarrow T'$ and $v \Rightarrow v'$. From the IH on both subexpressions we get $T' \Rightarrow T^*$ and $v' \Rightarrow v^*$. Proposition 4.54 yields $e' \equiv v'[T'/x] \Rightarrow v^*[T^*/x] \equiv e^*$.$\Box$

Now, it is easy to prove the diamond-property for $\Rightarrow$.

Theorem 4.57 (Diamond-property for $\Rightarrow$)
If $e_1 \Leftarrow e \Rightarrow e_2$, then $e_1 \Rightarrow e' \Leftarrow e_2$ for some e'.

Finally, as a consequence of Theorem 4.57, we obtain the confluence of the λ_V^{Gtz}-calculus.

Theorem 4.58 (Confluence of the λ_V^{Gtz}-calculus)
If $e_1 \twoheadleftarrow e \twoheadrightarrow e_2$ then $e_1 \twoheadrightarrow e' \twoheadleftarrow e_2$ for some e'.

If the definitions of the parallel reductions and e^* are changed in accordance to the syntax and reduction rules of the λ_L^{Gtz}-calculus, the proof of confluence of the other λ^{Gtz} sub-calculus is analogous to the one for λ_V^{Gtz}. In order to avoid repetition, here we present only the two notions that differ significantly.

Definition 4.59 (Parallel reductions for the λ_L^{Gtz}-calculus)

$$\frac{}{x \Rightarrow x}\ (g1) \qquad \frac{t \Rightarrow t'}{\lambda x.t \Rightarrow \lambda x.t'}\ (g2) \qquad \frac{t \Rightarrow t',\ k \Rightarrow k'}{tk \Rightarrow t'k'}\ (g3)$$

$$\frac{t \Rightarrow t'}{\widehat{x}.t \Rightarrow \widehat{x}.t'}\ (g4) \qquad \frac{t \Rightarrow t',\ k \Rightarrow k'}{t :: k \Rightarrow t' :: k'}\ (g5)$$

$$\frac{t \Rightarrow t',\ u \Rightarrow u',\ k \Rightarrow k'}{(\lambda x.t)(u :: k) \Rightarrow u'\widehat{x}.(t'k')}\ (g6) \qquad \frac{u \Rightarrow u',\ t \Rightarrow t'}{u(\widehat{x}.t) \Rightarrow t'[u'/x]}\ (g7)$$

$$\frac{t \Rightarrow t',\ k \Rightarrow k',\ l \Rightarrow l'}{(tk)l \Rightarrow t'(k'@l')}\ (g8) \qquad \frac{k \Rightarrow k'}{\widehat{x}.xk \Rightarrow k'}\ (g9)$$

Definition 4.60 *Let e be an expression. The expression e^* for the λ_L^{Gtz}-calculus is inductively defined as follows:*

$(*1)$ $x^* \equiv x$

$(*2)$ $(\lambda x.t)^* \equiv \lambda x.t^*$

$(*3)$ $(\widehat{x}.t)^* \equiv \widehat{x}.t^*$

$(*4)$ $(t :: k)^* \equiv t^* :: k^*$

$(*5)$ $(tk)^* \equiv t^*k^*$ *if* $tk \neq (\lambda x.v)(u :: k_1)$ *and* $tk \neq t_1(\widehat{x}.v)$ *and* $tk \neq (uk)l$

$(*6)$ $((\lambda x.t)(u :: k))^* \equiv u^*(\widehat{x}.t^*k^*)$

$(*7)$ $(t(\widehat{x}.v))^* \equiv v^*[t^*/x]$

$(*8)$ $((tk)l)^* \equiv t^*(k^*@l^*)$.

Chapter 5

$\lambda_{®}^{\mathsf{Gtz}}$-calculus

In this chapter we present the $\lambda_{®}^{\mathsf{Gtz}}$-calculus (pronounced *resource control lambda Gentzen*), which provides the Curry-Howard correspondence for the implicative fragment of intuitionistic logic formulated in Gentzen's sequent calculus with explicit structural rules of weakening and contraction. It is obtained by extending the λ^{Gtz}-calculus of Espírito Santo [27], presented in the previous chapter.

The design of the $\lambda_{®}^{\mathsf{Gtz}}$-calculus has been motivated by Kesner and Lendgrand's $\lambda\mathsf{lxr}$-calculus [45], an intuitionistic calculus featuring linearity and explicit operators for substitution, erasure and duplication in the natural deduction setting. This calculus was designed to provide a correspondence with multiplicative exponential fragment of linear logic, whereas the $\lambda_{®}^{\mathsf{Gtz}}$-calculus brings the correspondence to the intuitionistic sequent calculus with explicit structural rules. The $\lambda\mathsf{lxr}$-calculus introduces new terms in a way that keeps the good underlying properties of its predecessor $\lambda\mathsf{x}$ [8, 62], a calculus with explicit substitution, such as strong normalisation, confluence and subject reduction. Moreover, it has a sound and complete correspondence with intuitionistic proof net model, and the computation preserves the set of free variables and the linearity of terms.

In order to obtain a sequent counterpart of the $\lambda\mathsf{lxr}$-calculus, we extended the λ^{Gtz}-calculus with explicit expressions for erasure and duplication, which on the logical side correspond to the explicit structural rules of weakening and contraction, respectively. This means that the important computational features such as erasure and duplication are explicitly controlled, rather than being hidden in the form of other reduction rules.

One natural consequence is the decomposition of reduction steps into more atomic ones, which reveals the details of computation that are usually left implicit. Since erasing and duplicating (sub)terms essentially changes the structure of a program, it is important to see how this mechanism really works and to be able to control this part of computation, and to eventually propose a way of optimization of the computation process in terms of the required resources. The other consequence is the change of the notion of an expression. In the presence of explicit resource control operators, the only well-formed expressions are the ones in which every binder binds exactly one occurrence of a free variable, and therefore only such expressions are in our focus[1]. However, this is not a restriction in comparison with ordinary lambda terms, since all lambda terms can be represented in the resource control calculus.

The structure of this chapter mimics the structure of the previous one - we first propose the untyped calculus, then introduce a type assignment system with simple types, and finally add the intersection type operator, in order to obtain the characterisation of strongly normalising $\lambda_{®}^{\mathsf{Gtz}}$- terms. The entire chapter represents original contribution of the thesis, and was developed by Pierre Lescanne, Silvia Ghilezan, Silvia Likavec, Dragiša Žunić and myself, and published in [35, 32, 33].

5.1 Type-free $\lambda_{®}^{\mathsf{Gtz}}$-calculus

The *resource control lambda Gentzen* calculus $\lambda_{®}^{\mathsf{Gtz}}$ is derived from the λ^{Gtz}-calculus (more precisely from its confluent sub-calculus $\lambda_{V}^{\mathsf{Gtz}}$ presented in Section 4.4) by adding the explicit operators for erasure and duplication to both terms and contexts. On the other hand, it can be seen as a sequent counterpart of the λlxr-calculus in which the substitution is treated as a meta-operator, i.e. implicitly. The first variant of this calculus was proposed in [35][2]. In accordance with its corresponding structural rule in the sequent calculus, the operator which performs duplication will from now on be called *contraction*, whereas the erasure operator will be referred to as *weakening*. This practise has been already established in the λ^{Gtz}-calculus, where the application operator was called cut.

5.1.1 Syntax

First of all, we introduce the syntactic category of *pre-expressions*. The abstract syntax of $\lambda_{®}^{\mathsf{Gtz}}$ pre-expressions is the following:

Pre-values	F	$::=$	$x \,\vert\, \lambda x.f \,\vert\, x \odot f \,\vert\, x <^{x_1}_{x_2} f$
Pre-terms	f	$::=$	$F \,\vert\, fc$
Pre-contexts	c	$::=$	$\widehat{x}.f \,\vert\, f :: c \,\vert\, x \odot c \,\vert\, x <^{x_1}_{x_2} c$

where x ranges over a denumerable set of term variables.

[1]Corresponding notion in λlxr-calculus is the notion of "linear terms".

[2]Where it was named *linear lambda Gentzen calculus* and denoted by $\ell\lambda^{\mathsf{Gtz}}$

A *pre-value* can be a variable, an abstraction, a weakening or a contraction; a *pre-term* is either a value or a cut (an application). A *pre-context* is one of the following: a selection, a context constructor (usually called “cons"), a weakening on pre-context or a contraction on a pre-context. Pre-terms and pre-contexts are together referred to as the *pre-expressions* and will be ranged over by E. Pre-contexts $x \odot c$ and $x <^{x_1}_{x_2} c$ behave exactly as the corresponding pre-terms $x \odot f$ and $x <^{x_1}_{x_2} f$ in the untyped calculus, so they will mostly not be treated separately.

Definition 5.1

(i) The list *of free variables of a pre-expression E, denoted by $Fv[E]$, is defined as follows (where l,m denotes the list obtained by the concatenation of the two lists l and m and $l \setminus x$ denotes the list obtained by removing all occurrences of an element x from the list l):*

$$\begin{array}{lcl}
Fv[x] & = & x; \\
Fv[\lambda x.f] & = & Fv[f] \setminus x; \\
Fv[fc] & = & Fv[f], Fv[c]; \\
Fv[\widehat{x}.f] & = & Fv[f] \setminus x; \\
Fv[f :: c] & = & Fv[f], Fv[c]; \\
Fv[x \odot E] & = & x, Fv[E]; \\
Fv[x <^{x_1}_{x_2} E] & = & x, ((Fv[E] \setminus x_1) \setminus x_2).
\end{array}$$

(ii) The set *of free variables of a pre-expression E, denoted by $Fv(E)$, is extracted out of the list $Fv[E]$, by un-ordering the list and removing multiple occurrences of each variable, if any.*

(iii) The set *of bound variables of a pre-expression E, denoted by $Bv(E)$, contains all variables that exist in E, but are not free in it.*

Example 5.2 *Let $E \equiv z \odot u <^{x_1}_{x_2} x(z :: x_2 :: x_1 :: \widehat{y}.y)$. Then:*

$$\begin{array}{rcl}
Fv[E] & = & z,u,x,z \\
Fv(E) & = & \{x,u,z\} \\
Bv(E) & = & \{x_1,x_2,y\}.
\end{array}$$

In $x <^{x_1}_{x_2} E$, the contraction binds the variables x_1 and x_2 in E and introduces a free variable x. The operator $x \odot E$ also introduces a free variable x. In order to avoid parentheses, we let the scope of all binders extend to the right as much as possible.

One may wonder why do we define both lists and sets of free variables. The notion of a list $Fv[E]$ is used in the definition of implicit substitution in the case of contraction (see Figure 5.4) where the order of variables needs to be controlled, whereas in all other situations, where the order of free variables is irrelevant, it is more convenient to work with sets. The following lemma provides the direct way for constructing $Fv(E)$, based on the structure of a pre-expression E.

Lemma 5.3 *The set $Fv(E)$ can be constructed as follows:*

$$\begin{array}{lcl} Fv(x) & = & x; \\ Fv(\lambda x.f) & = & Fv(f) \setminus \{x\}; \\ Fv(fc) & = & Fv(f) \cup Fv(c); \\ Fv(\widehat{x}.f) & = & Fv(f) \setminus \{x\}; \\ Fv(f :: c) & = & Fv(f) \cup Fv(c); \\ Fv(x \odot E) & = & \{x\} \cup Fv(E); \\ Fv(x <^{x_1}_{x_2} E) & = & \{x\} \cup (Fv(E) \setminus \{x_1, x_2\}). \end{array}$$

Proof: The proof follows directly from items (i) and (ii) of Definition 5.1. □

Now, using the notion of the set of free variables, we are able to extract the set of $\lambda_{®}^{\mathsf{Gtz}}$-expressions (namely values, terms and contexts) out of the set of $\lambda_{®}^{\mathsf{Gtz}}$ pre-expressions. The set of $\lambda_{®}^{\mathsf{Gtz}}$-expressions $\Lambda_{®}^{\mathsf{Gtz}} = \mathrm{V}_{®}^{\mathsf{Gtz}} \cup \mathrm{T}_{®}^{\mathsf{Gtz}} \cup \mathrm{C}_{®}^{\mathsf{Gtz}}$, where $\mathrm{V}_{®}^{\mathsf{Gtz}}$ denotes the set of $\lambda_{®}^{\mathsf{Gtz}}$-values, $\mathrm{T}_{®}^{\mathsf{Gtz}}$ denotes the set of $\lambda_{®}^{\mathsf{Gtz}}$-terms and $\mathrm{C}_{®}^{\mathsf{Gtz}}$ denotes the set of $\lambda_{®}^{\mathsf{Gtz}}$-contexts.

Definition 5.4 *The set of $\lambda_{®}^{\mathsf{Gtz}}$-expressions denoted by $\Lambda_{®}^{\mathsf{Gtz}}$, is a subset of the set of pre-expressions, defined in Figure 5.1.*

In the rest of the chapter, we will use the following notation:

- $T, T', T_1 ...$ for values;
- $t, u, v...$ for terms;
- $k, k', k_1 ...$ for contexts and
- $e, e', e_1 ...$ for expressions.

Informally, we say that an expression is a pre-expression in which in every sub-expression every free variable occurs exactly once, and every binder binds (exactly one occurrence of) a free variable. When restricted to terms, this notion corresponds to the notion of linear terms in [45]. In that sense, only linear expressions are in the focus of our investigation. These conditions will be assumed throughout this chapter without explicit mentioning.

However, this assumption is not a restriction, since every λ^{Gtz}-expression has a corresponding $\lambda_{®}^{\mathsf{Gtz}}$-expression.

$$\frac{}{x \in \mathrm{V}^{\mathsf{Gtz}}_{\circledR}} \qquad \frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \quad x \in Fv(f)}{\lambda x.f \in \mathrm{V}^{\mathsf{Gtz}}_{\circledR}}$$

$$\frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \quad c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR} \quad Fv(f) \cap Fv(c) = \emptyset}{fc \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR}} \qquad \frac{F \in \mathrm{V}^{\mathsf{Gtz}}_{\circledR}}{F \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR}}$$

$$\frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \quad x \in Fv(f)}{\widehat{x}.f \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR}} \qquad \frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \quad c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR} \quad Fv(f) \cap Fv(c) = \emptyset}{f :: c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR}}$$

$$\frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \quad x \notin Fv(f)}{x \odot f \in \mathrm{V}^{\mathsf{Gtz}}_{\circledR}} \qquad \frac{c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR} \quad x \notin Fv(c)}{x \odot c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR}}$$

$$\frac{f \in \mathrm{T}^{\mathsf{Gtz}}_{\circledR} \quad x_1 \neq x_2 \quad x_1, x_2 \in Fv(f) \quad x \notin Fv(f) \setminus \{x_1, x_2\}}{x <^{x_1}_{x_2} f \in \mathrm{V}^{\mathsf{Gtz}}_{\circledR}}$$

$$\frac{c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR} \quad x_1 \neq x_2 \quad x_1, x_2 \in Fv(c) \quad x \notin Fv(c) \setminus \{x_1, x_2\}}{x <^{x_1}_{x_2} c \in \mathrm{C}^{\mathsf{Gtz}}_{\circledR}}$$

Figure 5.1: $\Lambda^{\mathsf{Gtz}}_{\circledR}$: $\lambda^{\mathsf{Gtz}}_{\circledR}$-expressions

Definition 5.5 *Mapping* $[\ ]_{rc} : \Lambda^{\mathsf{Gtz}} \to \Lambda^{\mathsf{Gtz}}_{®}$ *is defined in the following way:*

$$
\begin{array}{rcl}
[x]_{rc} & = & x \\
[\lambda x.t]_{rc} & = & \begin{cases} \lambda x.[t]_{rc}, & x \in Fv(t) \\ \lambda x.x \odot [t]_{rc}, & x \notin Fv(t) \end{cases} \\
[\widehat{x}.t]_{rc} & = & \begin{cases} \widehat{x}.[t]_{rc}, & x \in Fv(t) \\ \widehat{x}.x \odot [t]_{rc}, & x \notin Fv(t) \end{cases} \\
[tk]_{rc} & = & \begin{cases} [t]_{rc}[k]_{rc}, & Fv(t) \cap Fv(k) = \emptyset \\ x <^{x_1}_{x_2} [t[x_1/x]k[x_2/x]]_{rc}, & x \in Fv(t) \cap Fv(k) \end{cases} \\
[t :: k]_{rc} & = & \begin{cases} [t]_{rc} :: [k]_{rc}, & Fv(t) \cap Fv(k) = \emptyset \\ x <^{x_1}_{x_2} [t[x_1/x] :: k[x_2/x]]_{rc}, & x \in Fv(t) \cap Fv(k) \end{cases}
\end{array}
$$

Proposition 5.6

(i) If $t \in \Lambda^{\mathsf{Gtz}}$*, then* $[t]_{rc} \in \mathrm{T}^{\mathsf{Gtz}}_{®}$.

(ii) If $k \in \Lambda^{\mathsf{Gtz}}_{C}$*, then* $[k]_{rc} \in \mathrm{C}^{\mathsf{Gtz}}_{®}$.

Proof: By mutual induction on the structure of t and k, using definition of $\lambda^{\mathsf{Gtz}}_{®}$-expressions, given in Figure 5.1. □

The correspondence between λ^{Gtz}-expressions and $\lambda^{\mathsf{Gtz}}_{®}$-expressions is illustrated by the following example.

Example 5.7 *Pre-expressions* $E_1 \equiv \lambda x.y$, $E_2 \equiv \widehat{x}.y$ *and* $E_3 \equiv \lambda x.x(x :: \widehat{y}.y)$ *are* λ^{Gtz}*-expressions, but are not* $\lambda^{\mathsf{Gtz}}_{®}$*-expressions. The reason is the presence of void abstraction or selection in* E_1 *and* E_2*, and two occurrences of free variable* x *in the sub-expression of* E_3*. On the other hand,* $\lambda x.x \odot y$, $\widehat{x}.x \odot y$ *and* $\lambda x.x <^{x_1}_{x_2} x_1(x_2 :: \widehat{y}.y)$ *are their corresponding* $\lambda^{\mathsf{Gtz}}_{®}$*-expressions.*

The previous example illustrates the role of the introduced operators of weakening and contraction in controlling the resources.

The role of the weakening (erasure) operator in the expression $x \odot e$ is to denote that the variable x does not occur in e. So we have to explicitly add it in order to perform the lambda abstraction or the selection of that variable, because void abstractions or selections are not allowed in the resource aware calculi. The origins of this approach can be traced back to Church's λI-calculus.

Similarly, the role of the contraction (duplication) operator is to control multiple occurrences of a variable in an expression. In the resource aware calculi, each variable represents a resource entity and therefore should not occur twice in an expression. If nevertheless, we want to have two positions for the same variable, we have to duplicate it explicitly, using fresh names. This is done by using the operator $x <^{x_1}_{x_2} e$ which creates two fresh variables x_1 and x_2 and denotes that the two variables have the same role in the expression e.

In the sequel, we will use the following notation:

- $X \odot e$ stands for $x_1 \odot \dots x_n \odot e$;
- $X <^{Y}_{Z} e$ stands for $x_1 <^{y_1}_{z_1} \dots x_n <^{y_n}_{z_n} e$;

where X, Y and Z are lists of the size n, consisting of all distinct variables $x_1,\dots,x_n$, $y_1,\dots,y_n$, $z_1,\dots,z_n$. If $n = 0$, i.e., if X, Y and Z are empty lists, then $X \odot e = X <^{Y}_{Z} e = e$. Note that due to the equivalence relation defined in Figure 5.5, we can use these notations also for sets of variables of the same size.

Alpha-conversion is the (standard) congruence generated by renaming of bound variables. For example, $w \odot \lambda x.x <^{y}_{z} y :: z :: \widehat{u}.u \equiv_{\alpha} w \odot \lambda x'.x' <^{y'}_{z'} y' :: z' :: \widehat{u}.'u'$. All the operations defined along this chapter are considered modulo alpha-conversion so that in particular capture of variables is not possible.

Using α-conversion, in what follows we consider Barendregt's convention [3] for variables: in the same expression a variable cannot be both free and bound, therefore bound variables should be renamed using fresh names. This applies to all three binders of the $\lambda_{®}^{\mathsf{Gtz}}$-calculus: $\lambda x.t$ and $\widehat{x}.t$, which bind x in t, and $x <^{x_1}_{x_2} e$ which binds x_1 and x_2 in e.

5.1.2 Operational semantics

The starting point for the reduction system of the $\lambda_{®}^{\mathsf{Gtz}}$-calculus is the reduction system of the $\lambda_{V}^{\mathsf{Gtz}}$-calculus, defined in Subsection 4.4.1, thus the computation over the set of $\lambda_{®}^{\mathsf{Gtz}}$-expressions reflects the cut-elimination process. But the presence of explicit weakening and contraction requires an additional number of reduction rules to discipline these new operators. Four groups of reductions in the $\lambda_{®}^{\mathsf{Gtz}}$-calculus are given in Figure 5.2.

Reductions are naturally divided into four groups. The first group consists of β, π, σ and μ reductions from the $\lambda_{V}^{\mathsf{Gtz}}$-calculus. New reductions are added to deal with explicit contraction (γ reductions) and weakening (ω reductions). These rules perform:

- propagation of contraction into an expression (γ-reductions);
- extraction of weakening out of an expression (ω-reductions).

This discipline allows us to optimize the computation by delaying the duplication of terms on the one hand, and by performing the erasure of terms as soon as

$$\begin{array}{rrcl}
(\beta) & (\lambda x.t)(u::k) & \to & u(\widehat{x}.tk) \\
(\sigma) & T(\widehat{x}.v) & \to & v[T/x] \\
(\pi) & (tk)k' & \to & t(k@k') \\
(\mu) & \widehat{x}.xk & \to & k \\
\\
(\gamma_1) & x <^{x_1}_{x_2} (\lambda y.t) & \to & \lambda y.x <^{x_1}_{x_2} t \\
(\gamma_2) & x <^{x_1}_{x_2} (tk) & \to & (x <^{x_1}_{x_2} t)k, \quad \text{if } x_1,x_2 \notin Fv(k) \\
(\gamma_3) & x <^{x_1}_{x_2} (tk) & \to & t(x <^{x_1}_{x_2} k), \quad \text{if } x_1,x_2 \notin Fv(t) \\
(\gamma_4) & x <^{x_1}_{x_2} (\widehat{y}.t) & \to & \widehat{y}.(x <^{x_1}_{x_2} t) \\
(\gamma_5) & x <^{x_1}_{x_2} (t::k) & \to & (x <^{x_1}_{x_2} t)::k, \quad \text{if } x_1,x_2 \notin Fv(k) \\
(\gamma_6) & x <^{x_1}_{x_2} (t::k) & \to & t::(x <^{x_1}_{x_2} k), \quad \text{if } x_1,x_2 \notin Fv(t) \\
\\
(\omega_1) & \lambda x.(y \odot t) & \to & y \odot (\lambda x.t), \quad x \neq y \\
(\omega_2) & (x \odot t)k & \to & x \odot (tk) \\
(\omega_3) & t(x \odot k) & \to & x \odot (tk) \\
(\omega_4) & \widehat{x}.(y \odot t) & \to & y \odot (\widehat{x}.t), \quad x \neq y \\
(\omega_5) & (x \odot t)::k & \to & x \odot (t::k) \\
(\omega_6) & t::(x \odot k) & \to & x \odot (t::k) \\
\\
(\gamma\omega_1) & x <^{x_1}_{x_2} (y \odot e) & \to & y \odot (x <^{x_1}_{x_2} e) \qquad x_1 \neq y \neq x_2 \\
(\gamma\omega_2) & x <^{x_1}_{x_2} (x_1 \odot e) & \to & e[x/x_2]
\end{array}$$

Figure 5.2: Reduction rules of the $\lambda_{®}^{\mathsf{Gtz}}$-calculus

possible on the other. Finally, the two rules in the $\gamma\omega$ group explain the interaction between explicit resource operators that are of different nature. Since none of these rules are performed on cuts, they can be considered as auxiliary (second class citizens) reductions.

As in the λ^{Gtz}-calculus, reductions (π) and (σ) are executed via meta-operators. The meta-operator for appending two contexts, $k@k'$, from the rule (π) is now defined by the rules presented in Figure 5.3. This definition represents an extension of the definition from Figure 4.3 with the cases covering new context operators, namely weakening on contexts and contraction on contexts.

$$\begin{array}{lcl}
(\widehat{x}.t)@k' & \triangleq & \widehat{x}.tk'; \\
(t::k)@k' & \triangleq & t::(k@k'); \\
(x \odot k)@k' & \triangleq & x \odot (k@k'); \\
(x <^{x_1}_{x_2} k)@k' & \triangleq & x <^{x_1}_{x_2} (k@k').
\end{array}$$

Figure 5.3: Meta-operator @ in the $\lambda_{®}^{\mathsf{Gtz}}$-calculus

The inductive definition of the meta operator $[\ /\ ]$, representing the implicit substitution of free variables, is given in Figure 5.4. There are several specific features of the substitution in the $\lambda_{\circledR}^{\mathsf{Gtz}}$-calculus that need to be emphasized.

First, the substitution is introduced in the (σ) reduction: $T(\widehat{x}.v) \to v[T/x]$. This means that we always substitute a value T for a variable, therefore this calculus supports the call-by-value computational strategy.

Second, since each bound variable actually appears in the $\lambda_{\circledR}^{\mathsf{Gtz}}$-expression, we know that in $v[T/x]$ it holds $x \in Fv(v)$ (because it is obtained from $T(\widehat{x}.v)$), therefore void substitutions are not possible here and the usual basic case $y[T/x] \triangleq y$ does not need to be defined.

Third, in order to obtain well-formed expression as the result of substitution $e[T/x]$, $Fv(e) \cap Fv(T) = \emptyset$ must hold in this definition. Moreover, notice that e must contain exactly one occurrence of the free variable x, therefore we can assume that $x \notin Fv(T)$.

Finally, notice the interaction of resource operators and the substitution. In the case $(x \odot e)[T/x]$ i.e. if we substitute T for a free variable introduced by weakening, the structure of T will be erased and only its free variables will be left in the resulting expression, thus the result of the substitution is $Fv(T) \odot e$. This behavior corresponds to the computational meaning of weakening as the erasure, and at the same time enables the feature known as the “preservation of interface".

On the other hand, if we substitute T for a free variable introduced by contraction (case $(x <_{x_2}^{x_1} e)[T/x]$) the T would have to appear twice in e - once substituted for x_1 and other time for x_2. But then, free variables of T would appear twice in the resulting expression, which contradicts the definition of the $\lambda_{\circledR}^{\mathsf{Gtz}}$-expression. The way to solve this problem is to make two copies of T (namely T_1 and T_2) using fresh names, then substitute each copy for one of the contracted variables, and eventually perform several contractions on all pairs of the corresponding free variables of T_1 and T_2, contained in the lists $Fv[T_1]$ and $Fv[T_2]$. Thus, the resulting expression is $Fv[T] <_{Fv[T_2]}^{Fv[T_1]} (e[T_1/x_1])[T_2/x_2]$. As showed by Lemma 5.8, this expression can be alternatively written as $Fv[T] <_{Fv[T_2]}^{Fv[T_1]} e[T_1/x_1, T_2/x_2]$, using a notion of parallel substitution. Again, this is in accordance with the computational meaning of contraction as the duplication.

Lemma 5.8 (Permutability of substitution in the contraction) *If $e \in \Lambda_{\circledR}^{\mathsf{Gtz}}$, $x_1, x_2 \in Fv(e)$ and*
$(Fv(e) \setminus \{x_1\}) \cap Fv(T_1) = (Fv(e) \setminus \{x_2\}) \cap Fv(T_2) = Fv(T_1) \cap Fv(T_2) = \emptyset$, then:

$$(e[T_2/x_2])[T_1/x_1] = (e[T_1/x_1])[T_2/x_2].$$

Proof: The proof is straightforward by induction on the structure of the expression

$$
\begin{array}{rcl}
x[T/x] & \triangleq & T \\
(\lambda y.t)[T/x] & \triangleq & \lambda y.t[T/x],\ \ x \neq y \\
(tk)[T/x] & \triangleq & t[T/x]k,\ \ x \notin Fv(k) \\
(tk)[T/x] & \triangleq & tk[T/x],\ \ x \notin Fv(t) \\
(\widehat{y}.t)[T/x] & \triangleq & \widehat{y}.t[T/x],\ \ x \neq y \\
(t :: k)[T/x] & \triangleq & t[T/x] :: k,\ \ x \notin Fv(k) \\
(t :: k)[T/x] & \triangleq & t :: k[T/x],\ \ x \notin Fv(t) \\
(y \odot e)[T/x] & \triangleq & y \odot e[T/x],\ \ x \neq y \\
(x \odot e)[T/x] & \triangleq & Fv(T) \odot e \\
(y <_{y_2}^{y_1} e)[T/x] & \triangleq & y <_{y_2}^{y_1} e[T/x],\ \ x \neq y \\
(x <_{x_2}^{x_1} e)[T/x] & \triangleq & Fv[T] <_{Fv[T_2]}^{Fv[T_1]} e[T_1/x_1, T_2/x_2]
\end{array}
$$

Figure 5.4: Substitution in the $\lambda_{®}^{\mathsf{Gtz}}$-calculus

e that contains free variables x_1 and x_2. □

Definition 5.9 (Parallel substitution) *The substitution that satisfies all conditions of Lemma 5.8 is called parallel substitution and denoted by* $e[T_1/x_1, T_2/x_2]$.

Notice that the parallel substitution is defined only in the presence of very strict conditions. Under these conditions, the notion of substitution is reduced to a significantly simpler notion of "simultaneous replacement" of two variables (that appear exactly once in e) by two terms (that share no free variables with the environment), without any further interaction.

Example 5.10 *Let us consider the substitution in the expression* $(x <_{x_2}^{x_1} e)[T/x]$ *where* $e = x_1 :: x_2 :: \widehat{u}.u$ *and* $T = \lambda y.y \odot z$. *In this case,* $Fv(T) = z$, $T_1 = \lambda y.y \odot z_1$, $T_2 = \lambda y.y \odot z_2$, $Fv(T_1) = z_1$ *and* $Fv(T_2) = z_2$. *Thus:*

$$
\begin{array}{rcl}
(x <_{x_2}^{x_1} x_1 :: x_2 :: \widehat{u}.u)[\lambda y.y \odot z/x] & \triangleq & (z <_{z_2}^{z_1} x_1 :: x_2 :: \widehat{u}.u)[\lambda y.y \odot z_1/x_1, \lambda y.y \odot z_2/x_2] \\
& \triangleq & z <_{z_2}^{z_1} \lambda y.y \odot z_1 :: \lambda y.y \odot z_2 :: \widehat{u}.u.
\end{array}
$$

Therefore, as expected, the value T *is duplicated during the substitution execution.*

Example 5.11 *Now, let us consider the substitution in the slightly modified expression* $(x <_{x_2}^{x_1} e)[T/x]$, *where* $e = x_1 \odot x_2$ *and* $T = \lambda y.y \odot z$. *Now:*

$$
\begin{array}{rcl}
(x <_{x_2}^{x_1} x_1 \odot x_2)[\lambda y.y \odot z/x] & \triangleq & (z <_{z_2}^{z_1} x_1 \odot x_2)[\lambda y.y \odot z_1/x_1, \lambda y.y \odot z_2/x_2] \\
& \triangleq & z <_{z_2}^{z_1} z_1 \odot \lambda y.y \odot z_2.
\end{array}
$$

$$\boxed{\begin{array}{llcl}
(\varepsilon_1) & x \odot (y \odot e) & \equiv & y \odot (x \odot e) \\
(\varepsilon_2) & x <^{x_1}_{x_2} e & \equiv & x <^{x_2}_{x_1} e \\
(\varepsilon_3) & x <^{y}_{z} (y <^{u}_{v} e) & \equiv & x <^{y}_{u} (y <^{z}_{v} e) \\
(\varepsilon_4) & x <^{x_1}_{x_2} (y <^{y_1}_{y_2} e) & \equiv & y <^{y_1}_{y_2} (x <^{x_1}_{x_2} e),\ x \neq y_1, y_2,\ y \neq x_1, x_2
\end{array}}$$

Figure 5.5: Equivalences in the $\lambda^{\mathsf{Gtz}}_{®}$-calculus

In this case, T was not duplicated, because one of the contracted variables, namely x_1, was introduced by weakening, therefore the duplicated value T_2 was erased, and only its free variable z_2 was kept, in order to preserve interface of the expression. One can also observe that in this example, the reduction $(\gamma\omega_2)$ could have been applied before the substitution execution, yielding:

$$\begin{array}{lcl}
(x <^{x_1}_{x_2} x_1 \odot x_2)[\lambda y.y \odot z/x] & \to_{\gamma\omega_2} & (x_1 \odot x_2)[x/x_2][\lambda y.y \odot z/x] \\
& \triangleq & (x_1 \odot x)[\lambda y.y \odot z/x] \\
& \triangleq & x_1 \odot \lambda y.y \odot z.
\end{array}$$

Besides reductions, operational semantics of the $\lambda^{\mathsf{Gtz}}_{®}$-calculus contains also the congruence relation defined by the equivalencies given in Figure 5.5.

Notice that because we work only with the $\lambda^{\mathsf{Gtz}}_{®}$-expressions, i.e. well-formed expressions, no variable is lost during the computation, which is stated by the following proposition.

Proposition 5.12 (Preservation of free variables)

(i) $Fv(e[T/x]) = (Fv(e) \setminus \{x\}) \cup Fv(T)$.

(ii) $Fv(k@k') = Fv(k) \cup Fv(k')$.

(iii) *If* $e \to e'$, *then* $Fv(e) = Fv(e')$.

(iv) *If* $e \equiv e'$, *then* $Fv(e) = Fv(e')$.

Proof: The proof of (i) is by induction on the structure of e and uses Lemma 5.3. We show the base case and the resource control related cases.

- Base case $e \equiv x$. Then $Fv(e) = \{x\}$, and

$$Fv(x[T/x]) \triangleq Fv(T) = (\{x\} \setminus \{x\}) \cup Fv(T).$$

- Case $e \equiv y \odot e'$, where $y \neq x$. Then $Fv(e) = \{y\} \cup Fv(e')$, hence

$$\begin{aligned}
Fv(e[T/x]) &\equiv Fv((y \odot e')[T/x]) \\
&\triangleq Fv(y \odot e'[T/x]) \\
&=_{IH} \{y\} \cup (Fv(e') \setminus \{x\}) \cup Fv(T) \\
&= (Fv(e) \setminus \{x\}) \cup Fv(T).
\end{aligned}$$

- Case $e \equiv x \odot e'$. Then $Fv(e') = Fv(e) \setminus \{x\}$, hence

$$\begin{aligned}
Fv(e[T/x]) &\equiv Fv((x \odot e')[T/x]) \\
&\triangleq Fv(Fv(T) \odot e') \\
&= Fv(e') \cup Fv(T) \\
&= (Fv(e) \setminus \{x\}) \cup Fv(T).
\end{aligned}$$

- Case $e \equiv y <^{y_1}_{y_2} e'$, where $y \neq x$. Then $Fv(e) = \{y\} \cup (Fv(e') \setminus \{y_1, y_2\}$, hence

$$\begin{aligned}
Fv(e[T/x]) &\equiv Fv((y <^{y_1}_{y_2} e')[T/x]) \\
&\triangleq Fv(y <^{y_1}_{y_2} e'[T/x]) \\
&=_{IH} \{y\} \cup (Fv(e') \setminus \{x, y_1, y_2\}) \cup Fv(T) \\
&= (Fv(e) \setminus \{x\}) \cup Fv(T).
\end{aligned}$$

- Case $e \equiv x <^{x_1}_{x_2} e'$. Then $Fv(e) = \{x\} \cup Fv(e') \setminus \{x_1, x_2\}$, hence

$$\begin{aligned}
Fv(e[T/x]) &\equiv Fv((x <^{x_1}_{x_2} e')[T/x]) \\
&\triangleq Fv(Fv[T] <^{Fv[T_1]}_{Fv[T_2]} e'[T_1/x_1, T_2/x_2]) \\
&\triangleq Fv(Fv[T] <^{Fv[T_1]}_{Fv[T_2]} (e'[T_1/x_1])[T_2/x_2]) \\
&= Fv(T) \cup Fv((e'[T_1/x_1])[T_2/x_2]) \setminus (Fv(T_1) \cup Fv(T_2)) \\
&=_{IH} Fv(T) \cup (Fv(e'[T_1/x_1]) \setminus \{x_2\} \cup Fv(T_2)) \setminus (Fv(T_1) \cup Fv(T_2)) \\
&=_{IH} Fv(T) \cup (Fv(e') \setminus \{x_1, x_2\} \cup Fv(T_1) \cup Fv(T_2)) \setminus (Fv(T_1) \cup Fv(T_2)) \\
&= Fv(T) \cup (Fv(e') \setminus \{x_1, x_2\}) \\
&= (Fv(e) \setminus \{x\}) \cup Fv(T).
\end{aligned}$$

The proof of (ii) is on the structure of the context k, also using Lemma 5.3.

- Base case $k \equiv \widehat{x}.t$. Then $Fv(k) = Fv(t) \setminus \{x\}$ and $x \notin Fv(k')$, hence

$$\begin{aligned}
Fv(k@k') &\equiv Fv((\widehat{x}.t)@k') \\
&\triangleq Fv(\widehat{x}.tk') \\
&= (Fv(t) \cup Fv(k')) \setminus \{x\} \\
&= Fv(k) \cup Fv(k').
\end{aligned}$$

- Case $k \equiv x <^{x_1}_{x_2} k''$. Then $Fv(k) = \{x\} \cup Fv(k'') \setminus \{x_1,x_2\}$ and $x,x_1,x_2 \notin Fv(k')$, hence

$$\begin{array}{rcl} Fv(k@k') & \equiv & Fv((x <^{x_1}_{x_2} k'')@k') \\ & \triangleq & Fv(x <^{x_1}_{x_2} k''@k') \\ & = & \{x\} \cup Fv(k''@k') \setminus \{x_1,x_2\} \\ & =_{IH} & \{x\} \cup (Fv(k'') \cup Fv(k')) \setminus \{x_1,x_2\} \\ & = & Fv(k) \cup Fv(k'). \end{array}$$

- The remaining two cases are similar.

The proofs of (iii) and (iv) are straightforward by case analysis on the reduction and equivalence rules, using (i) in the case of σ-reduction and (ii) in the case of π-reduction. □

We conclude the presentation of the untyped $\lambda^{\mathsf{Gtz}}_{®}$-calculus with an example of computation.

Example 5.13 *Let us recall the λ^{Gtz}-term $t \equiv (\lambda x.y)(y(\widehat{z}.z) :: \widehat{u}.\lambda y'.u)$ from the Example 4.2. Using Definition 5.5, we obtain its corresponding $\lambda^{\mathsf{Gtz}}_{®}$-term:*

$$t \equiv y <^{y_1}_{y_2} (\lambda x.x \odot y_1)(y_2(\widehat{z}.z) :: \widehat{u}.(\lambda y'.y' \odot u)).$$

One possible way to reduce t to its normal form is:

$$\begin{array}{rcl} t & \equiv & y <^{y_1}_{y_2} (\lambda x.x \odot y_1)(y_2(\widehat{z}.z) :: \widehat{u}.(\lambda y'.y' \odot u)) \\ & \to_{\sigma} & y <^{y_1}_{y_2} (\lambda x.x \odot y_1)(z[y_2/z] :: \widehat{u}.(\lambda y'.y' \odot u)) \\ & \triangleq & y <^{y_1}_{y_2} (\lambda x.x \odot y_1)(y_2 :: \widehat{u}.(\lambda y'.y' \odot u)) \\ & \to_{\beta} & y <^{y_1}_{y_2} y_2(\widehat{x}.(x \odot y_1)(\widehat{u}.(\lambda y'.y' \odot u))) \\ & \to_{\sigma} & y <^{y_1}_{y_2} ((x \odot y_1)(\widehat{u}.(\lambda y'.y' \odot u))[y_2/x]) \\ & \triangleq & y <^{y_1}_{y_2} (y_2 \odot y_1)(\widehat{u}.(\lambda y'.y' \odot u)) \\ & \to_{\gamma_2} & (y <^{y_1}_{y_2} y_2 \odot y_1)(\widehat{u}.(\lambda y'.y' \odot u)) \\ & \equiv_{\varepsilon_2} & (y <^{y_2}_{y_1} y_2 \odot y_1)(\widehat{u}.(\lambda y'.y' \odot u)) \\ & \to_{\gamma\omega_2} & y_1[y/y_1](\widehat{u}.(\lambda y'.y' \odot u)) \\ & \triangleq & y(\widehat{u}.(\lambda y'.y' \odot u)) \\ & \to_{\sigma} & (\lambda y'.y' \odot u)[y/u] \\ & \triangleq & \lambda y'.y' \odot y. \end{array}$$

Notice that the resulting $\lambda^{\mathsf{Gtz}}_{®}$-term $\lambda y'.y' \odot y$ corresponds exactly to the λ^{Gtz}-term $\lambda y'.y$, which was the result of the computation in the Example 4.2. Therefore, two calculi are roughly speaking doing the same job, except for the fact that the computation steps are significantly smaller in $\lambda^{\mathsf{Gtz}}_{®}$, hence the insight in the process of computation is more complete with the explicit resource control.

5.2 Simply typed $\lambda_{®}^{\mathsf{Gtz}}$-calculus

5.2.1 The system $\lambda_{®}^{\mathsf{Gtz}} \to$

The type assignment system that assigns simple types to $\lambda_{®}^{\mathsf{Gtz}}$-expressions, denoted by $\lambda_{®}^{\mathsf{Gtz}} \to$, is derived from the system $\lambda^{\mathsf{Gtz}} \to$. Therefore, the definitions of types and related basic notions coincide with definitions 4.4 and 4.5. With respect to the $\lambda^{\mathsf{Gtz}} \to$, the system $\lambda_{®}^{\mathsf{Gtz}} \to$ has four new rules, namely $(Weak_t)$, $(Cont_t)$, $(Weak_k)$ and $(Cont_k)$, for assigning types to the expressions containing explicit operators of weakening and contraction.

This system is constructed in a way that only (well-formed) expressions can receive a type, hence the typing rules with two premises, namely $(\to_L)$ and (Cut), are presented in a context-splitting, i.e. multiplicative rather than a context-sharing, i.e. additive style. In these rules, it is assumed that bases Γ and Γ' are disjoint sets of the basic type assignments, and Γ, Γ' represents their disjoint union.

The formulation of the (Ax) rule is also modified, since it does not include a basis Γ on the left-hand side of the sequent. This modification is in accordance with the Gentzen's sequent system *LJ* with explicit structural rules, and implies that all declared variables must actually appear in the expression.

The four newly introduced rules, the ones assigning arrow types to expressions with explicit operators for duplication and erasure on the surface position, are proposed in a natural way. This means that the two rules without the stoup correspond to the structural rules for contraction and weakening of the Gentzen's system $G1$ (presented in Figure 2.1) whereas the two rules with the stoup are their counterparts for assigning types to corresponding $\lambda_{®}^{\mathsf{Gtz}}$-contexts. Therefore, in these rules, types on the right hand sides of the turnstyle stay unchanged, while the bases are modified. In $(Cont_t)$ and $(Cont_k)$ rules, the two declarations of contracted variables are replaced by one declaration of a fresh variable. Notice that the types of all three variables coincide. On the other hand, in the rules $(Weak_t)$ and $(Weak_k)$ bases are enriched with a declaration of a fresh variable, whose type is arbitrary.

The type assignment system $\lambda_{®}^{\mathsf{Gtz}} \to$ is given in Figure 5.6.

The proposed system $\lambda_{®}^{\mathsf{Gtz}} \to$ satisfies the following basic features, which will be stated here without proofs, since their proofs are analogous (and in most of the cases simpler) than the proofs of the corresponding features of the system with intersection types $\lambda_{®}^{\mathsf{Gtz}}\cap$ which will be presented in details in the remaining subsections of this chapter.

Proposition 5.14 (Domain correspondence for $\lambda_{®}^{\mathsf{Gtz}} \to$)

(i) If $\Gamma \vdash t : \alpha$*, then* $Dom(\Gamma) = Fv(t)$.

$$\frac{}{x:\alpha\vdash x:\alpha}\ (Ax)$$

$$\frac{\Gamma,x:\alpha\vdash t:\beta}{\Gamma\vdash\lambda x.t:\alpha\to\beta}\ (\to_R)\qquad\frac{\Gamma\vdash t:\alpha\quad\Gamma';\beta\vdash k:\gamma}{\Gamma,\Gamma';\alpha\to\beta\vdash t::k:\gamma}\ (\to_L)$$

$$\frac{\Gamma\vdash t:\alpha\quad\Gamma';\alpha\vdash k:\beta}{\Gamma,\Gamma'\vdash tk:\beta}\ (Cut)\qquad\frac{\Gamma,x:\alpha\vdash t:\beta}{\Gamma;\alpha\vdash\widehat{x}.t:\beta}\ (Sel)$$

$$\frac{\Gamma,x:\alpha,y:\alpha\vdash t:\beta}{\Gamma,z:\alpha\vdash z<^x_y t:\beta}\ (Cont_t)\qquad\frac{\Gamma\vdash t:\beta}{\Gamma,x:\alpha\vdash x\odot t:\beta}\ (Weak_t)$$

$$\frac{\Gamma,x:\alpha,y:\alpha;\gamma\vdash k:\beta}{\Gamma,z:\alpha;\gamma\vdash z<^x_y k:\beta}\ (Cont_k)\qquad\frac{\Gamma;\gamma\vdash k:\beta}{\Gamma,x:\alpha;\gamma\vdash x\odot k:\beta}\ (Weak_k)$$

Figure 5.6: $\lambda^{\mathsf{Gtz}}_{®}\to$: simply typed $\lambda^{\mathsf{Gtz}}_{®}$-calculus

(ii) *If* $\Gamma;\alpha\vdash k:\beta$, *then* $Dom(\Gamma)=Fv(k)$.

Proof: By case analysis on the applied type assignment rule.□

Proposition 5.15 (Generation lemma for $\lambda^{\mathsf{Gtz}}_{®}\to$)

(i) $\Gamma\vdash\lambda x.t:\gamma$ *iff* $\gamma\equiv\alpha\to\beta$ *and* $\Gamma,x:\alpha\vdash t:\beta$.

(ii) $\Gamma;\delta\vdash t::k:\gamma$ *iff* $\Gamma=\Gamma',\Gamma''$ *and* $\delta\equiv\alpha\to\beta$ *and* $\Gamma';\beta\vdash k:\gamma$ *and* $\Gamma''\vdash t:\alpha$.

(iii) $\Gamma\vdash tk:\beta$ *iff* $\Gamma=\Gamma',\Gamma''$ *and there is a type* α *such that* $\Gamma'\vdash t:\alpha$ *and* $\Gamma'';\alpha\vdash k:\beta$.

(iv) $\Gamma;\alpha\vdash\widehat{x}.t:\beta$ *iff* $\Gamma,x:\alpha\vdash t:\beta$.

(v) $\Gamma,z:\alpha\vdash z<^x_y t:\beta$ *iff* $\Gamma,x:\alpha,y:\alpha\vdash t:\beta$.

(vi) $\Gamma,x:\alpha\vdash x\odot t:\beta$ *iff* $\Gamma\vdash t:\beta$.

(vii) $\Gamma,z:\alpha;\gamma\vdash z<^x_y k:\beta$ *iff* $\Gamma,x:\alpha,y:\alpha;\gamma\vdash k:\beta$.

(viii) $\Gamma,x:\alpha;\gamma\vdash x\odot k:\beta$ *iff* $\Gamma;\gamma\vdash k:\beta$.

Proof: The proof is straightforward because the type assignment system is syntax-directed, and relies on Proposition 5.14. □

Proposition 5.16 (Substitution lemma for $\lambda_{®}^{\mathsf{Gtz}}\rightarrow$)

(i) If $\Gamma, x:\beta \vdash t:\alpha$ *and* $\Delta \vdash T:\beta$, *then* $\Gamma, \Delta \vdash t[T/x]:\alpha$.

(ii) If $\Gamma, x:\beta;\gamma \vdash k:\alpha$ *and* $\Delta \vdash T:\beta$, *then* $\Gamma, \Delta;\gamma \vdash k[T/x]:\alpha$.

Proof: By mutual induction on the structure of terms and contexts. □

Proposition 5.17 (Append lemma for $\lambda_{®}^{\mathsf{Gtz}}\rightarrow$) *If* $\Gamma;\gamma \vdash k:\beta$ *and* $\Delta;\beta \vdash k':\alpha$, *then* $\Gamma, \Delta;\gamma \vdash k@k':\alpha$.

Proof: By induction on the structure of the context k. □

Proposition 5.18 (Subject equivalence for $\lambda_{®}^{\mathsf{Gtz}}\rightarrow$)

(i) If $\Gamma \vdash t:\alpha$ *and* $t \equiv t'$, *then* $\Gamma \vdash t':\alpha$.

(ii) If $\Gamma;\alpha \vdash k:\beta$ *and* $k \equiv k'$, *then* $\Gamma;\alpha \vdash k':\beta$.

Proof: By case analysis on the performed equivalence rule. □

Proposition 5.19 (Subject reduction for $\lambda_{®}^{\mathsf{Gtz}}\rightarrow$)

(i) If $\Gamma \vdash t:\alpha$ *and* $t \rightarrow t'$, *then* $\Gamma \vdash t':\alpha$.

(ii) If $\Gamma;\alpha \vdash k:\beta$ *and* $k \rightarrow k'$, *then* $\Gamma;\alpha \vdash k':\beta$.

Proof: By case analysis on the performed reduction rule. □

$$\boxed{\begin{array}{lrcl}
(\sigma_1) & T(\widehat{x}.x) & \to & T\\
(\sigma_2) & T(\widehat{x}.\lambda y.v) & \to & \lambda y.(T(\widehat{x}.v))\\
(\sigma_3) & T(\widehat{x}.uk) & \to & (T\widehat{x}.u)k, \quad \text{if } x \in Fv(u)\\
(\sigma_4) & T(\widehat{x}.x \odot u) & \to & Fv(T) \odot u\\
(\sigma_5) & T(\widehat{x}.x <^{x_1}_{x_2} u) & \to & Fv(T) <^{Fv(T_1)}_{Fv(T_2)} T_1(\widehat{x}_1.T_2(\widehat{x}_2.u))
\end{array}}$$

Figure 5.7: The group of σ-reductions in the variant of the $\lambda_{\circledR}^{\mathsf{Gtz}}$-calculus

5.2.2 Typeability $\Rightarrow$ SN in $\lambda_{\circledR}^{\mathsf{Gtz}} \to$

In this subsection, we prove that all $\lambda_{\circledR}^{\mathsf{Gtz}}$-expressions typeable in the system $\lambda_{\circledR}^{\mathsf{Gtz}} \to$ are terminating, which is the feature known as strong normalisation. The termination is proved by showing that the reduction relation on the set of the typeable $\lambda_{\circledR}^{\mathsf{Gtz}}$-expressions is included in a particular well-founded relation, which we define as the lexicographic product of five well-founded component relations.

The first of them is based on the mapping of the $\lambda_{\circledR}^{\mathsf{Gtz}}$-expressions into the terms of the λlxr-calculus, presented in Subsection 3.3. We show that the introduced mapping preserves types (Proposition 5.21), and that all reductions and equivalencies of the $\lambda_{\circledR}^{\mathsf{Gtz}}$-calculus can be simulated by the operational semantics of the λlxr-calculus (Theorem 5.27). The other four well founded orderings are based on the introduced measures (Definitions 5.29, 5.30, 5.31 and 5.33), designed to decrease the size of the particular $\lambda_{\circledR}^{\mathsf{Gtz}}$-expressions during the computation. This subsection contains the results from [35].

Since the proof of strong normalisation relies on the embedding of the $\lambda_{\circledR}^{\mathsf{Gtz}}$-expressions into the λlxr-terms, here we use a variant of the $\lambda_{\circledR}^{\mathsf{Gtz}}$-calculus in which the meta-operator of implicit substitution is omitted, and the substitution is incorporated directly into reduction rules. Instead of the definition of implicit substitution (Figure 5.4), the reduction σ from Figure 5.2 is now split into a family of reductions, namely $\sigma_1 - \sigma_5$, presented in Figure 5.7. This alternative approach was used in order to bring the $\lambda_{\circledR}^{\mathsf{Gtz}}$-calculus closer to the λlxr-calculus.

In what follows, we use the notation $\mathrm{T}_{\circledR}^{\mathsf{Gtz}}$ for the set of $\lambda_{\circledR}^{\mathsf{Gtz}}$-terms, $\mathrm{C}_{\circledR}^{\mathsf{Gtz}}$ for the set of $\lambda_{\circledR}^{\mathsf{Gtz}}$-contexts and Λ^{lxr} for the set of λlxr-terms. We also use indexed notations $\vdash_{\mathsf{lxr}}$, $\to_{\mathsf{lxr}}$ and $\equiv_{\mathsf{lxr}}$ for type assignment, reductions and equivalencies in the λlxr-calculus, whereas the plain notations denote the same objects in the $\lambda_{\circledR}^{\mathsf{Gtz}}$-calculus. More details about the λlxr-calculus can be found in Section 3.3.

Definition 5.20 *The mapping* $\lfloor\ \rfloor : \mathrm{T}_{\circledR}^{\mathsf{Gtz}} \to \Lambda^{\mathsf{lxr}}$ *is defined together with the aux-*

iliary mapping $\lfloor\ \rfloor_{\mathsf{k}} : \mathsf{C}_{®}^{\mathsf{Gtz}} \to (\Lambda^{\mathsf{lxr}} \to \Lambda^{\mathsf{lxr}})$ *in the following way:*

$$\begin{array}{rcl}
\lfloor x \rfloor & = & x \\
\lfloor \lambda x.t \rfloor & = & \lambda x.\lfloor t \rfloor \\
\lfloor x \odot t \rfloor & = & \mathcal{W}_x(\lfloor t \rfloor) \\
\lfloor x <^{y}_{z} t \rfloor & = & \mathcal{C}_x^{y|z}\,(\lfloor t \rfloor) \\
\\
\lfloor tk \rfloor & = & \lfloor k \rfloor_{\mathsf{k}}(\lfloor t \rfloor) \\
\\
\lfloor \widehat{x}.t \rfloor_{\mathsf{k}}(M) & = & \lfloor t \rfloor \langle x = M \rangle \\
\lfloor t :: k \rfloor_{\mathsf{k}}(M) & = & \lfloor k \rfloor_{\mathsf{k}}(M\lfloor t \rfloor) \\
\lfloor x \odot k \rfloor_{\mathsf{k}}(M) & = & \mathcal{W}_x(\lfloor k \rfloor_{\mathsf{k}}(M)) \\
\lfloor x <^{y}_{z} k \rfloor_{\mathsf{k}}(M) & = & \mathcal{C}_x^{y|z}\,(\lfloor k \rfloor_{\mathsf{k}}(M))
\end{array}$$

Now, we prove that the introduced mapping preserves types. In the sequel, the notation $\Lambda^{\Gamma \vdash_{\mathsf{lxr}} A}$ stands for the set $\{M \mid M \in \Lambda^{\mathsf{lxr}} \;\&\; \Gamma \vdash_{\mathsf{lxr}} M : A\}$.

Proposition 5.21 (Type preservation of $\lfloor\ \rfloor$)

(i) If $\Gamma \vdash t : \alpha$, *then* $\Gamma \vdash_{\mathsf{lxr}} \lfloor t \rfloor : \alpha$.

(ii) If $\Gamma; \alpha \vdash k : \beta$, *then* $\lfloor k \rfloor_{\mathsf{k}} : \Lambda^{\Gamma' \vdash_{\mathsf{lxr}} \alpha} \to \Lambda^{\Gamma,\Gamma' \vdash_{\mathsf{lxr}} \beta}$, *for some* Γ'.

Proof: The proposition is proved by simultaneous induction on derivations. We distinguish cases according to the last typing rule used.

- Cases (Ax), $(\to_R)$, $(Weak_t)$ and $(Cont_t)$ are easy, because the type assignment system of λlxr has exactly the same rules.
- Case (Sel): the derivation ends with the rule

  $$\frac{\Gamma, x : \alpha \vdash t : \beta}{\Gamma; \alpha \vdash \widehat{x}.t : \beta}\ (Sel)$$

  By IH we have that $\Gamma, x : \alpha \vdash_{\mathsf{lxr}} \lfloor t \rfloor : \beta$. For any $M \in \Lambda^{\mathsf{lxr}}$ such that $\Gamma' \vdash_{\mathsf{lxr}} M : \alpha$, for some Γ', we have

  $$\frac{\Gamma, x : \alpha \vdash_{\mathsf{lxr}} \lfloor t \rfloor : \beta \quad \Gamma' \vdash_{\mathsf{lxr}} M : \alpha}{\Gamma, \Gamma' \vdash_{\mathsf{lxr}} \lfloor t \rfloor \langle x = M \rangle : \beta}\ (Subs)$$

  Since $\lfloor t \rfloor \langle x = M \rangle = \lfloor \widehat{x}.t \rfloor_{\mathsf{k}}(M)$, we conclude that $\lfloor \widehat{x}.t \rfloor_{\mathsf{k}} : \Lambda^{\Gamma' \vdash_{\mathsf{lxr}} \alpha} \to \Lambda^{\Gamma,\Gamma' \vdash_{\mathsf{lxr}} \beta}$.

- Case $(\rightarrow_L)$: the derivation ends with the rule

$$\frac{\Gamma \vdash t : \alpha \quad \Gamma' ; \beta \vdash k : \gamma}{\Gamma, \Gamma' ; \alpha \rightarrow \beta \vdash t :: k : \gamma}\ (\rightarrow_L)$$

By IH we have that $\Gamma \vdash_{\mathsf{lxr}} \lfloor t \rfloor : \alpha$. For any $M \in \Lambda^{\mathsf{lxr}}$ such that $\Gamma'' \vdash_{\mathsf{lxr}} M : \alpha \rightarrow \beta$, we have

$$\frac{\Gamma'' \vdash_{\mathsf{lxr}} M : \alpha \rightarrow \beta \quad \Gamma \vdash_{\mathsf{lxr}} \lfloor t \rfloor : \alpha}{\Gamma, \Gamma'' \vdash_{\mathsf{lxr}} M \lfloor t \rfloor : \beta}\ (App)$$

From the right-hand side premise in the $(\rightarrow_L)$ rule, by IH, we get that $\lfloor k \rfloor_{\mathrm{k}}$ is the function with the scope $\lfloor k \rfloor_{\mathrm{k}} : \Lambda^{\Gamma''' \vdash_{\mathsf{lxr}} \beta} \rightarrow \Lambda^{\Gamma''', \Gamma' \vdash_{\mathsf{lxr}} \gamma}$, for some Γ'''. For $\Gamma''' \equiv \Gamma, \Gamma''$ and by taking $M \lfloor t \rfloor$ as the argument of the function $\lfloor k \rfloor_{\mathrm{k}}$, we get $\Gamma, \Gamma', \Gamma'' \vdash_{\mathsf{lxr}} \lfloor k \rfloor_{\mathrm{k}}(M \lfloor t \rfloor) : \gamma$. Since $\lfloor k \rfloor_{\mathrm{k}}(M \lfloor t \rfloor) = \lfloor t :: k \rfloor_{\mathrm{k}}(M)$, we have that $\Gamma, \Gamma', \Gamma'' \vdash_{\mathsf{lxr}} \lfloor t :: k \rfloor_{\mathrm{k}}(M) : \gamma$. This holds for any M of the appropriate type, yielding $\lfloor t :: k \rfloor_{\mathrm{k}} : \Lambda^{\Gamma'' \vdash_{\mathsf{lxr}} \alpha \rightarrow \beta} \rightarrow \Lambda^{\Gamma, \Gamma', \Gamma'' \vdash_{\mathsf{lxr}} \gamma}$, which is exactly what we wanted to prove.

- Case (Cut): the derivation ends with the rule

$$\frac{\Gamma \vdash t : \alpha \quad \Gamma' ; \alpha \vdash k : \beta}{\Gamma, \Gamma' \vdash tk : \beta}\ (Cut)$$

By IH we have that $\Gamma \vdash_{\mathsf{lxr}} \lfloor t \rfloor : \alpha$ and $\lfloor k \rfloor_{\mathrm{k}} : \Lambda^{\Gamma'' \vdash_{\mathsf{lxr}} \alpha} \rightarrow \Lambda^{\Gamma', \Gamma'' \vdash_{\mathsf{lxr}} \beta}$ for some Γ''. Hence, for any $M \in \Lambda^{\mathsf{lxr}}$ such that $\Gamma'' \vdash_{\mathsf{lxr}} M : \alpha$, it holds that $\Gamma', \Gamma'' \vdash_{\mathsf{lxr}} \lfloor k \rfloor_{\mathrm{k}}(M) : \beta$. By taking $M \equiv \lfloor t \rfloor$ and $\Gamma'' \equiv \Gamma$, we get $\Gamma, \Gamma' \vdash_{\mathsf{lxr}} \lfloor k \rfloor_{\mathrm{k}}(\lfloor t \rfloor) : \beta$. But $\lfloor k \rfloor_{\mathrm{k}}(\lfloor t \rfloor) = \lfloor tk \rfloor$, so the proof is done.

- Case $(Weak_k)$: the derivation ends with the rule

$$\frac{\Gamma ; \gamma \vdash k : \beta}{\Gamma, x : \alpha ; \gamma \vdash x \odot k : \beta}\ (Weak_k)$$

By IH we have that $\lfloor k \rfloor_{\mathrm{k}}$ is the function with the scope $\lfloor k \rfloor_{\mathrm{k}} : \Lambda^{\Gamma' \vdash_{\mathsf{lxr}} \gamma} \rightarrow \Lambda^{\Gamma, \Gamma' \vdash_{\mathsf{lxr}} \beta}$ for some Γ', meaning that for each $M \in \Lambda^{\mathsf{lxr}}$ such that $\Gamma' \vdash_{\mathsf{lxr}} M : \gamma$ it holds that $\Gamma, \Gamma' \vdash_{\mathsf{lxr}} \lfloor k \rfloor_{\mathrm{k}}(M) : \beta$. Now, we can apply $(Weak)$ rule:

$$\frac{\Gamma, \Gamma' \vdash \lfloor k \rfloor_{\mathrm{k}}(M) : \beta}{\Gamma, \Gamma', x : \alpha \vdash \mathcal{W}_x(\lfloor k \rfloor_{\mathrm{k}}(M)) : \beta}\ (Weak)$$

Since $\mathcal{W}_x(\lfloor k \rfloor_{\mathrm{k}}(M)) = \lfloor x \odot k \rfloor_{\mathrm{k}}(M)$, this means that $\lfloor x \odot k \rfloor_{\mathrm{k}} : \Lambda^{\Gamma' \vdash_{\mathsf{lxr}} \gamma} \rightarrow \Lambda^{\Gamma, \Gamma', x : A \vdash_{\mathsf{lxr}} \beta}$, which is exactly what we wanted to prove.

- Case $(Cont_k)$: the derivation ends with the rule

$$\frac{\Gamma, x:\alpha, y:\alpha;\gamma \vdash k:\beta}{\Gamma, z:\alpha;\gamma \vdash z <^{x}_{y} k:\beta}\ (Cont_k)$$

By IH we have that $\lfloor k \rfloor_{\mathsf{k}}$ is the function with the scope $\lfloor k \rfloor_{\mathsf{k}} : \Lambda^{\Gamma' \vdash_{\mathsf{lxr}} \gamma} \to \Lambda^{\Gamma, x:\alpha, y:\alpha, \Gamma' \vdash_{\mathsf{lxr}} \beta}$ for some Γ', meaning that for each $M \in \Lambda^{\mathsf{lxr}}$ such that $\Gamma' \vdash_{\mathsf{lxr}} M:\gamma$ it holds that $\Gamma, x:\alpha, y:\alpha, \Gamma' \vdash_{\mathsf{lxr}} \lfloor k \rfloor_{\mathsf{k}}(M):\beta$. Now, we can apply $(Cont)$ rule:

$$\frac{\Gamma, x:\alpha, y:\alpha, \Gamma' \vdash \lfloor k \rfloor_{\mathsf{k}}(M):\beta}{\Gamma, z:\alpha, \Gamma' \vdash \mathcal{C}_{z}^{x|y}\ (\lfloor k \rfloor_{\mathsf{k}}(M)):\beta}\ (Cont)$$

but $\mathcal{C}_{z}^{x|y}\ (\lfloor k \rfloor_{\mathsf{k}}(M)) = \lfloor z <^{x}_{y} k \rfloor_{\mathsf{k}}(M)$, so $\lfloor z <^{x}_{y} k \rfloor_{\mathsf{k}}$ is the function with scope $\lfloor z <^{x}_{y} k \rfloor_{\mathsf{k}} : \Lambda^{\Gamma' \vdash_{\mathsf{lxr}} \gamma} \to \Lambda^{\Gamma, \Gamma', z:\alpha \vdash_{\mathsf{lxr}} \beta}$, which completes the proof. □

For the given encoding $\lfloor\ \rfloor$, we will now show that each of the $\lambda_{®}^{\mathsf{Gtz}}$ reduction or equivalence steps can be simulated by λlxr reductions or equivalences. In order to do so, we first prove the following lemmas.

Lemma 5.22 *If $M \to_{\mathsf{lxr}} M'$, then $\lfloor k \rfloor_{\mathsf{k}}(M) \to_{\mathsf{lxr}} \lfloor k \rfloor_{\mathsf{k}}(M')$.*

Proof: By induction on the structure of k.

- Basic case: $k \equiv \widehat{x}.t$.
  $M \to_{\mathsf{lxr}} M'$ implies $t\langle x = M\rangle \to_{\mathsf{lxr}} t\langle x = M'\rangle$. Since $\lfloor \widehat{x}.t \rfloor_{\mathsf{k}}(M) = \lfloor t \rfloor \langle x = M\rangle$, the statement is proved.

- Case: $k \equiv t :: k'$.
  $\lfloor t :: k' \rfloor_{\mathsf{k}}(M) = \lfloor k' \rfloor_{\mathsf{k}}(M\lfloor t \rfloor)$, hence the proof is done by using IH on k' and the fact that reductions in λlxr are context closed.

- Cases $k \equiv x \odot k'$ and $k \equiv x <^{x_1}_{x_2} k'$ are analogous. □

Since the given mapping encodes $\lambda_{®}^{\mathsf{Gtz}}$-contexts into functions, it is natural to show that $k@k'$ is interpreted as the composition of the corresponding encodings of k' and k, respectively.

Lemma 5.23 *If $M \in \Lambda^{\mathsf{lxr}}$ and $k, k' \in \mathsf{C}_{®}^{\mathsf{Gtz}}$, then $\lfloor k@k' \rfloor_{\mathsf{k}}(M) = \lfloor k' \rfloor_{\mathsf{k}} \circ \lfloor k \rfloor_{\mathsf{k}}(M)$.*

Proof: By induction on the structure of k.

- Basic case: $k \equiv \widehat{x}.t$.
  By definitions of @ and $\lfloor\ \rfloor$ we have

$$\lfloor \widehat{x}.t @ k' \rfloor_{\mathrm{k}}(M) \triangleq \lfloor \widehat{x}.(tk') \rfloor_{\mathrm{k}}(M) = \lfloor tk' \rfloor \langle x = M \rangle = \lfloor k' \rfloor_{\mathrm{k}}(\lfloor t \rfloor \langle x = M \rangle) =$$

$$\lfloor k' \rfloor_{\mathrm{k}}(\lfloor \widehat{x}.t \rfloor_{\mathrm{k}}(M)) = \lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor \widehat{x}.t \rfloor_{\mathrm{k}}(M) = \lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor k \rfloor_{\mathrm{k}}(M).$$

- Case: $k \equiv t :: k''$.

$$\lfloor (t :: k'') @ k' \rfloor_{\mathrm{k}}(M) \triangleq \lfloor t :: (k'' @ k') \rfloor_{\mathrm{k}}(M) = \lfloor k'' @ k' \rfloor_{\mathrm{k}}(M \lfloor t \rfloor) =_{IH}$$

$$= \lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor k'' \rfloor_{\mathrm{k}}(M \lfloor t \rfloor) = \lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor t :: k'' \rfloor_{\mathrm{k}}(M) = \lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor k \rfloor_{\mathrm{k}}(M).$$

- Case: $k \equiv x \odot k''$.

$$\lfloor (x \odot k'') @ k' \rfloor_{\mathrm{k}}(M) \triangleq \lfloor x \odot (k'' @ k') \rfloor_{\mathrm{k}}(M) = \mathcal{W}_x(\lfloor (k'' @ k') \rfloor_{\mathrm{k}}(M)) =_{IH}$$

$$= \mathcal{W}_x(\lfloor (\lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor k'' \rfloor_{\mathrm{k}}) \rfloor_{\mathrm{k}}(M)) = \lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor x \odot k'' \rfloor_{\mathrm{k}}(M) = \lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor k \rfloor_{\mathrm{k}}(M).$$

- Case $k \equiv x <^{x_1}_{x_2} k'$ is analogous. □

The proofs of the following two lemmas, stating propagation of contraction and extraction of weakening for the encoded $\lambda_{\circledR}^{\mathsf{Gtz}}$-contexts, follow the same pattern, thus they are presented without the proofs.

Lemma 5.24 *If $x, y \notin k$, then $\mathcal{C}_z^{x|y}\,((\lfloor k \rfloor_{\mathrm{k}}(M))) \to_{\mathsf{lxr}} \lfloor k \rfloor_{\mathrm{k}}(\mathcal{C}_z^{x|y}\,(M))$.*

Proof: By induction on the structure of k. □

Lemma 5.25 $\lfloor k \rfloor_{\mathrm{k}}(\mathcal{W}_x(M)) \to_{\mathsf{lxr}} \mathcal{W}_x(\lfloor k \rfloor_{\mathrm{k}}(M))$.

Proof: By induction on the structure of k. □

The remaining technical proposition explains the interaction of an encoded $\lambda_{\circledR}^{\mathsf{Gtz}}$-context and the explicit substitution in the target calculus.

Lemma 5.26 *If $x \notin k$, then $(\lfloor k \rfloor_{\mathrm{k}}(M))\langle x = N \rangle \to_{\mathsf{lxr}} \lfloor k \rfloor_{\mathrm{k}}(M \langle x = N \rangle)$.*

Proof: By induction on the structure of k.

- Basic case: $k \equiv \widehat{y}.t$.

  $$(\lfloor \widehat{y}.t \rfloor_{\mathsf{k}}(M))\langle x = N\rangle = (\lfloor t \rfloor \langle y = M\rangle)\langle x = N\rangle.$$

  $x \notin k$ implies $x \in M$, hence $x \notin Fv(t) \setminus \{y\}$ so we can apply $(comp)$ reduction in $\lambda\mathsf{lxr}$:

  $$(\lfloor t \rfloor \langle y = M\rangle)\langle x = N\rangle \to_{comp} t\langle y = M\langle x = N\rangle\rangle = \lfloor \widehat{y}.t \rfloor_{\mathsf{k}}(M\langle x = N\rangle).$$

- Case: $k \equiv t :: k''$.

  $$(\lfloor t :: k' \rfloor_{\mathsf{k}}(M))\langle x = N\rangle = (\lfloor k' \rfloor_{\mathsf{k}}(M\lfloor t \rfloor))\langle x = N\rangle$$

  which by IH on k' followed by the reduction rule $(App1)$ (which we can apply since $x \notin k$ implies $x \in M$) yields

  $$\lfloor k' \rfloor_{\mathsf{k}}((M\lfloor t \rfloor)\langle x{=}N\rangle) \to_{App1} \lfloor k' \rfloor_{\mathsf{k}}((M\langle x{=}N\rangle)\lfloor t \rfloor) = \lfloor t :: k' \rfloor_{\mathsf{k}}(M\langle x{=}N\rangle).\Box$$

Now we can prove one of the central propositions for the proof of strong normalization, stating that the reduction and equivalence rules of $\lambda_{\circledR}^{\mathsf{Gtz}}$-calculus can be simulated by the reduction and equivalence rules of Kesner and Lengrand's $\lambda\mathsf{lxr}$-calculus, given in Figures 3.7 and 3.8.

In the following proposition, we use $\twoheadrightarrow$ to denote the reflexive and transitive closure of a reduction relation $\to$.

Theorem 5.27 (Simulation of operational semantics)

(i) If a $\lambda_{\circledR}^{\mathsf{Gtz}}$-term $M \to M'$, then $\lfloor M \rfloor \twoheadrightarrow_{\mathsf{lxr}} \lfloor M' \rfloor$.

(ii) If a $\lambda_{\circledR}^{\mathsf{Gtz}}$-context $K \to K'$, then $\lfloor K \rfloor_{\mathsf{k}}(M) \twoheadrightarrow_{\mathsf{lxr}} \lfloor K' \rfloor_{\mathsf{k}}(M)$, for any $M \in \Lambda^{\mathsf{lxr}}$.

Proof: Without losing generality, we prove the statement only for the outermost reductions.

(β) $(\lambda x.t)(u :: k) \to u(\widehat{x}.tk)$.

On the one hand we have
$\lfloor M \rfloor = \lfloor (\lambda x.t)(u :: k) \rfloor = \lfloor u :: k \rfloor_{\mathsf{k}}(\lfloor \lambda x.t \rfloor) = \lfloor k \rfloor_{\mathsf{k}}((\lambda x.\lfloor t \rfloor)\lfloor u \rfloor)$.

On the other hand,
$\lfloor M' \rfloor = \lfloor u(\widehat{x}.tk) \rfloor = \lfloor \widehat{x}.tk \rfloor_{\mathsf{k}}(\lfloor u \rfloor) = \lfloor tk \rfloor \langle x = \lfloor u \rfloor\rangle =$
$(\lfloor k \rfloor_{\mathsf{k}}(\lfloor t \rfloor))\langle x = \lfloor u \rfloor\rangle = \lfloor k \rfloor_{\mathsf{k}}(\lfloor t \rfloor \langle x = \lfloor u \rfloor\rangle)$.

The last equality follows from the definition of terms, since we know that $x \in t$. So, $\lfloor M \rfloor \twoheadrightarrow_{\mathsf{lxr}} \lfloor M' \rfloor$ by Lemma 5.22 and reduction (B).

(π) $(tk)k' \to t(k@k')$.

$\lfloor M\rfloor = \lfloor (tk)k'\rfloor = \lfloor k'\rfloor_{\mathsf{k}}(\lfloor tk\rfloor) = \lfloor k'\rfloor_{\mathsf{k}}(\lfloor k\rfloor_{\mathsf{k}}(\lfloor t\rfloor))$.

$\lfloor M'\rfloor = \lfloor t(k@k')\rfloor = \lfloor k@k'\rfloor_{\mathsf{k}}(\lfloor t\rfloor)$.

Applying Lemma 5.23 we get that $\lfloor M\rfloor = \lfloor M'\rfloor$ in $\lambda\mathsf{lxr}$.

(μ) $\widehat{x}.xk \to k$.

This reduction reduces context to context, so we have:

$\lfloor \widehat{x}.xk\rfloor_{\mathsf{k}}(M) = \lfloor xk\rfloor\langle x = M\rangle = (\lfloor k\rfloor_{\mathsf{k}}(x))\langle x = M\rangle = \lfloor k\rfloor_{\mathsf{k}}(M)$.

The last equality holds due to the definition of $\lambda^{\mathsf{Gtz}}_{®}$-expressions, because we know that $x \notin Fv(k)$.

(σ_1) $T(\widehat{x}.x) \to T$.

On the one hand we have
$\lfloor M\rfloor = \lfloor T(\widehat{x}.x)\rfloor = \lfloor \widehat{x}.x\rfloor_{\mathsf{k}}(\lfloor T\rfloor) = \lfloor x\rfloor\langle x = \lfloor T\rfloor\rangle = \lfloor T\rfloor$.

On the other hand, $\lfloor M'\rfloor = \lfloor T\rfloor$, so $\lfloor M\rfloor = \lfloor M'\rfloor$ in $\lambda\mathsf{lxr}$.

(σ_2) $T(\widehat{x}.\lambda y.v) \to \lambda y.(T(\widehat{x}.v))$.

On the one hand we have

$\lfloor M\rfloor = \lfloor T(\widehat{x}.\lambda y.v)\rfloor = \lfloor \widehat{x}.\lambda y.v\rfloor_{\mathsf{k}}(\lfloor T\rfloor) = \lfloor \lambda y.v\rfloor\langle x = \lfloor T\rfloor\rangle =$

$(\lambda y.\lfloor v\rfloor)\langle x = \lfloor T\rfloor\rangle$.

On the other hand,

$\lfloor M'\rfloor = \lfloor \lambda y.(T(\widehat{x}.v))\rfloor = \lambda y.\lfloor \widehat{x}.v\rfloor_{\mathsf{k}}(\lfloor T\rfloor) = \lambda y.\lfloor v\rfloor\langle x = \lfloor T\rfloor\rangle$.

So $\lfloor M\rfloor \to_{\mathsf{lxr}} \lfloor M'\rfloor$ by the rule (Abs) in $\lambda\mathsf{lxr}$.

(σ_3) $T(\widehat{x}.uk) \to (T(\widehat{x}.u))k$, if $x \in u$.

On the one hand,
$\lfloor M\rfloor = \lfloor T(\widehat{x}.uk)\rfloor = \lfloor \widehat{x}.uk\rfloor_{\mathsf{k}}(\lfloor T\rfloor) = \lfloor uk\rfloor\langle x := \lfloor T\rfloor\rangle = (\lfloor k\rfloor_{\mathsf{k}}(\lfloor u\rfloor))\langle x := T\rangle$.

On the other hand,
$\lfloor M'\rfloor = \lfloor (T(\widehat{x}.u))k\rfloor = \lfloor k\rfloor_{\mathsf{k}}(\lfloor T(\widehat{x}.u)\rfloor) = \lfloor k\rfloor_{\mathsf{k}}(\lfloor \widehat{x}.u\rfloor_{\mathsf{k}}(\lfloor T\rfloor)) =$

$\lfloor k\rfloor_{\mathsf{k}}(\lfloor u\rfloor\langle x := \lfloor T\rfloor\rangle)$.

$\lfloor M\rfloor \to_{\mathsf{lxr}} \lfloor M'\rfloor$ by Lemma 5.26.

(σ_4) $T(\widehat{x}.x \odot t) \to Fv(T) \odot t$.

On the one hand
$\lfloor M\rfloor = \lfloor T(\widehat{x}.x \odot t)\rfloor = \lfloor \widehat{x}.x \odot t\rfloor_{\mathsf{k}}(\lfloor T\rfloor) = \lfloor x \odot t\rfloor\langle x := \lfloor T\rfloor\rangle =$

$\mathcal{W}_x\lfloor t\rfloor\langle x := \lfloor T\rfloor\rangle = \mathcal{W}_{Fv(T)}(\lfloor t\rfloor)$.

On the other hand,

$\lfloor M'\rfloor = \lfloor Fv(T)\odot t\rfloor = \mathcal{W}_{Fv(T)}(\lfloor t\rfloor)$.

So $\lfloor M\rfloor \to_{\mathsf{lxr}} \lfloor M'\rfloor$ by the rule $(Weak1)$ in $\lambda\mathsf{lxr}$.

(σ_5) $T(\widehat{x}.x <^{x_1}_{x_2} u) \to Fv(T) <^{Fv(T_1)}_{Fv(T_2)} T_1(\widehat{x}_1.T_2(\widehat{x}_2.u))$.

On the one hand

$$\begin{array}{lll}
\lfloor M\rfloor & = & \lfloor T(\widehat{x}.x <^{x_1}_{x_2} u)\rfloor \\
& = & \lfloor \widehat{x}.x <^{x_1}_{x_2} u\rfloor_{\mathsf{k}}(\lfloor T\rfloor) \\
& = & \lfloor x <^{x_1}_{x_2} u\rfloor\langle x := \lfloor T\rfloor\rangle \\
& = & \mathcal{C}_x^{x_1|x_2}(\lfloor u\rfloor)\langle x := \lfloor T\rfloor\rangle \\
& \xrightarrow{Cont} & \mathcal{C}_{Fv(T)}^{Fv(T_1)|Fv(T_2)}(\lfloor u\rfloor)\langle x_1 := \lfloor T_1\rfloor\rangle\langle x_2 := \lfloor T_2\rfloor\rangle \\
& \equiv_{Ps} & \mathcal{C}_{Fv(T)}^{Fv(T_1)|Fv(T_2)}(\lfloor u\rfloor)\langle x_2 := \lfloor T_2\rfloor\rangle\langle x_1 := \lfloor T_1\rfloor\rangle, \text{ when } x_1,x_2 \in u.
\end{array}$$

On the other hand,

$$\begin{array}{lll}
\lfloor M'\rfloor & = & \lfloor Fv(T) <^{Fv(T_1)}_{Fv(T_2)} T_1(\widehat{x}_1.T_2(\widehat{x}_2.u))\rfloor \\
& = & \mathcal{C}_{Fv(T)}^{Fv(T_1)|Fv(T_2)}(\lfloor T_1(\widehat{x}_1.T_2(\widehat{x}_2.u))\rfloor) \\
& = & \mathcal{C}_{Fv(T)}^{Fv(T_1)|Fv(T_2)}(\lfloor \widehat{x}_1.T_2(\widehat{x}_2.u))\rfloor_{\mathsf{k}})(\lfloor T_1\rfloor) \\
& = & \mathcal{C}_{Fv(T)}^{Fv(T_1)|Fv(T_2)}(\lfloor T_2(\widehat{x}_2.u))\rfloor)\langle x_1 := \lfloor T_1\rfloor\rangle \\
& = & \mathcal{C}_{Fv(T)}^{Fv(T_1)|Fv(T_2)}(\lfloor u\rfloor)\langle x_2 := \lfloor T_2\rfloor\rangle\langle x_1 := \lfloor T_1\rfloor\rangle.
\end{array}$$

(γ_1) $x <^{x_1}_{x_2} (\lambda y.t) \to \lambda y.x <^{x_1}_{x_2} t$.

$\lfloor M\rfloor = \lfloor x <^{x_1}_{x_2} (\lambda y.t)\rfloor = \mathcal{C}_x^{x_1|x_2}((\lambda y.\lfloor t\rfloor))$.

$\lfloor M'\rfloor = \lfloor \lambda y.x <^{x_1}_{x_2} t\rfloor = \lambda y.\mathcal{C}_x^{x_1|x_2}(\lfloor t\rfloor)$,

hence $\lfloor M\rfloor \to_{\mathsf{lxr}} \lfloor M'\rfloor$ by the rule $(Cabs)$ in $\lambda\mathsf{lxr}$.

(γ_2) $x <^{x_1}_{x_2} (tk) \to (x <^{x_1}_{x_2} t)k$, if $x_1,x_2 \in t$.

$\lfloor M\rfloor = \lfloor x <^{x_1}_{x_2} (tk)\rfloor = \mathcal{C}_x^{x_1|x_2}(\lfloor k\rfloor_{\mathsf{k}}(\lfloor t\rfloor))$.

$\lfloor M'\rfloor = \lfloor (x <^{x_1}_{x_2} t)k\rfloor = \lfloor k\rfloor_{\mathsf{k}}(\mathcal{C}_x^{x_1|x_2}(\lfloor t\rfloor))$.

Since $x_1,x_2 \notin k$, we apply Lemma 5.59 and conclude that $\lfloor M\rfloor \to_{\mathsf{lxr}} \lfloor M'\rfloor$ in $\lambda\mathsf{lxr}$.

(γ_3) $x <_{x_2}^{x_1} (tk) \to t(x <_{x_2}^{x_1} k)$, if $x_1, x_2 \in k$.

$\lfloor M \rfloor = \lfloor x <_{x_2}^{x_1} (tk) \rfloor = \mathcal{C}_x^{x_1|x_2} (\lfloor k \rfloor_{\mathsf{k}}(\lfloor t \rfloor))$.

$\lfloor M' \rfloor = \lfloor t(x <_{x_2}^{x_1} k) \rfloor = \lfloor x <_{x_2}^{x_1} k \rfloor_{\mathsf{k}}(\lfloor t \rfloor) = \mathcal{C}_x^{x_1|x_2} (\lfloor k \rfloor_{\mathsf{k}}(\lfloor t \rfloor))$,

so $\lfloor M \rfloor = \lfloor M' \rfloor$ in $\lambda\mathsf{lxr}$.

(γ_4) $x <_{x_2}^{x_1} (\widehat{y}.t) \to \widehat{y}.(x <_{x_2}^{x_1} t)$.

$\lfloor K \rfloor_{\mathsf{k}}(M) = \lfloor x <_{x_2}^{x_1} (\widehat{y}.t) \rfloor_{\mathsf{k}}(M) = \mathcal{C}_x^{x_1|x_2} ((\widehat{y}.t)(M)) = \mathcal{C}_x^{x_1|x_2} (\lfloor t \rfloor \langle y = M \rangle)$.

On the other hand,
$\lfloor K' \rfloor_{\mathsf{k}}(M) = \lfloor \widehat{y}.(x <_{x_2}^{x_1} t) \rfloor_{\mathsf{k}}(M) = x <_{x_2}^{x_1} t \langle y = M \rangle = \mathcal{C}_x^{x_1|x_2} (\lfloor t \rfloor) \langle y = M \rangle$.

In $\lambda\mathsf{lxr}$ $\lfloor K \rfloor_{\mathsf{k}}(M) \equiv \lfloor K' \rfloor_{\mathsf{k}}(M)$ by equality $(\equiv_{Pcs})$.

(γ_5) $x <_{x_2}^{x_1} (t :: k) \to (x <_{x_2}^{x_1} t) :: k$, if $x_1, x_2 \in t$.

$\lfloor K \rfloor_{\mathsf{k}}(M) = \lfloor x <_{x_2}^{x_1} (t :: k) \rfloor_{\mathsf{k}}(M) = \mathcal{C}_x^{x_1|x_2} ((\lfloor t :: k \rfloor_{\mathsf{k}}(M))) = \mathcal{C}_x^{x_1|x_2} ((\lfloor k \rfloor_{\mathsf{k}}(M \lfloor t \rfloor)))$.

$\lfloor K' \rfloor_{\mathsf{k}}(M) = \lfloor (x <_{x_2}^{x_1} t) :: k \rfloor_{\mathsf{k}}(M) = \lfloor k \rfloor_{\mathsf{k}}(M \mathcal{C}_x^{x_1|x_2} (\lfloor t \rfloor)$.

$x_1, x_2 \in t$ implies $x_1, x_2 \in M \lfloor t \rfloor$ so we can apply Lemma 5.59, followed by the reduction $(CApp2)$ and conclude

$\mathcal{C}_x^{x_1|x_2} ((\lfloor k \rfloor_{\mathsf{k}}(M \lfloor t \rfloor))) \to_{lxr} \lfloor k \rfloor_{\mathsf{k}}(\mathcal{C}_x^{x_1|x_2} ((M \lfloor t \rfloor))) \to_{lxr} \lfloor k \rfloor_{\mathsf{k}}(M \mathcal{C}_x^{x_1|x_2} (\lfloor t \rfloor)$.

(γ_6) $x <_{x_2}^{x_1} (t :: k) \to t :: (x <_{x_2}^{x_1} k)$, if $x_1, x_2 \in k$.

$\lfloor K \rfloor_{\mathsf{k}}(M) = \lfloor x <_{x_2}^{x_1} (t :: k) \rfloor_{\mathsf{k}}(M) = \mathcal{C}_x^{x_1|x_2} (\lfloor t :: k \rfloor_{\mathsf{k}}(M)) = \mathcal{C}_x^{x_1|x_2} (\lfloor k \rfloor_{\mathsf{k}}(M \lfloor t \rfloor))$.

On the other hand,
$\lfloor K' \rfloor_{\mathsf{k}}(M) = \lfloor t :: (x <_{x_2}^{x_1} k) \rfloor_{\mathsf{k}}(M) = \lfloor x <_{x_2}^{x_1} k) \rfloor_{\mathsf{k}}(M \lfloor t \rfloor) = \mathcal{C}_x^{x_1|x_2} (\lfloor k \rfloor_{\mathsf{k}}(M \lfloor t \rfloor))$.

So $\lfloor K \rfloor_{\mathsf{k}}(M) = \lfloor K' \rfloor_{\mathsf{k}}(M)$ in $\lambda\mathsf{lxr}$.

($\gamma\omega_1$) $x <_{x_2}^{x_1} (y \odot e) \to y \odot (x <_{x_2}^{x_1} e)$.

In both cases, $e \equiv t$ and $e \equiv k$, we get that $\lfloor M \rfloor \to_{\mathsf{lxr}} \lfloor M' \rfloor$, i.e. $\lfloor K \rfloor_{\mathsf{k}}(M) \to_{\mathsf{lxr}} \lfloor K' \rfloor_{\mathsf{k}}(M)$, by the rule $(Cross)$.

($\gamma\omega_2$) $x <_{x_2}^{x_1} (x_1 \odot e) \to e\{x_2 := x\}$.

In both cases, $e \equiv t$ and $e \equiv k$, we get that $\lfloor M \rfloor \to_{\mathsf{lxr}} \lfloor M' \rfloor$, i.e. $\lfloor K \rfloor_{\mathsf{k}}(M) \to_{\mathsf{lxr}} \lfloor K' \rfloor_{\mathsf{k}}(M)$, by the rule $(Merge)$.

We used the fact that $\lfloor t\{x := y\} \rfloor = \mathcal{R}_y^x \lfloor t \rfloor$ and $\lfloor k\{x := y\} \rfloor_{\mathsf{k}}(M) = \mathcal{R}_y^x(\lfloor k \rfloor_{\mathsf{k}}(M))$.

(ω_1) $\lambda x.(y \odot t) \to y \odot (\lambda x.t), \quad x \neq y.$

$\lfloor M \rfloor = \lfloor \lambda x.(y \odot t) \rfloor = \lambda x.\mathcal{W}_y(\lfloor t \rfloor).$

$\lfloor M' \rfloor = \lfloor y \odot (\lambda x.t) \rfloor = \mathcal{W}_y(\lambda x.\lfloor t \rfloor).$

So $\lfloor M \rfloor \to_{\mathsf{lxr}} \lfloor M' \rfloor$ by the rule $(WAbs)$ in $\lambda\mathsf{lxr}$.

(ω_2) $(y \odot t)k \to y \odot (tk).$

$\lfloor M \rfloor = \lfloor (y \odot t)k \rfloor = \lfloor k \rfloor_{\mathrm{k}}(\mathcal{W}_y(\lfloor t \rfloor)).$

$\lfloor M' \rfloor = \lfloor y \odot (tk) \rfloor = \mathcal{W}_y(\lfloor k \rfloor_{\mathrm{k}}(\lfloor t \rfloor)).$

So $\lfloor M \rfloor \to_{\mathsf{lxr}} \lfloor M' \rfloor$ by Lemma 5.59.

(ω_3) $t(y \odot k) \to y \odot (tk).$

$\lfloor M \rfloor = \lfloor t(y \odot k) \rfloor = \lfloor y \odot k \rfloor_{\mathrm{k}}(\lfloor t \rfloor) = \mathcal{W}_y(\lfloor k \rfloor_{\mathrm{k}}(\lfloor t \rfloor)).$

$\lfloor M' \rfloor = \lfloor y \odot (tk) \rfloor = \mathcal{W}_y(\lfloor k \rfloor_{\mathrm{k}}(\lfloor t \rfloor)).$

So $\lfloor M \rfloor = \lfloor M' \rfloor$ in $\lambda\mathsf{lxr}$.

(ω_4) $\widehat{x}.(y \odot t) \to y \odot (\widehat{x}.t), \quad x \neq y.$

$\lfloor K \rfloor_{\mathrm{k}}(M) = \lfloor \widehat{x}.(y \odot t) \rfloor_{\mathrm{k}}(M) = \lfloor y \odot t \rfloor \langle x = M \rangle = \mathcal{W}_y(\lfloor t \rfloor)\langle x = M \rangle.$

$\lfloor K' \rfloor_{\mathrm{k}}(M) = \lfloor y \odot (\widehat{x}.t) \rfloor_{\mathrm{k}}(M) = \mathcal{W}_y(\lfloor \widehat{x}.t \rfloor_{\mathrm{k}}(M)) = \mathcal{W}_y(\lfloor t \rfloor \langle x = M \rangle).$

So $\lfloor K \rfloor_{\mathrm{k}}(M) \to_{\mathsf{lxr}} \lfloor K' \rfloor_{\mathrm{k}}(M)$ by the rule $(Weak_2)$ in $\lambda\mathsf{lxr}$.

(ω_5) $(x \odot t) :: k \to x \odot (t :: k).$

$\lfloor K \rfloor_{\mathrm{k}}(M) = \lfloor (x \odot t) :: k \rfloor_{\mathrm{k}}(M) = \lfloor k \rfloor_{\mathrm{k}}(M \lfloor x \odot t \rfloor) = \lfloor k \rfloor_{\mathrm{k}}(M\mathcal{W}_x(\lfloor t \rfloor)).$

$\lfloor K' \rfloor_{\mathrm{k}}(M) = \lfloor x \odot (t :: k) \rfloor_{\mathrm{k}}(M) = \mathcal{W}_x(\lfloor t :: k \rfloor_{\mathrm{k}}(M)) = \mathcal{W}_x(\lfloor k \rfloor_{\mathrm{k}}(M \lfloor t \rfloor)).$

Applying the rule $(WApp_2)$ of the $\lambda\mathsf{lxr}$-calculus followed by Lemma 5.59 we get that

$\lfloor K \rfloor_{\mathrm{k}}(M) \twoheadrightarrow_{\mathsf{lxr}} \lfloor K' \rfloor_{\mathrm{k}}(M)$ in $\lambda\mathsf{lxr}$.

(ω_6) $t :: (x \odot k) \to x \odot (t :: k)$

$\lfloor K \rfloor_{\mathrm{k}}(M) = \lfloor t :: (x \odot t) \rfloor_{\mathrm{k}}(M) = \lfloor x \odot k \rfloor_{\mathrm{k}}(M \lfloor t \rfloor) = \mathcal{W}_x(\lfloor k \rfloor_{\mathrm{k}}(M \lfloor t \rfloor)) = \lfloor K' \rfloor_{\mathrm{k}}(M).$
□

From the previous proposition, we see that there are two groups of reduction rules, with respect to their interpretation in the $\lambda\mathsf{lxr}$-calculus.

The first group consists of 15 $\lambda^{\mathsf{Gtz}}_{®}$-reductions that are interpreted as reductions in $\lambda\mathsf{lxr}$:

$$\beta,\ \mu,\ \sigma_2,\ \sigma_3,\ \sigma_4,\ \sigma_5,\ \gamma_1,\ \gamma_2,\ \gamma_5,\ \gamma\omega_1,\ \gamma\omega_2,\ \omega_1,\ \omega_2,\ \omega_4 \text{ and } \omega_5.$$

The second group consists of 8 $\lambda_{\circledR}^{\mathsf{Gtz}}$-reductions that are interpreted as equivalencies or identities in $\lambda\mathsf{lxr}$:

$$\pi,\ \mu,\ \sigma_1,\ \gamma_3,\ \gamma_4,\ \gamma_6,\ \omega_3 \text{ and } \omega_6.$$

That leads to the conclusion that it is possible to introduce an ordering relation based on the given mapping which will not increase the size of the $\lambda_{\circledR}^{\mathsf{Gtz}}$ reduct.

However, in order to find an ordering that strictly decreases the size of all $\lambda_{\circledR}^{\mathsf{Gtz}}$ reducts, we introduce the following norms on the set $\Lambda_{\circledR}^{\mathsf{Gtz}}$ of $\lambda_{\circledR}^{\mathsf{Gtz}}$-expressions. These norms are specially designed to show that the 8 $\lambda_{\circledR}^{\mathsf{Gtz}}$-reductions of the second group are terminating.

Definition 5.28 *The size of a $\lambda_{\circledR}^{\mathsf{Gtz}}$-expression is the function $\mathcal{S} : \Lambda_{\circledR}^{\mathsf{Gtz}} \to \mathbb{N}$, defined as follows:*

$$\begin{array}{rcl}
\mathcal{S}(x) & = & 1 \\
\mathcal{S}(\lambda x.t) & = & 1+\mathcal{S}(t) \\
\mathcal{S}(x\odot t) & = & 1+\mathcal{S}(t) \\
\mathcal{S}(x <^{y}_{z} t) & = & 1+\mathcal{S}(t) \\
\mathcal{S}(tk) & = & \mathcal{S}(t)+\mathcal{S}(k) \\
\mathcal{S}(\widehat{x}.t) & = & 1+\mathcal{S}(t) \\
\mathcal{S}(t :: k) & = & \mathcal{S}(t)+\mathcal{S}(k) \\
\mathcal{S}(x\odot k) & = & 1+\mathcal{S}(k) \\
\mathcal{S}(x <^{y}_{z} k) & = & 1+\mathcal{S}(k)
\end{array}$$

The main purpose of the following measure is to decrease during reduction steps that perform a contraction propagation.

Definition 5.29 *The function $\|\ \|_C : \Lambda_{\circledR}^{\mathsf{Gtz}} \to \mathbb{N}_0$, is defined as follows:*

$$\begin{array}{rcl}
\|x\|_C & = & 0 \\
\|\lambda x.t\|_C & = & \|t\|_C \\
\|x\odot t\|_C & = & \|t\|_C \\
\|x <^{y}_{z} t\|_C & = & \|t\|_C+\mathcal{S}(t) \\
\|tk\|_C & = & \|t\|_C+\|k\|_C \\
\|\widehat{x}.t\|_C & = & \|t\|_C \\
\|t :: k\|_C & = & \|t\|_C+\|k\|_C \\
\|x\odot k\|_C & = & \|k\|_C \\
\|x <^{y}_{z} k\|_C & = & \|k\|_C+\mathcal{S}(k)
\end{array}$$

The next norm is designed with the purpose of decreasing during the computational steps that perform an extraction of weakening.

Definition 5.30 *The function* $\| \,\|_W : \Lambda^{\mathsf{Gtz}}_{\circledR} \to \mathbb{N}_0$, *is defined as follows:*

$$\begin{array}{rcl}
\|x\|_W & = & 1 \\
\|\lambda x.t\|_W & = & 1+\|t\|_W \\
\|x \odot t\|_W & = & 0 \\
\|x <^y_z t\|_W & = & 1+\|t\|_W \\
\|tk\|_W & = & 1+\|t\|_W+\|k\|_W \\
\|\widehat{x}.t\|_W & = & 1+\|t\|_W \\
\|t :: k\|_W & = & 1+\|t\|_W+\|k\|_W \\
\|x \odot k\|_W & = & 0 \\
\|x <^y_z k\|_W & = & 1+\|k\|_W
\end{array}$$

Finally, a measure called “P-norm” is specially designed to witness the reduced size of a term participating in the application in the π-reduction. This norm needs an auxiliary norm, “A-norm”, proposed in the following definition.

Definition 5.31 *The function* $\| \,\|_A : \Lambda^{\mathsf{Gtz}}_{\circledR} \to \mathbb{N}_0$, *is defined as follows:*

$$\begin{array}{rcl}
\|x\|_A & = & 0 \\
\|\lambda x.t\|_A & = & \|t\|_A \\
\|x \odot t\|_A & = & \|t\|_A \\
\|x <^y_z t\|_A & = & \|t\|_A \\
\|tk\|_A & = & 1+\|t\|_A+\|k\|_A \\
\|\widehat{x}.t\|_A & = & \|t\|_A \\
\|t :: k\|_A & = & \|t\|_A+\|k\|_A \\
\|x \odot k\|_A & = & \|k\|_A \\
\|x <^y_z k\|_A & = & \|k\|_A
\end{array}$$

Definition 5.32 *The function* $\| \,\|_P : \Lambda^{\mathsf{Gtz}}_{\circledR} \to \mathbb{N}$, *is defined as follows:*

$$\begin{array}{rcl}
\|x\|_P & = & 1 \\
\|\lambda x.t\|_P & = & \|t\|_P \\
\|x \odot t\|_P & = & \|t\|_P \\
\|x <^y_z t\|_P & = & \|t\|_P \\
\|tk\|_P & = & \|t\|_P+\|k\|_P+\|t\|_A \\
\|\widehat{x}.t\|_P & = & \|t\|_P \\
\|t :: k\|_P & = & \|t\|_P+\|k\|_P \\
\|x \odot k\|_P & = & \|k\|_P \\
\|x <^y_z k\|_P & = & \|k\|_P
\end{array}$$

Lemma 5.34 examines the behaviour of the @ operator with respect to some of the introduced norms [3]. The item (iii) uses a notion of a *kernel* of a context, set up by the following definition.

Definition 5.33 *A term* $\ker(k)$*, representing the kernel of a context k, is inductively defined on the structure of k.*

$$\begin{array}{rcl} \ker(\widehat{x}.t) & = & t \\ \ker(t :: k) & = & \ker(k) \\ \ker(x \odot k) & = & \ker(k) \\ \ker(x <^{y}_{z} k) & = & \ker(k) \end{array}$$

Lemma 5.34

(i) $\mathcal{S}(k@k') = \mathcal{S}(k)+\mathcal{S}(k')$*;*

(ii) $\|k@k'\|_A = 1+\|k\|_A+\|k'\|_A$*;*

(iii) $\|k@k'\|_P = \|k\|_P+\|k'\|_P+\|\ker(k)\|_A$*.*

Proof: By induction on the structure of k, with the base case $k \equiv \widehat{x}.t$.

(i) Let us prove $\mathcal{S}(k@k') = \mathcal{S}(k)+\mathcal{S}(k')$.

- For the base case $k \equiv \widehat{x}.t$, $\mathcal{S}(k) = 1+\mathcal{S}(t)$, so the following holds

$$\mathcal{S}((\widehat{x}.t)@k') \triangleq \mathcal{S}(\widehat{x}.tk') = 1+\mathcal{S}(t)+\mathcal{S}(k') = \mathcal{S}(k)+\mathcal{S}(k').$$

- For $k \equiv t :: k''$, $\mathcal{S}(k) = \mathcal{S}(t)+\mathcal{S}(k'')$, and we have that

$$\mathcal{S}((t :: k'')@k') = \mathcal{S}(t)+\mathcal{S}(k''@k') =_{\text{IH}} \mathcal{S}(t)+\mathcal{S}(k'')+\mathcal{S}(k') = \mathcal{S}(k)+\mathcal{S}(k').$$

- For $k \equiv x \odot k''$, $\mathcal{S}(k) = 1+\mathcal{S}(k'')$, and we have that

$$\mathcal{S}((x \odot k'')@k') = 1+\mathcal{S}(k''@k') =_{\text{IH}} 1+\mathcal{S}(k'')+\mathcal{S}(k') = \mathcal{S}(k)+\mathcal{S}(k').$$

- The remaining case $k \equiv x <^{y}_{z} k''$ is analogous.

(ii) Let us prove $\|k@k'\|_A = 1+\|k\|_A+\|k'\|_A$.

- For the base case $k \equiv \widehat{x}.t$, $\|k\|_A = \|t\|_A$, so the following holds

$$\|(\widehat{x}.t)@k'\|_A \triangleq \|\widehat{x}.tk'\|_A = 1+\|t\|_A+\|k'\|_A = 1+\|k\|_A+\|k'\|_A.$$

[3]One may notice that the lemma does not treat the cases of w-norm and c-norm. However, these cases are not necessary for the proof of termination.

– For the case $k \equiv t :: k''$, $\|k\|_A = \|t\|_A + \|k''\|_A$, and we have that

$$\|(t :: k'')@k'\|_A = \|t\|_A + \|k''@k'\|_A =_{\mathrm{IH}} \|t\|_A + 1 + \|k''\|_A + \|k'\|_A = 1 + \|k\|_A + \|k'\|_A.$$

– For the case $k \equiv x \odot k''$, $\|k\|_A = \|k''\|_A$, and we have that

$$\|(x \odot k'')@k'\|_A = \|k''@k'\|_A =_{\mathrm{IH}} 1 + \|k''\|_A + \|k'\|_A = 1 + \|k\|_A + \|k'\|_A.$$

– The remaining case $k \equiv x <^y_z k''$ is analogous.

(iii) Let us prove $\|k@k'\|_P = \|k\|_P + \|k'\|_P + \|\mathrm{ker}(k)\|_A$.

– For the base case $k \equiv \widehat{x}.t$, $\|k\|_P = \|t\|_P$ and $\mathrm{ker}(k) = t$, thus the following holds

$$\|(\widehat{x}.t)@k'\|_P \triangleq \|\widehat{x}.tk'\|_P = \|t\|_P + \|k'\|_P + \|t\|_A = \|k\|_P + \|k'\|_P + \|\mathrm{ker}(k)\|_A.$$

– For the case $k \equiv t :: k''$, $\|k\|_P = \|t\|_P + \|k''\|_P$ and $\mathrm{ker}(k) = \mathrm{ker}(k'')$, so we have that

$$\begin{gathered}\|(t :: k'')@k'\|_P = \|t\|_P + \|k''@k'\|_P =_{\mathrm{IH}} \\ \|t\|_P + \|k''\|_P + \|k'\|_P + \|\mathrm{ker}(k'')\|_A = \\ \|k\|_P + \|k'\|_P + \|\mathrm{ker}(k)\|_A.\end{gathered}$$

– For the case $k \equiv x \odot k''$, $\|k\|_P = \|k''\|_P$ and $\mathrm{ker}(k) = \mathrm{ker}(k'')$, so we have that

$$\begin{gathered}\|(x \odot k'')@k'\|_P = \|k''@k'\|_P =_{\mathrm{IH}} \|k''\|_P + \|k'\|_P + \|\mathrm{ker}(k'')\|_A = \\ \|k\|_P + \|k'\|_P + \|\mathrm{ker}(k)\|_A.\end{gathered}$$

– The case $k \equiv x <^y_z k''$ is analogous, hence the proof is done. □

The following four lemmas treat each of the introduced norms, and show how they change during the computational steps, with respect only to the eight reductions that are simulated by equalities in the λlxr-calculus. Since the reductions in the $\lambda_{®}^{\mathsf{Gtz}}$-calculus are closed under any contexts, we will argue only the case of the outermost reductions, without losing generality.

Lemma 5.35 *For all $\lambda_{®}^{\mathsf{Gtz}}$ expressions:*

(i) For $r \in \{\mu, \sigma_1\}$: if $e \to_r e'$, then $\mathcal{S}(e) > \mathcal{S}(e')$.

(ii) For $r \in \{\pi, \gamma_3, \gamma_4, \gamma_6, \omega_3, \omega_6\}$: if $e \to_r e'$, then $\mathcal{S}(e) = \mathcal{S}(e')$.

Proof:

(i)
- In the case of μ-reduction, it is sufficient to show that $\mathcal{S}(\widehat{x}.xk) - \mathcal{S}(k) > 0$.
  Indeed, $\mathcal{S}(\widehat{x}.xk) - \mathcal{S}(k) = (1+\mathcal{S}(x)+\mathcal{S}(k)) - \mathcal{S}(k) = 2$.
-] In the case of σ_1-reduction, it is sufficient to show that $\mathcal{S}(T(\widehat{x}.x)) - \mathcal{S}(T) > 0$.
  Indeed, $\mathcal{S}(T(\widehat{x}.x)) - \mathcal{S}(T) = \mathcal{S}(T) + (1+\mathcal{S}(x)) - \mathcal{S}(T) = 2$.

(ii)
- If $e \to_\pi e'$, it is sufficient to show that $\mathcal{S}((tk)k') = \mathcal{S}(t(k@k'))$, which is true, since both sizes are equal to $\mathcal{S}(t)+\mathcal{S}(k)+\mathcal{S}(k')$.
- If $e \to_{\gamma_3} e'$, it is sufficient to show that $\mathcal{S}(x <^{x_1}_{x_2} (tk)) = \mathcal{S}(t(x <^{x_1}_{x_2} k))$, which is true, since both sizes are equal to $1+\mathcal{S}(t)+\mathcal{S}(k)$.
- If $e \to_{\gamma_4} e'$, it is sufficient to show that $\mathcal{S}(x <^{x_1}_{x_2} (\widehat{y}.t)) = \mathcal{S}(\widehat{y}.(x <^{x_1}_{x_2} t))$which is true, since both sizes are equal to $2+\mathcal{S}(t)$.
- If $e \to_{\gamma_6} e'$, it is sufficient to show that $\mathcal{S}(x <^{x_1}_{x_2} (t :: k)) = \mathcal{S}(t :: (x <^{x_1}_{x_2} k))$, which is true, since both sizes are equal to $1+\mathcal{S}(t)+\mathcal{S}(k)$.
- If $e \to_{\omega_3} e'$, it is sufficient to show that $\mathcal{S}(t(x \odot k)) = \mathcal{S}(x \odot (tk))$, which is true, since both sizes are equal to $1+\mathcal{S}(t)+\mathcal{S}(k)$.
- If $e \to_{\omega_6} e'$, it is sufficient to show that $\mathcal{S}(t :: (x \odot k)) = \mathcal{S}(x \odot (t :: k))$, which is true, since both sizes are equal to $1+\mathcal{S}(t)+\mathcal{S}(k)$. □

Lemma 5.36 *For all $\lambda_{®}^{\mathsf{Gtz}}$ expressions:*

(i) If $e \to_\pi e'$, then $\|e\|_P > \|e'\|_P$.

(ii) For $r \in \{\gamma_3, \gamma_4, \gamma_6, \omega_3, \omega_6\}$: if $e \to_r e'$, then $\|e\|_P = \|e'\|_P$.

Proof:

(i) If $e \to_\pi e'$, it is sufficient to show that $\|(tk)k'\|_P - \|t(k@k')\|_P > 0$.
From Lemma 5.34, we have that

$$\|t(k@k')\|_P = \|t\|_P + \|k\|_P + \|k'\|_P + \|t\|_A + \|\mathrm{ker}(k)\|_A.$$

Then,

$$\begin{gathered}\|(tk)k'\|_P - \|t(k@k')\|_P = \\ \|t\|_P + \|k\|_P + \|k'\|_P + \|tk\|_A + \|t\|_A - (\|t\|_P + \|k\|_P + \|k'\|_P + \|t\|_A + \\ \|\mathrm{ker}(k)\|_A) = \\ \|tk\|_A - \|\mathrm{ker}(k)\|_A = 1 + \|t\|_A + \|k\|_A - \|\mathrm{ker}(k)\|_A \geq^{(*)} 1 + \|t\|_A > 0.\end{gathered}$$

The inequality (*) holds because $\|k\|_A \geq \|\mathrm{ker}(k)\|_A$, which can be very easily proved by inspecting the cases according to the structure of the context k.

(ii)
- If $e \rightarrow_{\gamma_3} e'$, it is sufficient to show that $\|x <^{x_1}_{x_2} (tk)\|_P = \|t(x <^{x_1}_{x_2} k)\|_P$, which is true, since both sizes are equal to $\|t\|_P + \|k\|_P + \|t\|_A$.
- If $e \rightarrow_{\gamma_4} e'$, it is sufficient to show that $\|x <^{x_1}_{x_2} (\widehat{y}.t)\|_P = \|\widehat{y}.(x <^{x_1}_{x_2} t)\|_P$ which is true, since both sizes are equal to $\|t\|_P$.
- If $e \rightarrow_{\gamma_6} e'$, it is sufficient to show that $\|x <^{x_1}_{x_2} (t :: k)\|_P = \|t :: (x <^{x_1}_{x_2} k)\|_P$, which is true, since both sizes are equal to $\|t\|_P + \|k\|_P$.
- If $e \rightarrow_{\omega_3} e'$, it is sufficient to show that $\|t(x \odot k)\|_P = \|x \odot (tk)\|_P$, which is true, since both sizes are equal to $\|t\|_P + \|k\|_P + \|t\|_A$.
- If $e \rightarrow_{\omega_6} e'$, it is sufficient to show that $\|t :: (x \odot k)\|_P = \|x \odot (t :: k)\|_P$, which is true, since both sizes are equal to $\|t\|_P + \|k\|_P$. $\square$

Lemma 5.37 *For all $\lambda_{®}^{\mathsf{Gtz}}$ expressions:*

(i) For $r \in \{\omega_3, \omega_6\}$: if $e \rightarrow_r e'$, then $\|e\|_W > \|e'\|_W$.

(ii) For $r \in \{\gamma_3, \gamma_4, \gamma_6\}$: if $e \rightarrow_r e'$, then $\|e\|_W = \|e'\|_W$.

Proof:

(i)
- In the case of ω_3-reduction, it is sufficient to show that
  $\|t(x \odot k)\|_W - \|x \odot (tk)\|_W > 0$.
  $\|t(x \odot k)\|_W - \|x \odot (tk)\|_W = (1 + \|t\|_W + 0) - 0 > 0$.
- In the case of ω_6-reduction, it is sufficient to show that
  $\|t :: (x \odot k)\|_W - \|x \odot (t :: k)\|_W > 0$.
  $\|t :: (x \odot k)\|_W - \|x \odot (t :: k)\|_W = (1 + \|t\|_W + 0) - 0 > 0$.

(ii)
- If $e \rightarrow_{\gamma_3} e'$, it is sufficient to show that $\|x <^{x_1}_{x_2} (tk)\|_W = \|t(x <^{x_1}_{x_2} k)\|_W$, which is true, since both sizes are equal to $2 + \|t\|_W + \|k\|_W$.
- If $e \rightarrow_{\gamma_4} e'$, it is sufficient to show that $\|x <^{x_1}_{x_2} (\widehat{y}.t)\|_W = \|\widehat{y}.(x <^{x_1}_{x_2} t)\|_W$ which is true, since both sizes are equal to $2 + \|t\|_W$.
- If $e \rightarrow_{\gamma_6} e'$, it is sufficient to show that $\|x <^{x_1}_{x_2} (t :: k)\|_W = \|t :: (x <^{x_1}_{x_2} k)\|_W$, which is true, since both sizes are equal to $2 + \|t\|_W + \|k\|_W$. $\square$

Lemma 5.38 *For all $\lambda_{®}^{\mathsf{Gtz}}$ expressions, if $r \in \{\gamma_3, \gamma_4, \gamma_6\}$ and $e \rightarrow_r e'$, then $\|e\|_C > \|e'\|_C$.*

Proof:

- In the case of γ_3-reduction, it is sufficient to show that
$\|x <_{x_2}^{x_1} (tk)\|_C - \|t(x <_{x_2}^{x_1} k)\|_C > 0$.
$\|x <_{x_2}^{x_1} (tk)\|_C - \|t(x <_{x_2}^{x_1} k)\|_C = (\|tk\|_C + \mathcal{S}(tk)) - (\|t\|_C + \|k\|_C + \mathcal{S}(k)) =$
$(\|t\|_C + \|k\|_C + \mathcal{S}(t) + \mathcal{S}(k)) - (\|t\|_C + \|k\|_C + \mathcal{S}(k)) = \mathcal{S}(t) > 0$, by definition of $\mathcal{S}$.

- In the case of γ_4-reduction, it is sufficient to show that
$\|x <_{x_2}^{x_1} (\widehat{y}.t)\|_C - \|\widehat{y}.(x <_{x_2}^{x_1} t)\|_C > 0$.
$\|x <_{x_2}^{x_1} (\widehat{y}.t)\|_C - \|\widehat{y}.(x <_{x_2}^{x_1} t)\|_C = (\|\widehat{y}.t\|_C + \mathcal{S}(\widehat{y}.t)) - (\|t\|_C + \mathcal{S}(t)) =$
$(\|t\|_C + 1 + \mathcal{S}(t)) - (\|t\|_C + \mathcal{S}(t)) = 1$.

- In the case of γ_6-reduction, it is sufficient to show that
$\|x <_{x_2}^{x_1} (t :: k)\|_C - \|t :: (x <_{x_2}^{x_1} k)\|_C > 0$.
$\|x <_{x_2}^{x_1} (t :: k)\|_C - \|t :: (x <_{x_2}^{x_1} k)\|_C = (\|t :: k\|_C + \mathcal{S}(t :: k)) - (\|t\|_C + \|k\|_C + \mathcal{S}(k)) =$
$(\|t\|_C + \|k\|_C + \mathcal{S}(t) + \mathcal{S}(k)) - (\|t\|_C + \|k\|_C + \mathcal{S}(k)) = \mathcal{S}(t) > 0$. $\square$

Now, we can define the following orderings based on the previously introduced mapping and norms. Notice that these relations are defined only on the set of simply typed $\lambda_{®}^{\mathsf{Gtz}}$ expressions.

Definition 5.39 *We define the following strict orders and equivalencies on the set of typed $\lambda_{®}^{\mathsf{Gtz}}$ expressions:*

i) $t >_{\mathsf{lxr}} t'$ *iff* $\lfloor t \rfloor \rightarrow_{\mathsf{lxr}}^{+} \lfloor t' \rfloor$;
$t =_{\mathsf{lxr}} t'$ *iff* $\lfloor t \rfloor \equiv \lfloor t' \rfloor$;
$k >_{\mathsf{lxr}} k'$ *iff* $\lfloor k \rfloor_{\mathsf{k}}(M) \rightarrow_{\mathsf{lxr}}^{+} \lfloor k' \rfloor(M)$ *for every* $\lambda\mathsf{lxr}$*-term* M *;*
$k =_{\mathsf{lxr}} k'$ *iff* $\lfloor k \rfloor_{\mathsf{k}}(M) \equiv \lfloor k' \rfloor_{\mathsf{k}}(M)$ *for every* $\lambda\mathsf{lxr}$*-term* M*;*

ii) $e >_s e'$ *iff* $\mathcal{S}(e) > \mathcal{S}(e')$;
$e =_s e'$ *iff* $\mathcal{S}(e) = \mathcal{S}(e')$;

iii) $e >_c e'$ *iff* $\|e\|_C > \|e'\|_C$;
$e =_c e'$ *iff* $\|e\|_C = \|e'\|_C$;

iv) $e >_w e'$ *iff* $\|e\|_W > \|e'\|_W$;
$e =_w e'$ *iff* $\|e\|_W = \|e'\|_W$;

v) $e >_p e'$ *iff* $\|e\|_P > \|e'\|_P$;
$e =_p e'$ *iff* $\|e\|_P = \|e'\|_P$.

A lexicographic product of the two orders $>_1$ and $>_2$ is usually defined as follows [2]:

$$a >_1 \times_{lex} >_2 b \Leftrightarrow a >_1 b \text{ or } a =_1 b \text{ and } a >_2 b.$$

Definition 5.40 *We define the relation $\gg$ on the set of typed $\lambda_{®}^{\mathsf{Gtz}}$-expressions as a lexicographic product:*

$$\gg = >_{\mathsf{lxr}} \times_{lex} >_s \times_{lex} >_p \times_{lex} >_w \times_{lex} >_c .$$

The following proposition proves that the reduction relation on the set of typed $\lambda_{®}^{\mathsf{Gtz}}$-expressions is included in the given lexicographic product $\gg$.

Proposition 5.41 *For each typed $\lambda_{®}^{\mathsf{Gtz}}$ expression e: if $e \to e'$, then $e \gg e'$.*

Proof: The proof is by the case analysis on the kind of reduction and the structure of $\gg$.

- If $e \to e'$ by β, μ, σ_2, σ_3, σ_4, σ_5, γ_1, γ_2, γ_5, $\gamma\omega_1$, $\gamma\omega_2$, ω_1, ω_2, ω_4 or ω_5 reduction, then $e >_{\mathsf{lxr}} e'$ by Proposition 5.27.
- If $e \to e'$ by μ or σ_1, then $e =_{\mathsf{lxr}} e'$ by Proposition 5.27 and $e >_s e'$ by Lemma 5.35.
- If $e \to e'$ by π, then $e =_{\mathsf{lxr}} e'$ by Proposition 5.27, $e =_s e'$ by Lemma 5.35, and $e >_p e'$ by Lemma 5.36.
- If $e \to e'$ by ω_3 or ω_6, then $e =_{lxr} e'$ by Proposition 5.27, $e =_s e'$ by Lemma 5.35, $e =_p e'$ by Lemma 5.36 and $e >_w e'$ by Lemma 5.37.
- Finally, If $e \to e'$ by γ_3, γ_4 or γ_6, then $e =_{\mathsf{lxr}} e'$ by Proposition 5.27, $e =_s e'$ by Lemma 5.35, $e =_p e'$ by Lemma 5.36, $e =_w e'$ by Lemma 5.37 and $e >_c e'$ by Lemma 5.38. $\square$

For proving the SN property, we use the notion of well-founded relation. It is said that the relation R is *well-founded* on the class X, if and only if every non-empty subset of X has a minimal element with respect to R. Therefore, strong normalisation can be seen as well-foundedness of the reduction relation on the set of typeable $\lambda_{®}^{\mathsf{Gtz}}$-expressions.

Theorem 5.42 (Strong normalisation) *If $\lambda_{®}^{\mathsf{Gtz}}$-expression is typeable in the system $\lambda_{®}^{\mathsf{Gtz}} \to$, then it is strongly normalising.*

Proof: In order to prove the SN property, we prove that the reduction relation is well-founded on the set of typed $\lambda^{\mathsf{Gtz}}_{®}$-expressions.
The relation $>_{\mathsf{lxr}}$ is based on the interpretation into the λlxr-calculus. Since by Proposition 5.21 the typeability is preserved by the interpretation, and the simply-typed λlxr-calculus is proved to be SN in [45] (hence reduction $\to_{\mathsf{lxr}}$ is well-founded on the set of typed λlxr-terms), we conclude that $>_{\mathsf{lxr}}$ is well-founded itself.
Similarly, the relations $>_s$, $>_c$, $>_w$ and $>_p$ are well-founded, because they are all based on the interpretation into the well-founded relation $>$ on the set of natural numbers $\mathbb{N}$.
Now, the relation $\gg$ is well-founded, because it is the lexicographic product of the well-founded components.
Finally, the reduction relation on the set of typed $\lambda^{\mathsf{Gtz}}_{®}$-expressions is well-founded because it is by Proposition 5.41 included in the other well-founded relation, namely $\gg$. $\square$

Simply-typed $\lambda^{\mathsf{Gtz}}_{®}$-calculus extends Curry-Howard correspondence to the intuitionistic sequent calculus with explicit structural rules of weakening and contraction. However, as expected, simple types are too narrow class if one wants to type all strongly normalising $\lambda^{\mathsf{Gtz}}_{®}$-expressions, hence the next logical step was to introduce intersection types into the $\lambda^{\mathsf{Gtz}}_{®}$-calculus.

5.3 Intersection types for the $\lambda^{\mathsf{Gtz}}_{®}$-calculus

5.3.1 The system $\lambda^{\mathsf{Gtz}}_{®}\cap$

Our main goal in this section is to propose a type assignment system for the $\lambda^{\mathsf{Gtz}}_{®}$-calculus that would characterise the set of strongly normalising expressions by means of typeability. For that purpose, we introduce an intersection type assignment system $\lambda^{\mathsf{Gtz}}_{®}\cap$. This system assigns a restricted form of intersection types, namely *strict types*, to $\lambda^{\mathsf{Gtz}}_{®}$-expressions. Strict types were proposed by van Bakel in [70] and already used by Espírito Santo et al. in [25] for an alternative characterisation of strong normalisation in the λ^{Gtz}-calculus.

The syntax of types is defined as follows:

$$\begin{array}{lccl} \text{Strict types} & \sigma & ::= & p \mid \alpha \to \sigma \\ \text{Types} & \alpha & ::= & \cap_i^n \sigma_i \end{array}$$

where p ranges over a denumerable set of type atoms, and the notation $\cap_i^n \sigma_i$ stands for $\sigma_1 \cap \ldots \cap \sigma_n$, $n \geq 0$.

Particularly, if $n = 0$, then $\cap_i^0 \sigma_i$ represents the *neutral element* for the intersection operator, denoted by $\top$.

We use the following notation to distinguish different kinds of types:

$\alpha, \beta, \gamma ...$ denote types,
$\sigma, \tau, \rho, \upsilon ...$ denote strict types,
$\top$ denotes a particular type-constant,
Types denotes the set of all types.

We assume that the intersection operator is idempotent, commutative and associative. We also assume that intersection has the priority over the arrow operator. Hence,

$$\cap_i^n \tau_i \to \sigma \;=\; (\cap_i^n \tau_i) \to \sigma.$$

Now, we define the following basic notions.

Definition 5.43

(i) A basic type assignment *is an expression of the form $x : \alpha$, where x is a term variable and α is a type.*

(ii) A basis Γ *is a set $\{x_1 : \alpha_1, \ldots, x_n : \alpha_n\}$ of basic type assignments, where all term variables are different.*

(iii) The domain *of the basis Γ is the set of all term variables declared in Γ:*

$$Dom(\Gamma) = \{x_1, \ldots, x_n\}.$$

(iv) A basis extension $\Gamma, x : \alpha$ *denotes the set $\Gamma \cup \{x : \alpha\}$, where we assume that $x \notin Dom(\Gamma)$.*

(v) A bases intersection *is defined as:*

$$\Gamma \sqcap \Delta = \{x : \alpha \cap \beta \mid x : \alpha \in \Gamma \;\&\; x : \beta \in \Delta \;\&\; Dom(\Gamma) = Dom(\Delta)\}.$$

(vi) $\Gamma^{\top} = \{x : \top \mid x \in Dom(\Gamma)\}$.

In what follows we assume that the bases intersection has the priority over the basis extension, i.e.

$$\Gamma, \Delta_1 \sqcap \ldots \sqcap \Delta_n \;=\; \Gamma, (\Delta_1 \sqcap \ldots \sqcap \Delta_n).$$

It is easy to show the following properties of the bases intersection operator.

Lemma 5.44 *For arbitrary bases Γ and Δ such that $Dom(\Gamma) = Dom(\Delta)$*

(i) $\Gamma^{\top} \sqcap \Delta = \Delta$*;*

(ii) $(\Gamma, x : \alpha) \sqcap (\Delta, x : \alpha) = \Gamma \sqcap \Delta, x : \alpha$*.*

Proof:
(i) Let $\Gamma = \{x_1 : \alpha_1, \ldots, x_n : \alpha_n\}$ and $\Delta = \{x_1 : \beta_1, \ldots, x_n : \beta_n\}$. Then:
$\Gamma^{\top} \sqcap \Delta = \{x_1 : \top, \ldots, x_n : \top\} \sqcap \{x_1 : \beta_1, \ldots, x_n : \beta_n\} =$
$\{x_1 : \top \cap \beta_1, \ldots, x_n : \top \cap \beta_n\} = \{x_1 : \beta_1, \ldots, x_n : \beta_n\} = \Delta$.
(ii) The statement follows directly from the definition of the bases intersection and the fact that the intersection operator is idempotent. □

Therefore, $\Gamma^{\top}$ can be considered the neutral element for the bases intersection.

The type assignment system $\lambda_{\circledR}^{\mathsf{Gtz}}\cap$ is given in Figure 5.8. Due to the sequent style of the $\lambda_{\circledR}^{\mathsf{Gtz}}$-calculus, we again distinguish two sorts of type assignments:

- $\Gamma \vdash t : \sigma$ for typing a term and
- $\Gamma; \beta \vdash k : \sigma$, a type assignment with a *stoup*, for typing a context.

The $\lambda_{\circledR}^{\mathsf{Gtz}}\cap$ system is also syntax-directed, i.e. the intersection is incorporated into already existing rules of the simply-typed system. In the style of sequent calculus, the left intersection introduction is managed by the contraction rules $(Cont_t)$ and $(Cont_k)$, whereas the right intersection introduction is performed by the cut rule (Cut) and the left arrow introduction rule $(\to_L)$. In these two rules it is assumed that the bases $\Gamma_1, \ldots, \Gamma_n$ can be intersected, i.e.

$$Dom(\Gamma_1) = \ldots = Dom(\Gamma_n).$$

Notice that in the syntax of the $\lambda_{\circledR}^{\mathsf{Gtz}}$-calculus there are three kinds of variables according to the way they are introduced, namely as a placeholder, as a result of a contraction or as a result of a weakening. In the type assignment system $\lambda_{\circledR}^{\mathsf{Gtz}}\cap$, each kind of variable receives a specific type. Variables as placeholders have a strict type, variables resulting from a contraction have an intersection type, whereas variables resulting from a weakening have the $\top$ type.

Moreover, notice that intersection types occur only in four inference rules. In the rules $(Cont_t)$ and $(Cont_k)$ the intersection type is created, and that is the only situation where this happens. This is justified by the fact that it corresponds to the duplication of a variable. In other words, the control of the duplication of variables entails the control of the introduction of intersections in building the type of the term (or the context) in question. In the rules $(\to_L)$ and (Cut) the intersection appears on the left hand side of the turnstyle, more precisely in the stoup. This corresponds to the usage of the intersection type after it has been created by the

$$\frac{}{x:\sigma\vdash x:\sigma}\,(Ax)$$

$$\frac{\Gamma,x:\alpha\vdash t:\sigma}{\Gamma\vdash\lambda x.t:\alpha\to\sigma}\,(\to_R)\qquad\frac{\Gamma,x:\alpha\vdash t:\sigma}{\Gamma;\alpha\vdash\widehat{x}.t:\sigma}\,(Sel)$$

$$\frac{\Gamma_0\vdash t:\sigma_0\quad\dots\quad\Gamma_n\vdash t:\sigma_n\quad\Delta;\cap_j^m\tau_j\vdash k:\rho}{\Gamma_0^\top\sqcap\Gamma_1\sqcap\dots\sqcap\Gamma_n,\Delta;\cap_j^m(\cap_i^n\sigma_i\to\tau_j)\vdash t::k:\rho}\,(\to_L)$$

$$\frac{\Gamma_0\vdash t:\sigma_0\quad\dots\quad\Gamma_n\vdash t:\sigma_n\quad\Delta;\cap_i^n\sigma_i\vdash k:\tau}{\Gamma_0^\top\sqcap\Gamma_1\sqcap\dots\sqcap\Gamma_n,\Delta\vdash tk:\tau}\,(Cut)$$

$$\frac{\Gamma,x:\alpha,y:\beta\vdash t:\sigma}{\Gamma,z:\alpha\cap\beta\vdash z<^x_y t:\sigma}\,(Cont_t)\qquad\frac{\Gamma\vdash t:\sigma}{\Gamma,x:\top\vdash x\odot t:\sigma}\,(Weak_t)$$

$$\frac{\Gamma,x:\alpha,y:\beta;\gamma\vdash k:\sigma}{\Gamma,z:\alpha\cap\beta;\gamma\vdash z<^x_y k:\sigma}\,(Cont_k)\qquad\frac{\Gamma;\gamma\vdash k:\sigma}{\Gamma,x:\top;\gamma\vdash x\odot k:\sigma}\,(Weak_k)$$

Figure 5.8: $\lambda^{\mathsf{Gtz}}_{\circledR}\cap$: the $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus with intersection types

rules $(Cont_t)$ or $(Cont_k)$, if $n>0$, or by some of the rules $(Weak_t)$ or $(Weak_k)$, if $n=0$. In terms of the corresponding expressions, this means that a variable introduced by contraction or selection can be selected, and then used for the continuation of the expression construction.

In the rules $(\to_L)$ and (Cut), the role of the basis Γ_0 should be noticed. It is needed only when $n=0$ to ensure that t has a type, i.e. that t is strongly normalising. Then, in the conclusion of the rule, the types of the free variables of t can be forgotten, hence all the free variables of t receive the type $\top$. All free variables of a typeable term must occur in the environment in which the term is typeable(see Lemma 5.45), therefore "useless" variables occur with the type $\top$. If n is not 0, then Γ_0 can be any of the other environments $\Gamma_1,\dots,\Gamma_n$ and the type σ_0 of t will be the associated type $\sigma_1,\dots,\sigma_n$. Further, since $\Gamma^\top$ is a neutral element for $\sqcap$, $\Gamma^\top$ disappears in the conclusion of the rule [4]. For an additional explanation of this phenomenon consult the Example 6.8 from Chapter 6.

In the rules $(Weak_t)$ and $(Weak_k)$ the choice of the type for x is $\top$, since this corresponds to a variable which does not occur anywhere inside the expression.

[4]In the corresponding ND calculus the similar rule resembles the rules $(drop)$ and/or $(K-cut)$ in [51] and was used to present the two cases, $n=0$ and $n\neq 0$ in a uniform way.

The remaining rules, namely (Ax), $(\rightarrow_R)$ and (Sel) are the same as in the simply typed λ^{Gtz}-calculus. Notice however that the type of the variable in (Ax) is a strict type.

Proposition 5.45 (Domain Correspondence for $\lambda^{\mathsf{Gtz}}_{®}\cap$)

(i) *Let $\Gamma \vdash t : \sigma$ be a typing judgment. Then $x \in Dom(\Gamma)$ if and only if $x \in Fv(t)$.*

(ii) *Let $\Gamma; \alpha \vdash k : \sigma$ be a typing judgment. Then $x \in Dom(\Gamma)$ if and only if $x \in Fv(k)$.*

Proof: Both items are proved together, by case analysis on the type assignment rules.
The rules of Figure 5.8 belong to the following three categories.

1. *The rules that introduce a variable*, namely *(Ax)*, $(Cont_t)$, $(Weak_t)$, $(Cont_k)$ and $(Weak_k)$. In all these rules it is easy to notice that the variable is introduced in the environment if and only it is introduced in the expression as a free variable.

2. *The rules that remove variables*, namely $(\rightarrow_R)$, (Sel), $(Cont_t)$ and $(Cont_k)$. In all these rules it is easy to notice that the variables are removed from the environment if and only if they are removed from the expression as a free variable, i.e. if they become bound.

3. *The rules that do not introduce nor remove a variable*; namely $(\rightarrow_L)$ and (Cut), are straightforward. □

The Generation lemma induced by the proposed system is the following:

Lemma 5.46 (Generation lemma for $\lambda^{\mathsf{Gtz}}_{®}\cap$)

(i) $\Gamma \vdash \lambda x.t : \tau$ *iff there exist* α *and* σ *such that* $\tau \equiv \alpha \rightarrow \sigma$ *and* $\Gamma, x : \alpha \vdash t : \sigma$.

(ii) $\Gamma; \gamma \vdash t :: k : \rho$ *iff* $\Gamma = {\Gamma'_0}^{\top} \sqcap \Gamma'_1 \sqcap \ldots \sqcap \Gamma'_n, \Delta$, $\gamma \equiv \cap_j^m(\cap_i^n \sigma_i \rightarrow \tau_j)$, $\Delta; \cap_j^m \tau_j \vdash k : \rho$ *and* $\Gamma'_l \vdash t : \sigma_l$ *for all* $l \in \{0, \ldots, n\}$.

(iii) $\Gamma \vdash tk : \sigma$ *iff* $\Gamma = {\Gamma'_0}^{\top} \sqcap \Gamma'_1 \sqcap \ldots \sqcap \Gamma'_n, \Delta$, *and there exist* $\tau_j, j = 0, \ldots, n$ *such that for all* $j \in \{0, \ldots, n\}$ *the following holds:* $\Gamma'_j \vdash t : \tau_j$, *and* $\Delta; \cap_i^n \tau_i \vdash k : \sigma$.

(iv) $\Gamma; \alpha \vdash \widehat{x}.t : \sigma$ *iff* $\Gamma, x : \alpha \vdash t : \sigma$.

(v) $\Gamma \vdash z <^x_y t : \sigma$ *iff there exist* Γ', α, β *such that* $\Gamma = \Gamma', z : \alpha \cap \beta$ *and* $\Gamma', x : \alpha, y : \beta \vdash t : \sigma$.

(vi) $\Gamma \vdash x \odot t : \sigma$ *iff* $\Gamma = \Gamma', x : \top$ *and* $\Gamma' \vdash t : \sigma$.

(vii) $\Gamma; \gamma \vdash z <^{x}_{y} k : \sigma$ *iff there exist* Γ', α, β *such that* $\Gamma = \Gamma', z : \alpha \cap \beta$ *and* $\Gamma', x : \alpha, y : \beta; \gamma \vdash k : \sigma$.

(viii) $\Gamma; \gamma \vdash x \odot k : \sigma$ *iff* $\Gamma = \Gamma', x : \top$ *and* $\Gamma'; \gamma \vdash k : \sigma$.

Proof: The proof is straightforward since all the rules are syntax-directed, and relies on Proposition 5.45. □

The following two lemmas regulate the typings of the two meta-operators of $\lambda_{®}^{\mathsf{Gtz}}$.

Lemma 5.47 (Substitution lemma for $\lambda_{®}^{\mathsf{Gtz}}\cap$)

(i) If $\Gamma, x : \cap_i^n \tau_i \vdash t : \sigma$ *and for all* $j = 0, \ldots, n$, $\Delta_j \vdash T : \tau_j$, *then* $\Gamma, \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n \vdash t[T/x] : \sigma$.

(ii) If $\Gamma, x : \cap_i^n \tau_i; \alpha \vdash k : \sigma$ *and for all* $j = 0, \ldots, n$, $\Delta_j \vdash T : \tau_j$, *then* $\Gamma, \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n; \alpha \vdash k[T/x] : \sigma$.

Proof: By mutual induction on the structure of terms and contexts. We only show the base case and some cases related to the resource operators.

- Base case $t \equiv x$. By the axiom $x : \tau \vdash x : \tau$ where $\tau = \cap_i^1 \tau_i$, i.e. $n = 1$, hence the second assumption is $\Delta \vdash T : \tau$ which proves the case since $T \triangleq x[T/x]$.
- $k \equiv x \odot k'$. Now we assume $\Gamma, x : \cap_i^0 \tau_i; \alpha \vdash x \odot k' : \sigma$ and $\Delta_j \vdash T : \tau_j$ for all $j \in \{0, \ldots, 0\}$. In other words $\Gamma, x : \top; \alpha \vdash x \odot k' : \sigma$ and $\Delta_0 \vdash T : \tau_0$, i.e. T is typeable. By Generation lemma 5.46(*viii*) we get $\Gamma; \alpha \vdash k' : \sigma$. Since $Dom(\Delta_0) = Fv(T)$ by applying the $(Weak_k)$ rule multiple times we get $\Gamma, \Delta_0^{\top}; \alpha \vdash Fv(T) \odot k' \vdash \sigma$ which is exactly what we aimed to prove.
- $t \equiv x <^{x_1}_{x_2} u$. We assume that $\Gamma, x : \cap_i^n \tau_i \vdash x <^{x_1}_{x_2} u : \sigma$ and $\Delta_j \vdash T : \tau_j$ for all $j \in \{0, \ldots, n\}$. From $\Gamma, x : \cap_i^n \tau_i \vdash x <^{x_1}_{x_2} u : \sigma$, by Generation lemma 5.46(v) we get that $\cap_i^n \tau_i = \cap_{i=1}^m \tau_i \cap \cap_{i=m+1}^n \tau_i$ for some $m < n$ and $\Gamma, x_1 : \cap_i^m \tau_i, x_2 : \cap_{i=m+1}^n \tau_i \vdash u : \sigma$. From the other assumption $\Delta_j \vdash T : \tau_j$ for all $j \in \{0, \ldots, n\}$, by renaming the variables in Δ_j, i.e. by renaming all free variables of T, we get two different sets of sequents: $\Delta'_j \vdash T_1 : \tau_j$ and $\Delta''_j \vdash T_2 : \tau_j$. By applying the IH twice, we get $\Gamma, \Delta_0'^{\top} \sqcap \Delta'_1 \sqcap \ldots \sqcap \Delta'_m, \Delta_0''^{\top} \sqcap \Delta''_{m+1} \sqcap \ldots \sqcap \Delta''_n \vdash (u[T_1/x_1])[T_2/x_2] : \sigma$. Now, we apply the definition of the parallel substitution, and perform contraction on all pairs of the corresponding, i.e. obtained

by the renaming of the same variable, elements of Δ'_l and Δ''_k by introducing again the original names of the free variables of T from Δ_j and finally get what we need:

$$\Gamma, \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n \vdash Fv(T) <^{Fv(T_1)}_{Fv(T_2)} u[T_1/x_1, T_2/x_2] : \sigma. \square$$

Proposition 5.48 (Append lemma) *If* $\Gamma_j; \alpha \vdash k : \tau_j$ *for all* $j = 0, \ldots, n$, *and* $\Delta; \cap_i^n \tau_i \vdash k' : \sigma$, *then* $\Gamma_0^{\top} \sqcap \Gamma_1 \sqcap \ldots \sqcap \Gamma_n, \Delta; \alpha \vdash k@k' : \sigma$.

Proof: By induction on the structure of the context k.

- Base case $k \equiv \widehat{x}.t$. From $\Gamma_j; \alpha \vdash \widehat{x}.t : \tau_j$ for all $j = 0, \ldots, n$, by Generation lemma 5.46(iv) we get $\Gamma_j, x : \alpha \vdash t : \tau_j$ for all $j = 0, \ldots, n$. Now

$$\dfrac{\dfrac{\Gamma_0, x : \alpha \vdash t : \tau_0 \;\ldots\; \Gamma_n, x : \alpha \vdash t : \tau_n \quad \Delta; \cap_i^n \tau_i \vdash k' : \sigma}{\Gamma_0^{\top} \sqcap \Gamma_1 \sqcap \ldots \sqcap \Gamma_n, \Delta, x : \alpha \vdash tk' : \sigma}\,(Cut)}{\Gamma_0^{\top} \sqcap \Gamma_1 \sqcap \ldots \sqcap \Gamma_n, \Delta; \alpha \vdash \widehat{x}.tk' : \sigma}\,(Sel)$$

  which is exactly what we want since $k@k' = (\widehat{x}.t)@k' = \widehat{x}.tk'$.

- Case $k \equiv x \odot k''$. From $\Gamma_j; \alpha \vdash x \odot k'' : \tau_j$ for all $j = 0, \ldots, n$, by Generation lemma 5.46($viii$) we get that $\Gamma_j = \Gamma'_j, x : \top$ and $\Gamma'_j; \alpha \vdash k'' : \tau_j$, for all $j = 0, \ldots, n$. Applying the IH to k'' yields

$$\dfrac{\Gamma_0'^{\top} \sqcap \Gamma'_1 \sqcap \ldots \sqcap \Gamma'_n, \Delta; \alpha \vdash k''@k' : \sigma}{\Gamma_0^{\top} \sqcap \Gamma_1 \sqcap \ldots \sqcap \Gamma_n, \Delta; \alpha \vdash x \odot (k''@k') : \sigma}\,(Weak_k)$$

  which is exactly what we want since $k@k' = (x \odot k'')@k' = x \odot (k''@k')$, and since $\Gamma_0'^{\top} \sqcap \Gamma'_1 \sqcap \ldots \sqcap \Gamma'_n, x : \top = \Gamma_1 \sqcap \ldots \sqcap \Gamma_n$.

- Cases $k \equiv t :: k''$ and $k \equiv x <^{x_1}_{x_2} k''$ are similar. $\square$

Now we can prove that the type of a $\lambda^{\mathsf{Gtz}}_{®}$-expression is preserved along reduction and equivalence.

Proposition 5.49 (Subject equivalence for $\lambda^{\mathsf{Gtz}}_{®}\cap$)

(i) For every $\lambda_{®}^{\mathsf{Gtz}}$*-term t: if* $\Gamma \vdash t : \sigma$ *and* $t \equiv t'$*, then* $\Gamma \vdash t' : \sigma$.

(ii) For every $\lambda_{®}^{\mathsf{Gtz}}$*-context k: if* $\Gamma; \alpha \vdash k : \sigma$ *and* $k \equiv k'$*, then* $\Gamma; \alpha \vdash k' : \sigma$.

Proof: By case analysis on the applied equivalence. One can see that the typing derivations for equivalent expressions differ only in the order of applied type assignment rules for the resource operators, and all these rules do not change the type of an expression. Moreover, since the equivalencies preserve free variables (Proposition 5.12) and the set of free variables corresponds to the domain of the basis (Proposition 5.45), the left hand sides of the sequents are also equal. □

Proposition 5.50 (Subject reduction for $\lambda_{®}^{\mathsf{Gtz}}\cap$**)**

(i) For every $\lambda_{®}^{\mathsf{Gtz}}$*-term t: if* $\Gamma \vdash t : \sigma$ *and* $t \to t'$*, then* $\Gamma \vdash t' : \sigma$.

(ii) For every $\lambda_{®}^{\mathsf{Gtz}}$*-context k: if* $\Gamma; \alpha \vdash k : \sigma$ *and* $k \to k'$*, then* $\Gamma; \alpha \vdash k' : \sigma$.

Proof: By case analysis on the applied reduction, using Lemmas 5.47 and 5.48 for (σ) and (π) rule, respectively. □

5.3.2 Typeability $\Rightarrow$ SN in $\lambda_{®}^{\mathsf{Gtz}}\cap$

In this subsection, we prove the strong normalisation property of the $\lambda_{®}^{\mathsf{Gtz}}$-calculus with intersection types. The structure of the proof is analogous to the one already presented in Subsection 5.2.2 for proving the same property of the simply typed $\lambda_{®}^{\mathsf{Gtz}}$-calculus. The key difference is that in this case, the target calculus of the mapping is a variant of the $\lambda_{®}$-calculus, presented in Subsection 3.4, a formal calculus obtained by extending the standard λ-calculus with explicit operators for erasure and duplication of terms. Thus, the $\lambda_{®}$-calculus can be seen as natural deduction counterpart of $\lambda_{®}^{\mathsf{Gtz}}$.

In order to prove that if a $\lambda_{®}^{\mathsf{Gtz}}$-expression is typeable in the system $\lambda_{®}^{\mathsf{Gtz}}\cap$, then it is strongly normalising, we will show that the reduction on the set of the typeable $\lambda_{®}^{\mathsf{Gtz}}$-expressions is included in a particular well-founded relation. This relation is defined as the lexicographic product of three well-founded component relations. The first one is based on the mapping of $\lambda_{®}^{\mathsf{Gtz}}$-expressions into $\lambda_{®}$-terms. We show that this mapping preserves types and that all $\lambda_{®}^{\mathsf{Gtz}}$-reductions can be simulated by the reductions or by an equality and each $\lambda_{®}^{\mathsf{Gtz}}$-equivalence can be simulated by an $\lambda_{®}$-equivalence. The other two well-founded orders are based on the introduction of quantities designed to decrease a global measure associated to specific $\lambda_{®}^{\mathsf{Gtz}}$-expressions during the computation.

First, as in Subsection 4.3.1, we enrich the operational semantics of the $\lambda_{®}$-calculus, given in Figure 3.11, with the permutation reduction rule $\pi = \pi_1 \cup \pi_2$, where π_1 and π_2 are defined as follows:

$$\begin{array}{lrcl} (\pi_1) & (\lambda x.M)NP & \to & (\lambda x.MP)N \\ (\pi_2) & M((\lambda x.P)N) & \to & (\lambda x.MP)N. \end{array}$$

Due to the explicit control of the duplication and the erasure of variables in the $\lambda_{®}$-calculus, we know that there is exactly one occurrence of each bound variable in the $\lambda_{®}$-term. More precisely, in $(\lambda x.M)NP$, we know that $x \in Fv(M)$, therefore $x \notin Fv(P)$. Having that, if we apply the (β)-reduction to both π_1-redex $(\lambda x.M)NP$ and π_1-contractum $(\lambda x.MP)N$, we obtain the same term:

$$\begin{array}{rl} (\lambda x.M)NP \to_\beta & M[N/x]P \\ (\lambda x.MP)N \to_\beta & (MP)[N/x] \triangleq M[N/x]P. \end{array}$$

Similarly, in $M((\lambda x.P)N)$ we know that $x \in Fv(P)$, therefore $x \notin Fv(M)$. Having that, if we apply (β)-reduction to both π_2-redex $M((\lambda x.P)N)$ and π_2-contractum $(\lambda x.MP)N$, we obtain the same term:

$$\begin{array}{rl} M((\lambda x.P)N) \to_\beta & MP[N/x] \\ (\lambda x.MP)N \to_\beta & (MP)[N/x] \triangleq MP[N/x]. \end{array}$$

To conclude, (π) reductions in the $\lambda_{®}$-calculus produce β-equivalent terms and they cannot create a new β-redex.

Now, we can prove that adding (π) reductions do not change the set of the strongly normalising $\lambda_{®}$-terms. In the following proposition, we use $\mathcal{R}$ to denote the union of all resource control $\lambda_{®}$ reductions and equivalencies: $\mathcal{R} = \gamma \cup \omega \cup \gamma\omega \cup \varepsilon$.

Proposition 5.51 *If $\lambda_{®}$-term M is $\beta \cup \mathcal{R}$-SN, then it is also $\beta \cup \mathcal{R} \cup \pi$-SN.*

Proof: Suppose that there exists a $\lambda_{®}$-term M that is $\beta \cup \mathcal{R}$-SN, but is not $\beta \cup \mathcal{R} \cup \pi$-SN. $\mathcal{R} \cup \pi$-reduction is terminating in the $\lambda_{®}$-calculus, the proof combines the termination of $\mathcal{R}$-reductions, proved in [45], and the termination of π-reductions, proved in [26]. This means that there is an infinite number of (β)-reductions starting from M. As seen from the previous discussion, (π)-reductions cannot produce new β-redexes, meaning that there was an infinite number of (β)-reductions starting from M in the system without (π) reductions, i.e. in the system $\beta \cup \mathcal{R}$. But this is in contradiction with the assumption that M is $\beta \cup \mathcal{R}$-SN. Thus, all $\beta \cup \mathcal{R}$-SN

$\lambda_{®}$-terms are also $\beta \cup \mathcal{R} \cup \pi$-SN. $\Box$

Next, we define a mapping that translates $\lambda_{®}^{\mathsf{Gtz}}$-terms into $\lambda_{®}$-terms, and $\lambda_{®}^{\mathsf{Gtz}}$-contexts into a function that takes an arbitrary $\lambda_{®}$-term and creates a β-redex involving that term.

Definition 5.52 *The mapping* $\lfloor\ \rfloor : \mathrm{T}_{®}^{\mathsf{Gtz}} \to \Lambda_{®}$ *is defined together with the auxiliary mapping* $\lfloor\ \rfloor_{\mathrm{k}} : \mathrm{C}_{®}^{\mathsf{Gtz}} \to (\Lambda_{®} \to \Lambda_{®})$ *in the following way:*

$$\begin{array}{lcllcl}
\lfloor x \rfloor & = & x & \lfloor \widehat{x}.t \rfloor_{\mathrm{k}}(M) & = & (\lambda x.\lfloor t \rfloor)M \\
\lfloor \lambda x.t \rfloor & = & \lambda x.\lfloor t \rfloor & \lfloor t :: k \rfloor_{\mathrm{k}}(M) & = & \lfloor k \rfloor_{\mathrm{k}}(M\lfloor t \rfloor) \\
\lfloor x \odot t \rfloor & = & x \odot \lfloor t \rfloor & \lfloor x \odot k \rfloor_{\mathrm{k}}(M) & = & x \odot \lfloor k \rfloor_{\mathrm{k}}(M) \\
\lfloor x <_{z}^{y} t \rfloor & = & x <_{z}^{y} \lfloor t \rfloor & \lfloor x <_{z}^{y} k \rfloor_{\mathrm{k}}(M) & = & x <_{z}^{y} \lfloor k \rfloor_{\mathrm{k}}(M) \\
\lfloor tk \rfloor & = & \lfloor k \rfloor_{\mathrm{k}}(\lfloor t \rfloor) & & &
\end{array}$$

In the previous definition, M is an arbitrary $\lambda_{®}$-term such that $\lfloor k \rfloor_{\mathrm{k}}(M)$ is the $\lambda_{®}$-term. Therefore, the condition $Fv(k) \cap Fv(M) = \emptyset$ must hold.

Next, we prove some basic features of the introduced mapping, namely preservation of the free variables and interpretation of substitution. We use the same notation for the free variables in both calculi, but the subject is always clear from the context. Moreover, the free variable definition for the terms of the two calculi coincide. The same applies to the notation of the implicit substitution.

Lemma 5.53

(i) $Fv(\lfloor t \rfloor) = Fv(t)$, *for* $t \in \mathrm{T}_{®}^{\mathsf{Gtz}}$.

(ii) $Fv(\lfloor k \rfloor_{\mathrm{k}}(M)) = Fv(k) \cup Fv(M)$, *for* $k \in \mathrm{C}_{®}^{\mathsf{Gtz}}$ *and* $M \in \Lambda_{®}$.

Proof: By mutual induction on the structure of $\lambda_{®}^{\mathsf{Gtz}}$-terms and $\lambda_{®}^{\mathsf{Gtz}}$-contexts.

- Basic case $t \equiv x$ is trivial, since $\lfloor x \rfloor = x$.
- Cases $t \equiv \lambda x.t$, $t \equiv x \odot t$ and $t \equiv x <_{z}^{y} t$ are easy, because the corresponding interpretations are the $\lambda_{®}$-terms of the same structure.
- Case $k \equiv \widehat{x}.t$:

$$\begin{aligned} Fv(\lfloor k \rfloor_{\mathrm{k}}(M)) &= Fv(\lfloor \widehat{x}.t \rfloor_{\mathrm{k}}(M)) = Fv((\lambda x.\lfloor t \rfloor)M) = Fv(\lfloor t \rfloor) \setminus \{x\} \cup Fv(M) \\ &=_{IH} Fv(t) \setminus \{x\} \cup Fv(M) = Fv(\widehat{x}.t) \cup Fv(M) = Fv(k) \cup Fv(M). \end{aligned}$$

- Case $k \equiv t :: k'$:

$$\begin{gathered}Fv(\lfloor k\rfloor_{\mathrm{k}}(M)) = Fv(\lfloor t :: k'\rfloor_{\mathrm{k}}(M)) = Fv(\lfloor k'\rfloor_{\mathrm{k}}(M\lfloor t\rfloor)) =_{IH} \\ Fv(k') \cup Fv(M\lfloor t\rfloor)) \\ =_{IH} Fv(k') \cup Fv(M) \cup Fv(t) = Fv(t :: k') \cup Fv(M) = Fv(k) \cup Fv(M).\end{gathered}$$

- Case $k \equiv x \odot k'$:

$$\begin{gathered}Fv(\lfloor k\rfloor_{\mathrm{k}}(M)) = Fv(\lfloor x \odot k'\rfloor_{\mathrm{k}}(M)) = Fv(x \odot \lfloor k'\rfloor_{\mathrm{k}}(M)) = \\ \{x\} \cup Fv(\lfloor k'\rfloor_{\mathrm{k}}(M)) \\ =_{IH} \{x\} \cup Fv(k') \cup Fv(M) = Fv(x \odot k') \cup Fv(M) = Fv(k) \cup Fv(M).\end{gathered}$$

- Case $t \equiv uk$:

$$\begin{gathered}Fv(\lfloor t\rfloor) = Fv(\lfloor uk\rfloor) = Fv(\lfloor k\rfloor_{\mathrm{k}}(\lfloor u\rfloor) =_{IH} Fv(k) \cup Fv(\lfloor u\rfloor) \\ =_{IH} Fv(k) \cup Fv(u) = Fv(t).\end{gathered}$$

- The remaining case $k \equiv x <_z^y k'$ is similar. □

Lemma 5.54

(i) $\lfloor v[t/x]\rfloor = \lfloor v\rfloor[\lfloor t\rfloor/x]$, *for* $v,t \in \mathrm{T}_{\circledR}^{\mathsf{Gtz}}$.

(ii) $\lfloor k[t/x]\rfloor_{\mathrm{k}}(M) = \lfloor k\rfloor_{\mathrm{k}}[\lfloor t\rfloor/x](M)$, *for* $t \in \mathrm{T}_{\circledR}^{\mathsf{Gtz}}$, $k \in \mathrm{C}_{\circledR}^{\mathsf{Gtz}}$ *and* $M \in \Lambda_{\circledR}$.

Proof: By mutual induction on the structure of $\lambda_{\circledR}^{\mathsf{Gtz}}$ terms and contexts. Notice that in the case of the substitution on contexts, we can write $\lfloor k\rfloor_{\mathrm{k}}[\lfloor t\rfloor/x](M)$ instead of $(\lfloor k\rfloor_{\mathrm{k}}(M))[\lfloor t\rfloor/x]$ because we know that k and M cannot share free variables, therefore $x \notin Fv(M)$. □

Now, we prove that the mappings $\lfloor\ \rfloor$ and $\lfloor\ \rfloor_{\mathrm{k}}$ preserve types. In the sequel, the notation $\Lambda_{\circledR(\Gamma'\vdash_{\lambda_{\circledR}}\alpha)}$ stands for $\{M \mid M \in \Lambda_{\circledR} \ \&\ \Gamma' \vdash_{\lambda_{\circledR}} M : \alpha\}$.

Proposition 5.55 (Type preservation by $\lfloor\ \rfloor$)

(i) If $\Gamma \vdash t : \sigma$, *then* $\Gamma \vdash_{\lambda_{\circledR}} \lfloor t\rfloor : \sigma$.

(ii) If $\Gamma; \cap_i^n \tau_i \vdash k : \sigma$, *then* $\lfloor k\rfloor_{\mathrm{k}} : \Lambda_{\circledR(\Delta_j\vdash_{\lambda_{\circledR}}\tau_j)} \to \Lambda_{\circledR(\Gamma,\Delta\vdash_{\lambda_{\circledR}}\sigma)}$, *for all* $j \in \{0,\dots,n\}$ *and for some* $\Delta = \Delta_0^{\top} \sqcap \Delta_1 \sqcap \dots \sqcap \Delta_n$.

Proof: The proposition is proved by simultaneous induction on derivations. We distinguish cases according to the last typing rule used.

- Cases (Ax), $(\to_R)$, $(Weak_t)$ and $(Cont_t)$ are easy, because the intersection type assignment system of $\lambda_{\circledR}$ has exactly the same rules.

- Case (Sel): the derivation ends with the rule

$$\frac{\Gamma, x:\alpha \vdash t:\sigma}{\Gamma;\alpha \vdash \widehat{x}.t:\sigma}\ (Sel)$$

By the IH we have that $\Gamma, x:\alpha \vdash_{\lambda_{®}} \lfloor t \rfloor : \sigma$, where $\alpha = \cap_i^n \tau_i$. For any $M \in \Lambda_{®}$ such that $\Delta_j \vdash_{\lambda_{®}} M:\tau_j$, for all $j \in \{0,\dots,n\}$, we have

$$\dfrac{\dfrac{\Gamma, x:\cap_i^n\tau_i \vdash_{\lambda_{®}} \lfloor t \rfloor : \sigma}{\Gamma \vdash_{\lambda_{®}} \lambda x.\lfloor t \rfloor : \cap_i^n\tau_i \to \sigma}\ (\to_I) \qquad \Delta_0 \vdash_{\lambda_{®}} M:\tau_0\ \dots\ \Delta_n \vdash_{\lambda_{®}} M:\tau_n}{\Gamma, \Delta_0^{\top} \sqcap \Delta_1 \sqcap \dots \sqcap \Delta_n \vdash_{\lambda_{®}} (\lambda x.\lfloor t \rfloor)M:\sigma}\ (\to_E)$$

Since $(\lambda x.\lfloor t \rfloor)M = \lfloor \widehat{x}.t \rfloor_{\mathrm{k}}(M)$, we conclude that

$$\lfloor \widehat{x}.t \rfloor_{\mathrm{k}} : \Lambda_{®(\Delta_j \vdash_{\lambda_{®}} \tau_j)} \to \Lambda_{®(\Gamma, \Delta_0^{\top} \sqcap \Delta_1 \sqcap \dots \sqcap \Delta_n \vdash_{\lambda_{®}} \sigma)}$$

.

- Case $(\to_L)$: the derivation ends with the rule

$$\frac{\Gamma_0 \vdash t:\sigma_0\ \dots\ \Gamma_n \vdash t:\sigma_n \quad \Delta;\cap_j^m\tau_j \vdash k:\rho}{\Gamma,\Delta;\cap_j^m(\cap_i^n\sigma_i \to \tau_j) \vdash t::k:\rho}\ (\to_L)$$

for $\Gamma = \Gamma_0^{\top} \sqcap \Gamma_1 \sqcap \dots \sqcap \Gamma_n$. By the IH we have that $\Gamma_l \vdash_{\lambda_{®}} \lfloor t \rfloor : \sigma_l$, for $l \in \{0,\dots,n\}$. For any $M \in \Lambda_{®}$ such that $\Gamma'_j \vdash_{\Lambda_{®}} M : \cap_i^n\sigma_i \to \tau_j$, $j = 1,\dots,m$ we have

$$\frac{\Gamma'_j \vdash_{\lambda_{®}} M:\cap_i^n\sigma_i \to \tau_j \quad \Gamma_0 \vdash_{\lambda_{®}} \lfloor t \rfloor:\sigma_0\ \dots\ \Gamma_n \vdash_{\lambda_{®}} \lfloor t \rfloor:\sigma_n}{\Gamma_0^{\top} \sqcap \Gamma_1 \sqcap \dots \sqcap \Gamma_n, \Gamma'_j \vdash_{\lambda_{®}} M\lfloor t \rfloor:\tau_j}\ (\to_E)$$

From the right-hand side premise in the $(\to_L)$ rule, by the IH, we get that $\lfloor k \rfloor_{\mathrm{k}}$ is the function with the scope $\lfloor k \rfloor_{\mathrm{k}} : \Lambda_{®(\Gamma'''_j \vdash_{\lambda_{®}} \tau_j)} \to \Lambda_{®(\Gamma''', \Gamma'' \vdash_{\lambda_{®}} \rho)}$, for some $\Gamma''' = {\Gamma'''_0}^{\top} \sqcap \Gamma'''_1 \sqcap \dots \sqcap \Gamma'''_n$. For $\Gamma''' \equiv \Gamma, \Gamma'$ and by taking $M\lfloor t \rfloor$ as the argument of the function $\lfloor k \rfloor_{\mathrm{k}}$, we get $\Gamma, \Delta, \Gamma' \vdash_{\lambda_{®}} \lfloor k \rfloor_{\mathrm{k}}(M\lfloor t \rfloor):\rho$. Since $\lfloor k \rfloor_{\mathrm{k}}(M\lfloor t \rfloor) = \lfloor t::k \rfloor_{\mathrm{k}}(M)$, we have that $\Gamma, \Delta, \Gamma' \vdash_{\lambda_{®}} \lfloor t::k \rfloor_{\mathrm{k}}(M):\rho$. This holds for any M of the appropriate type, yielding
$\lfloor t::k \rfloor_{\mathrm{k}} : \Lambda_{®(\Gamma' \vdash_{\lambda_{®}} \cap_i^n\sigma_i \to \tau_j)} \to \Lambda_{®(\Gamma, \Delta, \Gamma' \vdash_{\lambda_{®}} \rho)}$, which is exactly what we need.

- Case (Cut): the derivation ends with the rule

  $$\frac{\Gamma_0 \vdash t : \tau_0 \ldots \Gamma_n \vdash t : \tau_n \quad \Delta; \cap \tau_i^n \vdash k : \sigma}{\Gamma_0^\top \sqcap \Gamma_1 \sqcap \ldots \sqcap \Gamma_n, \Delta \vdash tk : \sigma}\ (Cut)$$

  By the IH we have that $\Gamma_j \vdash_{\lambda_{\circledR}} \lfloor t \rfloor : \tau_j$ and $\lfloor k \rfloor_{\mathrm{k}} : \Lambda_{\circledR}(\Gamma'_j \vdash_{\lambda_{\circledR}} \tau_j) \to \Lambda_{\circledR}(\Gamma', \Delta \vdash_{\lambda_{\circledR}} \sigma)$ for all $j = 0, \ldots, n$ and for $\Gamma' = \Gamma_0^\top \sqcap \Gamma'_1 \sqcap \ldots \sqcap \Gamma'_n$. Hence, for any $M \in \Lambda_{\circledR}$ such that $\Gamma'_j \vdash_{\lambda_{\circledR}} M : \tau_j$, $\Gamma', \Delta \vdash_{\lambda_{\circledR}} \lfloor k \rfloor_{\mathrm{k}}(M) : \sigma$ holds. By taking $M \equiv \lfloor t \rfloor$ and $\Gamma' \equiv \Gamma$, we get $\Gamma, \Delta \vdash_{\lambda_{\circledR}} \lfloor k \rfloor_{\mathrm{k}}(\lfloor t \rfloor) : \sigma$. But $\lfloor k \rfloor_{\mathrm{k}}(\lfloor t \rfloor) = \lfloor tk \rfloor$, so the proof is done.

- Case $(Weak_k)$: the derivation ends with the rule

  $$\frac{\Gamma; \beta \vdash k : \sigma}{\Gamma, x : \top; \beta \vdash x \odot k : \sigma}\ (Weak_k)$$

  where $\beta = \cap_j^n \tau_j$. By the IH we have that $\lfloor k \rfloor_{\mathrm{k}}$ is the function with the scope $\lfloor k \rfloor_{\mathrm{k}} : \Lambda_{\circledR}(\Gamma'_j \vdash_{\lambda_{\circledR}} \tau_j) \to \Lambda_{\circledR}(\Gamma, {\Gamma'_0}^\top \sqcap \Gamma'_1 \sqcap \ldots \sqcap \Gamma'_n \vdash_{\lambda_{\circledR}} \sigma)$, meaning that for each $M \in \Lambda_{\circledR}$ such that $\Gamma'_j \vdash_{\lambda_{\circledR}} M : \tau_j$ for all $j \in \{0, \ldots, n\}$ ${\Gamma'_0}^\top \sqcap \Gamma'_1 \sqcap \ldots \sqcap \Gamma'_n, \Gamma \vdash_{\lambda_{\circledR}} \lfloor k \rfloor_{\mathrm{k}}(M) : \sigma$ holds. Now, we can apply $(Weak)$ rule:

  $$\frac{\Gamma, {\Gamma'_0}^\top \sqcap \Gamma'_1 \sqcap \ldots \sqcap \Gamma'_n \vdash \lfloor k \rfloor_{\mathrm{k}}(M) : \sigma}{\Gamma, {\Gamma'_0}^\top \sqcap \Gamma'_1 \sqcap \ldots \sqcap \Gamma'_n, x : \top \vdash x \odot \lfloor k \rfloor_{\mathrm{k}}(M) : \sigma}\ (Weak)$$

  Since $x \odot \lfloor k \rfloor_{\mathrm{k}}(M) = \lfloor x \odot k \rfloor_{\mathrm{k}}(M)$, this means that $\lfloor x \odot k \rfloor_{\mathrm{k}} : \Lambda_{\circledR}(\Gamma'_j \vdash_{\lambda_{\circledR}} \tau_j) \to \Lambda_{\circledR}(\Gamma, {\Gamma'_0}^\top \sqcap \Gamma'_1 \sqcap \ldots \sqcap \Gamma'_n, x : \top \vdash_{\lambda_{\circledR}} \sigma)$, which is exactly what we wanted to get.

- Case $(Cont_k)$: similar to the case $(Weak_k)$, relying on the rule $(Cont)$ in the $\lambda_{\circledR}$-calculus. $\Box$

For the given encoding $\lfloor\ \rfloor$, we show that each $\lambda_{\circledR}^{\mathsf{Gtz}}$-reduction step can be simulated by some $\lambda_{\circledR}$-reduction or by an equality. In order to do so, we prove the following lemmas. The notation $\to_{\lambda_{\circledR}}$ stands for the union of all $\lambda_{\circledR}$-reductions, including (π_1) and (π_2).

Lemma 5.56 *If $M \to_{\lambda_{\circledR}} M'$, then $\lfloor k \rfloor_{\mathrm{k}}(M) \to_{\lambda_{\circledR}} \lfloor k \rfloor_{\mathrm{k}}(M')$.*

Proof: By induction on the structure of k.

- Basic case: $k \equiv \widehat{x}.t$.
  $M \to_{\lambda_{\circledR}} M'$ implies $(\lambda x. \lfloor t \rfloor)M \to_{\lambda_{\circledR}} (\lambda x. \lfloor t \rfloor)M'$. Since $\lfloor \widehat{x}.t \rfloor_{\mathrm{k}}(M) = (\lambda x. \lfloor t \rfloor)M$, the statement is proved.

- Case: $k \equiv t :: k'$.
  $M \to_{\lambda_®} M'$ implies $M\lfloor t \rfloor \to_{\lambda_®} M'\lfloor t \rfloor$. By the IH $\lfloor k' \rfloor_{\mathrm{k}}(M\lfloor t \rfloor) \to_{\lambda_®} \lfloor k' \rfloor_{\mathrm{k}}(M'\lfloor t \rfloor)$. $\lfloor t :: k' \rfloor_{\mathrm{k}}(M) = \lfloor k' \rfloor_{\mathrm{k}}(M\lfloor t \rfloor)$, hence the proof is done.
- Cases $k \equiv x \odot k'$ and $k \equiv x <^{x_1}_{x_2} k'$ are analogous. □

Lemma 5.57 $\lfloor k \rfloor_{\mathrm{k}}((\lambda x.P)N) \to_{\lambda_®} (\lambda x.\lfloor k \rfloor_{\mathrm{k}}(P))N$.

Proof: By induction on the structure of k, analogous to the proof of Lemma 4.36.□

Lemma 5.58 *If $M \in \Lambda^{®}$ and $k, k' \in \mathrm{C}^{\mathsf{Gtz}}_{®}$, then* $\lfloor k' \rfloor_{\mathrm{k}} \circ \lfloor k \rfloor_{\mathrm{k}}(M) \to_{\lambda_®} \lfloor k@k' \rfloor_{\mathrm{k}}(M)$.

Proof: By induction on the structure of k, analogous to the proof of Lemma 4.37.□

Lemma 5.59

(i) If $x \notin Fv(k)$, then $(\lfloor k \rfloor_{\mathrm{k}}(M))[N/x] = \lfloor k \rfloor_{\mathrm{k}}(M[N/x])$.

(ii) If $x, y \notin Fv(k)$, then $z <^x_y (\lfloor k \rfloor_{\mathrm{k}}(M)) \to_{\lambda_®} \lfloor k \rfloor_{\mathrm{k}}(z <^x_y M)$.

(iii) $\lfloor k \rfloor_{\mathrm{k}}(x \odot M) \to_{\lambda_®} x \odot \lfloor k \rfloor_{\mathrm{k}}(M)$.

Proof:

(i) By induction on the structure of k.

- Basic case: $k \equiv \widehat{y}.t$, where $x \notin Fv(t)$.

$$\begin{array}{rcl} (\lfloor k \rfloor_{\mathrm{k}}(M))[N/x] & = & (\lfloor \widehat{y}.t \rfloor_{\mathrm{k}}(M))[N/x] \\ & = & ((\lambda y.\lfloor t \rfloor)M)[N/x] \\ & \triangleq & (\lambda y.\lfloor t \rfloor)(M[N/x]) \\ & = & \lfloor \widehat{y}.t \rfloor_{\mathrm{k}}(M[N/x]) \\ & = & \lfloor k \rfloor_{\mathrm{k}}(M[N/x]). \end{array}$$

- Case: $k \equiv t :: k'$, where $x \notin Fv(t) \cup Fv(k')$.

$$\begin{array}{rcl} (\lfloor k \rfloor_{\mathrm{k}}(M))[N/x] & = & (\lfloor t :: k' \rfloor_{\mathrm{k}}(M))[N/x] \\ & = & (\lfloor k' \rfloor_{\mathrm{k}}(M\lfloor t \rfloor))[N/x] \\ & =_{IH} & \lfloor k' \rfloor_{\mathrm{k}}((M\lfloor t \rfloor)[N/x]) \\ & \triangleq & \lfloor k' \rfloor_{\mathrm{k}}((M[N/x]\lfloor t \rfloor)) \\ & = & \lfloor t :: k' \rfloor_{\mathrm{k}}(M[N/x]) \\ & = & \lfloor k \rfloor_{\mathrm{k}}(M[N/x]). \end{array}$$

- Case: $k \equiv y <^{z}_{w} k'$, where $x \notin Fv(k')$.

$$\begin{array}{rcl}
(\lfloor k \rfloor_{\mathrm{k}}(M))[N/x] & = & (\lfloor y <^{z}_{w} k' \rfloor_{\mathrm{k}}(M))[N/x] \\
& = & (y <^{z}_{w} \lfloor k' \rfloor_{\mathrm{k}}(M))[N/x] \\
& \triangleq & y <^{z}_{w} (\lfloor k' \rfloor_{\mathrm{k}}(M))[N/x] \\
& =_{IH} & y <^{z}_{w} \lfloor k' \rfloor_{\mathrm{k}}(M[N/x]) \\
& = & \lfloor y <^{z}_{w} k' \rfloor_{\mathrm{k}}(M[N/x]) \\
& = & \lfloor k \rfloor_{\mathrm{k}}(M[N/x]).
\end{array}$$

- Case: $k \equiv y \odot k'$, where $x \notin Fv(k')$ is similar.

(ii) By induction on the structure of k.

- Basic case: $k \equiv \widehat{w}.t$, where $x, y \notin Fv(t)$.

$$\begin{array}{rcl}
z <^{x}_{y} (\lfloor k \rfloor_{\mathrm{k}}(M)) & = & z <^{x}_{y} (\lfloor \widehat{w}.t \rfloor_{\mathrm{k}}(M)) \\
& = & z <^{x}_{y} ((\lambda w.\lfloor t \rfloor)M) \\
& \to_{\lambda_{®}} & (\lambda w.\lfloor t \rfloor)z <^{x}_{y} M \\
& = & \lfloor \widehat{w}.t \rfloor_{\mathrm{k}}(z <^{x}_{y} M) \\
& = & \lfloor k \rfloor_{\mathrm{k}}(z <^{x}_{y} M).
\end{array}$$

- Case: $k \equiv t :: k'$, where $x, y \notin Fv(t) \cup Fv(k')$.

$$\begin{array}{rcl}
z <^{x}_{y} (\lfloor k \rfloor_{\mathrm{k}}(M)) & = & z <^{x}_{y} (\lfloor t :: k' \rfloor_{\mathrm{k}}(M)) \\
& = & z <^{x}_{y} (\lfloor k' \rfloor_{\mathrm{k}}(M\lfloor t \rfloor)) \\
& \to_{IH} & \lfloor k' \rfloor_{\mathrm{k}}(z <^{x}_{y} (M\lfloor t \rfloor)) \\
& \to_{\lambda_{®}} & \lfloor k' \rfloor_{\mathrm{k}}((z <^{x}_{y} M)\lfloor t \rfloor) \\
& = & \lfloor t :: k' \rfloor_{\mathrm{k}}(z <^{x}_{y} M) \\
& = & \lfloor k \rfloor_{\mathrm{k}}(z <^{x}_{y} M).
\end{array}$$

(iii) By easy induction on the structure of k, since we know that $x \notin Fv(k)$, because otherwise the result of the mapping $\lfloor k \rfloor_{\mathrm{k}}(x \odot M)$ would not be the $\lambda_{®}$-term. □

Now we can prove that the reduction rules of the $\lambda_{®}^{\mathsf{Gtz}}$-calculus can be simulated by the reduction rules or an equality in the $\lambda_{®}$-calculus. Moreover, the equivalences of the $\lambda_{®}^{\mathsf{Gtz}}$-calculus are preserved in the $\lambda_{®}$-calculus.

Theorem 5.60 (Simulation of $\lambda_{®}^{\mathsf{Gtz}}$-reduction by $\lambda_{®}$-reduction)

(i) If a term $t \to_{\lambda_{®}^{\mathsf{Gtz}}} t'$, then $\lfloor t \rfloor \to_{\lambda_{®}} \lfloor t' \rfloor$.

(ii) If a context $k \to_{\lambda_{®}^{\mathsf{Gtz}}} k'$ by (γ_6) or (ω_6) reduction, then $\lfloor k \rfloor_{\mathrm{k}}(M) \equiv \lfloor k' \rfloor_{\mathrm{k}}(M)$, for any $M \in \Lambda^{\lambda_{®}}$.

(iii) *If a context $k \to_{\lambda_{®}^{\mathsf{Gtz}}} k'$ by a reduction different from (γ_6) or (ω_6), then $\lfloor k \rfloor_{\mathrm{k}}(M) \to_{\lambda_{®}} \lfloor k' \rfloor_{\mathrm{k}}(M)$, for any $M \in \Lambda^{®}$.*

(iv) *If $t \equiv_{\lambda_{®}^{\mathsf{Gtz}}} t'$, then $\lfloor t \rfloor \equiv_{\lambda_{®}} \lfloor t' \rfloor$, and if $k \equiv_{\lambda_{®}^{\mathsf{Gtz}}} k'$, then $\lfloor k \rfloor_{\mathrm{k}}(M) \equiv_{\lambda_{®}} \lfloor k' \rfloor_{\mathrm{k}}(M)$, for any $M \in \Lambda_{®}$.*

Proof: Without losing generality, we prove the statement only for the outermost reductions. We will use the notation $\to$ instead of $\to_{\lambda_{®}^{\mathsf{Gtz}}}$ since it is clear from the context where the reductions happen.

(β) $(\lambda x.v)(u :: k) \to u(\widehat{x}.vk)$.

On the one hand we have

$\lfloor t \rfloor = \lfloor (\lambda x.v)(u :: k) \rfloor = \lfloor u :: k \rfloor_{\mathrm{k}}(\lfloor \lambda x.v \rfloor) = \lfloor k \rfloor_{\mathrm{k}}((\lambda x.\lfloor v \rfloor)\lfloor u \rfloor)$

On the other hand,

$\lfloor t' \rfloor = \lfloor u(\widehat{x}.vk) \rfloor = \lfloor \widehat{x}.vk \rfloor_{\mathrm{k}}(\lfloor u \rfloor) = (\lambda x.\lfloor vk \rfloor)\lfloor u \rfloor = (\lambda x.\lfloor k \rfloor_{\mathrm{k}}(\lfloor v \rfloor))\lfloor u \rfloor$.

So, $\lfloor t \rfloor \to_{\lambda_{®}} \lfloor t' \rfloor$ by Lemma 5.57.

(σ) $T(\widehat{x}.v) \to v[T/x]$

$\lfloor t \rfloor = \lfloor T(\widehat{x}.v) \rfloor = \lfloor \widehat{x}.v \rfloor_{\mathrm{k}}(\lfloor T \rfloor) = (\lambda x.\lfloor v \rfloor)\lfloor T \rfloor$

On the other hand, by Lemma 5.54 $\lfloor t' \rfloor = \lfloor v[T/x] \rfloor = \lfloor v \rfloor[\lfloor T \rfloor/x]$.

Thus $\lfloor t \rfloor \to_{\lambda_{®}} \lfloor t' \rfloor$ by (β)-reduction together with Lemma 5.54.

(π) $(vk)k' \to v(k@k')$

$\lfloor t \rfloor = \lfloor (vk)k' \rfloor = \lfloor k' \rfloor_{\mathrm{k}}(\lfloor vk \rfloor) = \lfloor k' \rfloor_{\mathrm{k}}(\lfloor k \rfloor_{\mathrm{k}}(\lfloor v \rfloor))$

$\lfloor t' \rfloor = \lfloor v(k@k') \rfloor = \lfloor k@k' \rfloor_{\mathrm{k}}(\lfloor v \rfloor)$.

Applying Lemma 5.58 we get that $\lfloor t \rfloor \to_{\lambda_{®}} \lfloor t' \rfloor$.

(μ) $\widehat{x}.xk \to k$.

This reduction reduces a context to a context, so we have:

$\lfloor k \rfloor_{\mathrm{k}}(M) = \lfloor \widehat{x}.xk'' \rfloor_{\mathrm{k}}(M) = (\lambda x.\lfloor xk'' \rfloor)M = (\lambda x.\lfloor k'' \rfloor_{\mathrm{k}}(x))M$ and $\lfloor k' \rfloor_{\mathrm{k}}(M) = \lfloor k'' \rfloor_{\mathrm{k}}(M)$.

Now, $(\lambda x.\lfloor k'' \rfloor_{\mathrm{k}}(x))M$ reduces by (β)-reduction to $(\lfloor k'' \rfloor_{\mathrm{k}}(x))[M/x]$. Since we know that $x \notin k''$, which is a side condition for the (μ) reduction, we can apply Lemma 5.59 and get $(\lfloor k'' \rfloor_{\mathrm{k}}(x))[M/x] = \lfloor k'' \rfloor_{\mathrm{k}}(x[M/x]) = \lfloor k'' \rfloor_{\mathrm{k}}(M)$. So $\lfloor k \rfloor_{\mathrm{k}}(M) \to \lfloor k' \rfloor_{\mathrm{k}}(M)$ by (β)-reduction followed by Lemma 5.59.

(γ_1) $\lfloor x <^{x_1}_{x_2} (\lambda y.t) \rfloor \to_{\lambda_{®}} \lfloor \lambda y.x <^{x_1}_{x_2} t \rfloor$ by the rule (γ_1) in $\lambda_{®}$.

(γ_2) $\lfloor x <^{x_1}_{x_2} (tk)\rfloor \to_{\lambda_{®}} \lfloor (x <^{x_1}_{x_2} t)k\rfloor$ by the rule (γ_2) in $\lambda_{®}$.

(γ_3) $\lfloor x <^{x_1}_{x_2} (tk)\rfloor \to_{\lambda_{®}} \lfloor t(x <^{x_1}_{x_2} k)\rfloor$ by the rule (γ_3) in $\lambda_{®}$.

(γ_4) $x <^{x_1}_{x_2} (\widehat{y}.t) \to \widehat{y}.(x <^{x_1}_{x_2} t)$

$\lfloor k\rfloor_{\mathrm{k}}(M) = (x <^{x_1}_{x_2} \lambda y.\lfloor t\rfloor)M.$

On the other hand,

$\lfloor k'\rfloor_{\mathrm{k}}(M) = (\lambda y.x <^{x_1}_{x_2} \lfloor t\rfloor)M.$

So $\lfloor k\rfloor_{\mathrm{k}}(M) \to_{\lambda_{®}} \lfloor k'\rfloor_{\mathrm{k}}(M)$ by (γ_1)-reduction.

(γ_5) $x <^{x_1}_{x_2} (t :: k'') \to (x <^{x_1}_{x_2} t) :: k''$, if $x_1, x_2 \in Fv(t)$

$\lfloor k\rfloor_{\mathrm{k}}(M) = x <^{x_1}_{x_2} (\lfloor k''\rfloor_{\mathrm{k}}(M\lfloor t\rfloor)).$

$\lfloor k'\rfloor_{\mathrm{k}}(M) = \lfloor k''\rfloor_{\mathrm{k}}(M(x <^{x_1}_{x_2} \lfloor t\rfloor)).$

$x_1, x_2 \in Fv(t)$ implies that $x_1, x_2 \in Fv(M\lfloor t\rfloor)$ so we can apply Lemma 5.59 followed by reduction (γ_3) and conclude that $\lfloor k\rfloor_{\mathrm{k}}(M) \to_{\lambda_{®}} \lfloor k'\rfloor_{\mathrm{k}}(M)$.

(γ_6) $x <^{x_1}_{x_2} (t :: k'') \to t :: (x <^{x_1}_{x_2} k'')$, if $x_1, x_2 \in Fv(k'')$

$\lfloor k\rfloor_{\mathrm{k}}(M) = \lfloor x <^{x_1}_{x_2} (t :: k'')\rfloor_{\mathrm{k}}(M) = x <^{x_1}_{x_2} \lfloor k''\rfloor_{\mathrm{k}}(M\lfloor t\rfloor).$

On the other hand,

$\lfloor k'\rfloor_{\mathrm{k}}(M) = \lfloor t :: (x <^{x_1}_{x_2} k'')\rfloor_{\mathrm{k}}(M) = x <^{x_1}_{x_2} \lfloor k''\rfloor_{\mathrm{k}}(M\lfloor t\rfloor).$

So $\lfloor k\rfloor_{\mathrm{k}}(M) \equiv \lfloor k'\rfloor_{\mathrm{k}}(M)$.

($\gamma\omega_1$) $x <^{x_1}_{x_2} (y \odot e) \to y \odot (x <^{x_1}_{x_2} e)$

There is the corresponding ($\gamma\omega_1$) rule in $\lambda_{®}$.

($\gamma\omega_2$) $x <^{x_1}_{x_2} (x_1 \odot e) \to e[x/x_2]$

There is the corresponding ($\gamma\omega_2$) rule in $\lambda_{®}$.

(ω_1) $\lfloor \lambda x.(y \odot t)\rfloor \to_{\lambda_{®}} \lfloor y \odot (\lambda x.t)\rfloor$ by the rule (ω_1) in $\lambda_{®}$.

(ω_2) $\lfloor (x \odot t)k\rfloor \to_{\lambda_{®}} \lfloor x \odot (tk)\rfloor$ by the rule (ω_2) in $\lambda_{®}$.

(ω_3) $\lfloor (x \odot t)k\rfloor \to_{\lambda_{®}} \lfloor x \odot (tk)\rfloor$ by the rule (ω_3) in $\lambda_{®}$.

(ω_4) $\widehat{x}.(y \odot t) \to y \odot (\widehat{x}.t),\ x \neq y$

$\lfloor k\rfloor_{\mathrm{k}}(M) = \lfloor \widehat{x}.(y \odot t)\rfloor_{\mathrm{k}}(M) = (\lambda x.y \odot \lfloor t\rfloor)M.$

$\lfloor k'\rfloor_{\mathrm{k}}(M) = \lfloor y \odot (\widehat{x}.t)\rfloor_{\mathrm{k}}(M) = (y \odot \lambda x.\lfloor t\rfloor)M.$

So $\lfloor k\rfloor_{\mathrm{k}}(M) \to_{\lambda_{®}} \lfloor k'\rfloor_{\mathrm{k}}(M)$ by the rule (ω_1) in $\lambda_{®}$.

(ω_5) $(x \odot t) :: k'' \to x \odot (t :: k'')$

$\lfloor k \rfloor_{\mathrm{k}}(M) = \lfloor (x \odot t) :: k'' \rfloor_{\mathrm{k}}(M) = \lfloor k'' \rfloor_{\mathrm{k}}(M \lfloor x \odot t \rfloor) = \lfloor k'' \rfloor_{\mathrm{k}}(Mx \odot \lfloor t \rfloor)$.

$\lfloor k' \rfloor_{\mathrm{k}}(M) = \lfloor x \odot (t :: k'') \rfloor_{\mathrm{k}}(M) = x \odot \lfloor t :: k'' \rfloor_{\mathrm{k}}(M) = x \odot \lfloor k'' \rfloor_{\mathrm{k}}(M \lfloor t \rfloor)$.

Applying the rule (ω_3) from the $\lambda_{®}$-calculus and Lemma 5.59 we get that

$\lfloor k \rfloor_{\mathrm{k}}(M) \twoheadrightarrow_{\lambda_{®}} \lfloor k' \rfloor_{\mathrm{k}}(M)$.

(ω_6) $t :: (x \odot k'') \to x \odot (t :: k'')$

$\lfloor k \rfloor_{\mathrm{k}}(M) = \lfloor t :: (x \odot k'') \rfloor_{\mathrm{k}}(M) = \lfloor x \odot k'' \rfloor_{\mathrm{k}}(M \lfloor t \rfloor) = x \odot \lfloor k'' \rfloor_{\mathrm{k}}(M \lfloor t \rfloor) = \lfloor k' \rfloor_{\mathrm{k}}(M)$.
So $\lfloor k \rfloor_{\mathrm{k}}(M) \equiv \lfloor k' \rfloor_{\mathrm{k}}(M)$.

($\varepsilon_1 - \varepsilon_4$) are trivial, because there exist corresponding equivalence rules in the $\lambda_{®}$-calculus. □

We see that most of the $\lambda_{®}^{\mathsf{Gtz}}$-reductions, more precisely all except (γ_6) and (ω_6) reductions, are interpreted by the $\lambda_{®}$-reductions. Since the set of equivalences of the two calculi coincide, they are trivially preserved. As in the case of simply-typed $\lambda_{®}^{\mathsf{Gtz}}$ calculus, in order to prove that there is no infinite sequence of $\lambda_{®}^{\mathsf{Gtz}}$-reductions one has to prove that there cannot exist an infinite sequence of $\lambda_{®}^{\mathsf{Gtz}}$-reductions which are all interpreted as equalities. For that purpose, we will use special measures proposed in Section 5.2.2, namely $\| \, \|_C$ (Definition 5.29) and $\| \, \|_W$ (Definition 5.30).

Lemma 5.61 *For all $e, e' \in \Lambda_{®}^{\mathsf{Gtz}}$:*

(i) If $e \to_{\omega_6} e'$, then $\|e\|_W > \|e'\|_W$.

(ii) If $e \to_{\gamma_6} e'$, then $\|e\|_W = \|e'\|_W$.

(iii) If $e \equiv_{\lambda_{®}^{\mathsf{Gtz}}} e'$, then $\|e\|_W = \|e'\|_W$.

Proof: Included in the proof of Lemma 5.37. □

Lemma 5.62

(i) For all $e, e' \in \Lambda_{®}^{\mathsf{Gtz}}$: if $e \to_{\gamma_6} e'$, then $\|e\|_C > \|e'\|_C$.

(ii) If $e \equiv_{\lambda_{®}^{\mathsf{Gtz}}} e'$, then $\|e\|_C = \|e'\|_C$.

Proof: Included in the proof of Lemma 5.38. □

Now, we define a lexicographic product based on the orders defined in Definition 5.39.

Definition 5.63 *We define the relation $\gg$ on $\Lambda_{®}^{\mathsf{Gtz}}$ as the lexicographic product:*

$$\gg \;=\; >_{\lambda_{®}} \times_{lex} >_{w} \times_{lex} >_{c} .$$

The following propositions proves that the reduction relation on the set of typed $\lambda_{®}^{\mathsf{Gtz}}$-expressions is included in the given lexicographic product $\gg$.

Proposition 5.64 *For each $e \in \Lambda_{®}^{\mathsf{Gtz}}$: if $e \to e'$, then $e \gg e'$.*

Proof: The proof is by case analysis on the kind of reduction and the structure of $\gg$.
If $e \to e'$ by β, σ, π, μ, γ_1, γ_2, γ_3, γ_4 γ_5, $\gamma\omega_1$, $\gamma\omega_2$, ω_1, ω_2, ω_3 ω_4 or ω_5 reduction, then $e >_{\lambda_{®}} e'$ by Proposition 5.60.
If $e \to e'$ by ω_6, then $e =_{\lambda_{®}} e'$ by Proposition 5.60 and $e >_{w} e'$ by Lemma 5.61.
Finally, if $e \to e'$ by γ_6, then $e =_{\lambda_{®}} e'$ by Proposition 5.60, $e =_{w} e'$ by Lemma 5.61 and $e >_{c} e'$ by Lemma 5.62.□

Finally, we can prove the desired feature of the $\lambda_{®}^{\mathsf{Gtz}}$-calculus with intersection types.

Theorem 5.65 (Strong normalisation of $\lambda_{®}^{\mathsf{Gtz}}$) *If a $\lambda_{®}^{\mathsf{Gtz}}$-expression is typeable in the system $\lambda_{®}^{\mathsf{Gtz}}\cap$, then it is SN.*

Proof: The reduction $\to$ is well-founded on $\lambda_{®}^{\mathsf{Gtz}}\cap$ as it is included (Proposition 5.64) in the relation $\gg$ which is well-founded as the lexicographic product of the well-founded relations $>_{\lambda_{®}}$, $>_{w}$ and $>_{c}$. Relation $>_{\lambda_{®}}$ is based on the interpretation $\lfloor\ \rfloor : \mathrm{T}_{®}^{\mathsf{Gtz}} \to \Lambda_{®}$. By Proposition 5.55 typeability is preserved by the interpretation $\lfloor\ \rfloor$ and $\to_{\lambda_{®}}$ is SN, i.e. well-founded, on $\Lambda_{®}\cap$ (Propositions 3.13 and 5.51), hence $>_{\lambda_{®}}$ is well-founded on $\lambda_{®}^{\mathsf{Gtz}}\cap$. Similarly, $>_{c}$ and $>_{w}$ are well-founded, as they are based on interpretations into the well-founded relation $>$ on the set $\mathbb{N}$ of natural numbers. □

5.3.3 SN $\Rightarrow$ Typeability in $\lambda_{®}^{\mathsf{Gtz}}\cap$

The next step toward the complete characterisation of the strongly normalising $\lambda_{®}^{\mathsf{Gtz}}$-expressions is to prove that if a $\lambda_{®}^{\mathsf{Gtz}}$-expression is SN, then it is typeable in the system $\lambda_{®}^{\mathsf{Gtz}}\cap$. Like in the case of the λ^{Gtz}-calculus (Section 4.3.2) we proceed in two steps:

1) we show that all $\lambda_{®}^{\mathsf{Gtz}}$-normal forms are typeable; and

2) we prove that typeability is preserved by head subject expansion.

In order to observe the structure of the $\lambda_{®}^{\mathsf{Gtz}}$-normal forms, let us revisit the $\lambda_{®}^{\mathsf{Gtz}}$-reductions, presented in Figure 5.2. The computation is directed towards:

- cut elimination;
- contraction propagation;
- weakening extraction;

and each of these three processes produces its own normal forms.

Cut elimination is performed by original λ^{Gtz}-reductions, namely β, σ and π, thus the corresponding normal forms are expressions of the form

$$x,\ \lambda x.t_{nf},\ x(t_{nf} :: k_{nf}),\ t_{nf} :: k_{nf},\ \widehat{x}.t_{nf},$$

i.e. λ^{Gtz}-normal forms.

Contraction propagation is performed by the group of γ-reductions, and the corresponding normal forms are expressions of the form

$$x <^{y}_{z} y(t_{nf} :: k_{nf}),\ z \in Fv(k_{nf}) \quad \text{and} \quad x <^{y}_{z} t_{nf} :: k_{nf},\ y \in Fv(t_{nf}), z \in Fv(k_{nf}),$$

i.e. contractions over expressions consisting of two parts, where each one contains exactly one of two contracted variables, and which cannot further interact.

Weakening extraction is performed by the group of ω-reductions, and the corresponding normal forms are expressions of the form

$$x \odot e_{nf},\ \lambda x.x \odot t_{nf},\ \widehat{x}..x \odot t_{nf},$$

i.e. expressions in which weakenings are either pushed out to the surface or blocked by binding (abstraction or selection).

Therefore, the $\lambda_{®}^{\mathsf{Gtz}}$-normal forms are the subset of the set $\mathbb{E}$ of $\lambda_{®}^{\mathsf{Gtz}}$-expressions given by the following abstract syntax:

$$\begin{array}{rcl}
t_{nf} & ::= & x \,|\, \lambda x.t_{nf} \,|\, \lambda x.x \odot t_{nf} \,|\, x(t_{nf} :: k_{nf}) \,|\, x <^{y}_{z} y(t_{nf} :: k_{nf}) \\
k_{nf} & ::= & \widehat{x}.t_{nf} \,|\, \widehat{x}.x \odot t_{nf} \,|\, t_{nf} :: k_{nf} \,|\, x <^{y}_{z} (t_{nf} :: k_{nf}),\ y \in Fv(t_{nf}), z \in Fv(k_{nf}) \\
w_{nf} & ::= & x \odot e_{nf} \\
e_{nf} & ::= & t_{nf} \,|\, k_{nf} \,|\, w_{nf}.
\end{array}$$

Notice that $\mathbb{E}$ is not exactly the set of $\lambda_{®}^{\mathsf{Gtz}}$-normal forms since there exist expressions that belong to $\mathbb{E}$ but are not normal forms (such as the context $\widehat{x}.x(t_{nf} :: k_{nf})$ which can be reduced by μ-reduction), yet it contains all normal forms, thus it is sufficient to prove the following proposition.

Proposition 5.66 *If a $\lambda_{®}^{\mathsf{Gtz}}$-expression $e_{nf} \in \mathbb{E}$, then e_{nf} is typeable in the system $\lambda_{®}^{\mathsf{Gtz}}\cap$. Therefore, all $\lambda_{®}^{\mathsf{Gtz}}$-normal forms are typeable in the system $\lambda_{®}^{\mathsf{Gtz}}\cap$.*

Proof: The proof goes by induction on the structure of e_{nf}. All cases are straightforward, because we know that e_{nf} are well-formed, and that domain correspondence holds (Lemma 5.45). □

The following two lemmas explain the behavior of the meta operators $[\ /\ ]$ and @ during the expansion.

Lemma 5.67 (Inverse substitution lemma for $\lambda_{®}^{\mathsf{Gtz}}$) *(i) Let $\Gamma \vdash t[T/x] : \sigma$ and T typeable, i.e. $\Delta_j \vdash T : \tau_j$, for $j \in \{0,...,n\}$ and $n \geq 0$. Then $\Gamma', x : \cap_i^n \tau_i \vdash t : \sigma$, where $\Gamma = \Gamma', \Delta_0^\top \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n$.*

(ii) Let $\Gamma; \gamma \vdash k[T/x] : \sigma$ and T typeable, i.e. $\Delta_j \vdash T : \tau_j$, for $j \in \{0,...,n\}$ and $n \geq 0$. Then $\Gamma', x : \cap_i^n \tau_i; \gamma \vdash k : \sigma$, where $\Gamma = \Gamma', \Delta_0^\top \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n$.

Proof: By mutual induction on the structure of the term t and the context k. We will only show the base case and some resource related cases.

- Base case $t \equiv x$. Then $t[T/x] \equiv x[T/x] \triangleq T$, thus the assumption yields $\Gamma \vdash T : \sigma$ and $\Delta_j \vdash T : \tau_j$, for all $j \in \{0,...,n\}$. Therefore $\Gamma = \Delta_0^\top \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n$, $\Gamma' = \emptyset$, $\sigma \equiv \tau_{j_0}$, for some $j_0 \in \{0,...,n\}$ and $x : \cap_i^n \tau_i \vdash t : \sigma$, by the axiom.

- Case $t \equiv x \odot u$. Then $t[T/x] \equiv (x \odot u)[T/x] \triangleq Fv(T) \odot u$, thus the assumption yields $\Gamma \vdash Fv(T) \odot u : \sigma$ and $\Delta_j \vdash T : \tau_j$, for all $j \in \{0,...,n\}$. Let $Fv(T) = \{y_1,...,y_m\} = Dom(\Delta_j)$. By Generation lemma 5.46(vi) we get that $\Gamma = \Gamma', y_1 : \top, ..., y_m : \top$ and $\Gamma' \vdash u : \sigma$. Therefore, for $n = 0$ we get what we need: $\Gamma = \Gamma', \Delta_0^\top$ and $\Gamma', x : \cap_i^0 \tau_i \vdash x \odot u : \sigma$. The last statement is correct since $\cap_i^0 \tau_i = \top$.

- Case $t \equiv y \odot u$. Then $t[T/x] \equiv (y \odot u)[T/x] \triangleq y \odot u[T/x]$, thus the assumption yields $\Gamma \vdash y \odot u[T/x] : \sigma$ and $\Delta_j \vdash T : \tau_j$, for all $j \in \{0,...,n\}$. By Generation lemma 5.46(vi) we get that $\Gamma = \Gamma', y : \top$ and $\Gamma' \vdash u[T/x] : \sigma$. Now, by the IH we get that $\Gamma' = \Gamma'', \Delta_0^\top \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n$ and $\Gamma'', x : \cap_i^n \tau_i \vdash u : \sigma$. Therefore, $\Gamma'', y : \top, x : \cap_i^n \tau_i \vdash y \odot u : \sigma$ and $\Gamma = \Gamma'', y : \top, \Delta_0^\top \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n$, which is exactly what we needed to prove.

- Case $t \equiv x <^{x_1}_{x_2} u$. Then $t[T/x] \equiv (x <^{x_1}_{x_2} u)[T/x] \triangleq Fv(T) <^{Fv(T_1)}_{Fv(T_2)} (u[T_1/x_1])[T_2/x_2]$. The assumptions are in this case $\Gamma \vdash Fv(T) <^{Fv(T_1)}_{Fv(T_2)} (u[T_1/x_1])[T_2/x_2] : \sigma$ and $\Delta_j \vdash T : \tau_j$, for all $j \in \{0,...,n\}$. Since T_1 and T_2 are obtained by renaming free variables of T, we can also assume that $\Delta'_j \vdash T_1 : \tau_j$, and $\Delta''_j \vdash T_2 : \tau_j$, for all $j \in \{0,...,n\}$. Let $Fv(T) = \{y_1,...,y_m\} = Dom(\Delta_j)$, $Fv(T_1) = \{y'_1,...,y'_m\} = Dom(\Delta'_j)$ and $Fv(T_2) = \{y''_1,...,y''_m\} = Dom(\Delta''_j)$. By m applications of Generation lemma 5.46(v) we get that $\Gamma = \Gamma', y_1 : \alpha_1 \cap \beta_1, ..., y_m : \alpha_m \cap \beta_m$ and $\Gamma', y'_1 : \alpha_1, ..., y'_m : \alpha_m, y''_1 : \beta_1, ..., y''_m : \beta_m \vdash (u[T_1/x_1])[T_2/x_2] : \sigma$. Since T_1, T_2 and u by construction have disjoint sets of free variables, by two applications of the IH we obtain $\Gamma', x_1 : \cap_i^n \tau_i, x_2 : \cap_i^n \tau_i \vdash u : \sigma$. Finally, due to the fact that intersection is idempotent we obtain $\Gamma', x : \cap_i^n \tau_i \vdash x <^{x_1}_{x_2} u : \sigma$. □

Lemma 5.68 (Inverse append lemma for $\lambda_{\circledR}^{\mathsf{Gtz}}$) *If* $\Gamma; \alpha \vdash k@k' : \sigma$, *then there are* Δ_j *and* τ_j, $j = 0, \dots, n$ *such that* $\Delta_j; \alpha \vdash k : \tau_j$ *and* $\Gamma'; \cap_i^n \tau_i \vdash k' : \sigma$, *where* $\Gamma = \Gamma', \Delta_0^\top \sqcap \Delta_1 \sqcap \dots \sqcap \Delta_n$.

Proof: The proof goes by the induction on the structure of the context k.

- Base case $k \equiv \widehat{x}.t$. In this case $\widehat{x}.t@k' \triangleq \widehat{x}.tk'$, thus by assumption $\Gamma; \alpha \vdash \widehat{x}.tk' : \sigma$. By Generation lemma 5.46(iv) we get $\Gamma, x : \alpha \vdash tk' : \sigma$. By Generation lemma 5.46(iii) we get that $\Gamma, x : \alpha = \Gamma', \Delta_0'^\top \sqcap \Delta'_1 \sqcap \dots \sqcap \Delta'_n$, $\Gamma'; \cap_i^n \tau_i \vdash k' : \sigma$ and $\Delta'_j \vdash t : \tau_j$ for $j = 0, \dots, n$. Now, since $\widehat{x}.t$ implies that $x \in Fv(t)$ and because of the domain correspondence (Lemma 5.45), we have that for all $j = 0, \dots, n$ it holds $\Delta'_j = \Delta_j, x : \alpha$. Finally, from $\Delta_j, x : \alpha \vdash t : \tau_j$ by (Sel) we get $\Delta_j; \alpha \vdash \widehat{x}.t : \tau_j$, which is exactly what we wanted. Moreover, from $\Gamma, x : \alpha = \Gamma', (\Delta_0, x : \alpha)^\top \sqcap (\Delta_1, x : \alpha) \sqcap \dots \sqcap (\Delta_n, x : \alpha)$ we get that $\Gamma = \Gamma', \Delta_0^\top \sqcap \Delta_1 \sqcap \dots \sqcap \Delta_n$ as required.

- Case $k \equiv x \odot k''$. In this case $(x \odot k'')@k' \triangleq x \odot (k''@k')$, thus by assumption $\Gamma; \alpha \vdash x \odot (k''@k') : \sigma$. By Generation lemma 5.46($viii$) we get $\Gamma = \Gamma'', \top$ and $\Gamma''; \alpha \vdash k''@k' : \sigma$. Now by induction hypothesis $\Gamma'' = \Gamma', \Delta_0^\top \sqcap \Delta_1 \sqcap \dots \sqcap \Delta_n$, $\Delta_j; \alpha \vdash k'' : \tau_j$ for $j = 0, \dots, n$ and $\Gamma'; \cap_i^n \tau_i \vdash k' : \sigma$. Applying $(Weak_k)$ rule on $\Delta_j; \alpha \vdash k'' : \tau_j$ for $j = 0, \dots, n$, we get $\Delta_j, x : \top; \alpha \vdash x \odot k'' : \tau_j$.

- Cases $k \equiv x <^y_z k''$ and $k \equiv t :: k''$ are similar. □

Now we prove that the type of a term is preserved during the expansion.

Proposition 5.69 (Head subject expansion for $\lambda_{\circledR}^{\mathsf{Gtz}}$)

(i) *For every $\lambda_{®}^{\mathsf{Gtz}}$-term t: if $t \to t'$, t is contracted redex and $\Gamma \vdash t' : \sigma$, then $\Gamma \vdash t : \sigma$.*

(ii) *For every $\lambda_{®}^{\mathsf{Gtz}}$-context k: if $k \to k'$, k is contracted redex and $\Gamma;\alpha \vdash k' : \sigma$, then $\Gamma;\alpha \vdash k : \sigma$.*

Proof: The proof goes by the case study according to the applied reduction.

- Case (β). We should show that if $\Gamma \vdash u(\widehat{x}.tk) : \sigma$, then $\Gamma \vdash (\lambda x.t)(u :: k) : \sigma$. From $\Gamma \vdash u(\widehat{x}.tk) : \sigma$ by Generation lemma 5.46(*iii*) we have that $\Gamma = \Gamma', \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n$, that $\Delta_0 \vdash u : \tau_0, \Delta_1 \vdash u : \tau_1, ..., \Delta_n \vdash u : \tau_n$ $(\star)$ and that $\Gamma'; \cap_i^n \tau_i \vdash \widehat{x}.tk : \sigma$. Next, by Generation lemma 5.46(*iv*) it holds that $\Gamma', x : \cap_i^n \tau_i \vdash tk : \sigma$. Next, by another application of Generation lemma 5.46(*iii*) (knowing that $x \in Fv(t)$) we get that $\Gamma' = \Gamma'', \Delta_0'^{\top} \sqcap \Delta_1' \sqcap \ldots \sqcap \Delta_m'$ and $\Delta_0', x : \cap_i^n \tau_i \vdash t : \rho_0$, $\Delta_1', x : \cap_i^n \tau_i \vdash t : \rho_1$, ..., $\Delta_m', x : \cap_i^n \tau_i \vdash t : \rho_m$ $(\star\star)$ and finally $\Gamma''; \cap_j^m \rho_j \vdash \widehat{x}.k : \sigma$ $(\star\star\star)$.
  Now, from $(\star)$ and $(\star\star\star)$ applying $(\to_L)$ rule, we get
  $$\Gamma'', \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n; \cap_j^m (\cap_i^n \tau_i \to \rho_j) \vdash u :: k : \sigma \ (\bullet).$$
  On the other hand, applying $(\to_R)$ rule to each one of $(\star\star)$ we get
  $$\Delta_0' \vdash \lambda x.t : \cap_i^n \tau_i \to \rho_0,\ ,...,\ \Delta_m' \vdash \lambda x.t : \cap_i^n \tau_i \to \rho_m \ (\bullet\bullet).$$
  Finally, from $(\bullet)$ and $(\bullet\bullet)$ by the rule (Cut) we get
  $$\Gamma'', \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n, \Delta_0'^{\top} \sqcap \Delta_1' \sqcap \ldots \sqcap \Delta_m' \vdash (\lambda x.t)(u :: k) : \sigma$$
  which is exactly what we want since $\Gamma = \Gamma'', \Delta_0^{\top} \sqcap \Delta_1 \sqcap \ldots \sqcap \Delta_n, \Delta_0'^{\top} \sqcap \Delta_1' \sqcap \ldots \sqcap \Delta_m'$.

- Case (σ) is the direct consequence of Proposition 5.67.

- Case (π) is the direct consequence of Proposition 5.68.

- Case (μ) is easy, because we know that $x \notin Fv(k)$.

- Case (γ_1). We should show that the typeability of $t' \equiv \lambda y.x <_{x_2}^{x_1} u$ leads to the typeability of $t \equiv x <_{x_2}^{x_1} \lambda y.u$.
  Assume that $\Gamma \vdash \lambda y.x <_{x_2}^{x_1} u : \sigma$. By Generation lemma 5.46(*i*) we have that $\sigma \equiv \alpha \to \tau$ and $\Gamma, y : \alpha \vdash x <_{x_2}^{x_1} u : \tau$. Further, by Generation lemma 5.46(*v*) we get that $\Gamma, y : \alpha = \Gamma', y : \alpha, x : \beta \cap \gamma$ and $\Gamma', y : \alpha, x_1 : \beta, x_2 : \gamma \vdash u : \tau$. Now:
  $$\dfrac{\dfrac{\Gamma', y : \alpha, x_1 : \beta, x_2 : \gamma \vdash u : \tau}{\Gamma', x_1 : \beta, x_2 : \gamma \vdash \lambda y.u : \alpha \to \tau}\ (\to_R)}{\Gamma', x : \beta \cap \gamma \vdash x <_{x_2}^{x_1} \lambda y.u : \alpha \to \tau}\ (Cont).$$

This is what we want since $\Gamma = \Gamma', x : \beta \cap \gamma$ and $\sigma \equiv \alpha \to \tau$.

- The cases concerning other $(\gamma), (\omega)$ and $(\gamma\omega)$ reductions are similar, considering that weakening and contraction type assignment rules do not change the type of the consequence, and that Preservation of free variables (Proposition 5.12) and Domain correspondence (Proposition 5.45) hold in this system. □

Theorem 5.70 (SN ⇒ typeability) *All strongly normalising $\lambda_{®}^{\mathsf{Gtz}}$ expressions are typeable in the $\lambda_{®}^{\mathsf{Gtz}}\cap$ system.*

Proof: The proof is by induction over the length of the longest reduction path out of a strongly normalising expressions e, with a subinduction on the size of e.

- If e is a normal form, then e is typeable by Proposition 5.66.
- If e is itself a redex, let e' be the expression obtained by contracting the redex e. Then e' is also strongly normalising, hence by the IH it is typeable. Then e is typeable, by Proposition 5.69. Notice that, if $e \equiv (\lambda x.t)(u :: k) \to_{\beta} u(\widehat{x}.tk) \equiv e'$, then, by the IH, u is typeable, since the length of the longest reduction path out of u is smaller than that of e, and the size of u is smaller than the size of e.
- Next, suppose that e is not itself a redex nor a normal form. Then e is of one of the following forms: $\lambda x.u$, uk, $\lambda x.x \odot u$, $x \odot u$, or $x <^{x_1}_{x_2} uk$, $x_1 \in Fv(u)$, $x_2 \in Fv(k)$, $\widehat{x}..u$, $u :: k$, $\widehat{x}.x \odot u$, $x \odot k$, or $x <^{x_1}_{x_2} u :: k$, $x_1 \in Fv(u)$, $x_2 \in Fv(k)$ (where some subexpression is not the normal form). All subexpressions of e are typeable by IH. It is easy to build the typing for e in all these cases, having in mind that e is an expression (i.e. it is well-formed). □

Finally, we are able to give a characterisation of strong normalisation in the $\lambda_{®}^{\mathsf{Gtz}}$-calculus.

Theorem 5.71 *In the $\lambda_{®}^{\mathsf{Gtz}}$-calculus, the term t is strongly normalising if and only if it is typeable in the system $\lambda_{®}^{\mathsf{Gtz}}\cap$.*

Proof: Immediate consequence of Theorems 5.65 and 5.70. □

Chapter 6

The resource control cube

In this chapter we generalise the results proposed in the previous two chapters, by introducing the notion of *resource control cube*. This structure consists of two systems, namely $\lambda_{\mathcal{R}}$ and $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$, which can be seen as two opposite sides of a cube . Both systems consist of four calculi with either implicit or explicit treatment of the resource control. The difference between the two systems is in the corresponding proof systems of intuitionistic logic - the $\lambda_{\mathcal{R}}$ system corresponds to natural deduction with either implicit or explicit structural rules of weakening and contraction, while the $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$ system does the same job in the sequent calculus setting. Therefore, we call the $\lambda_{\mathcal{R}}$ system the *ND-base of the cube*, and the $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$ system the *LJ-base of the cube*. In the ND-base of the cube we consider four calculi, namely $\lambda_{\emptyset}$, λ_{c}, λ_{w} and λ_{cw}. The $\lambda_{\emptyset}$-calculus is actually regular λ-calculus, in which both resource operators are treated implicitly. The λ_{c}-calculus treats contraction (duplication) explicitly and weakening (erasure) implicitly; in the λ_{w}-calculus weakening is explicit whereas contraction is implicit and finally in the λ_{cw}-calculus both contraction and weakening are explicit. These four calculi very closely correspond to the implicit base of the Kesner and Renaud's Prismoid of Resources from [46], with certain differences in operational semantics caused by the substitution definition.

In the LJ-base of the cube we consider the four sequent style counterparts of the $\lambda_{\mathcal{R}}$-calculi, namely $\lambda_{\emptyset}^{\mathsf{Gtz}}$, $\lambda_{\mathsf{c}}^{\mathsf{Gtz}}$, $\lambda_{\mathsf{w}}^{\mathsf{Gtz}}$ and $\lambda_{\mathsf{cw}}^{\mathsf{Gtz}}$. The $\lambda_{\emptyset}^{\mathsf{Gtz}}$-calculus is actually the λ^{Gtz}-calculus, whereas the $\lambda_{\mathsf{cw}}^{\mathsf{Gtz}}$-calculus corresponds to the $\lambda_{®}^{\mathsf{Gtz}}$-calculus, presented in Chapter 5.

For all the calculi of the resource control cube, we use a notation along the lines of Žunić's work from [75] and close to van Oostrom's notation introduced in [72]. It is slightly modified w.r.t. [46] in order to emphasize the correspondence between these calculi and their sequent counterparts. Note that in [46] the main focus is on the treatment of substitution (thus the authors distinguish the *implicit* and the *explicit* base of the Prismoid of resources), whereas in the work proposed here we distinguish the *ND* and the *LJ* base of the Resource control cube, emphasizing differences in underlying logical proof systems.

The structure of this chapter is as follows. In Section 6.1 we present the untyped cube - its syntax and operational semantics. We start by defining the notions of pre-terms and free-variables, followed by the definition of terms. We move then to the operational semantics: reduction and equivalence rules and the definition of implicit substitution. Finally, by instantiating the set $\mathcal{R}$ we specify each of the eight calculi of the cube.

Section 6.2 revisits and summarizes basic type assignment systems with simple types, that are already known for the most of the calculi of the resource control cube (for all but the new calculi $\lambda_{\mathsf{c}}^{\mathsf{Gtz}}$ and $\lambda_{\mathsf{w}}^{\mathsf{Gtz}}$). Type systems of each base, $\lambda_{\mathcal{R}} \to$ and $\lambda_{\mathcal{R}}^{\mathsf{Gtz}} \to$, are presented in a uniform way.

In Section 6.3, we introduce intersection type assignment systems, $\lambda_{\mathcal{R}}\cap$ and $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}\cap$. These systems are based on the system $\lambda_{®}^{\mathsf{Gtz}}\cap$, introduced in Subsection 5.2.2, which is incorporated here as the system $\lambda_{\mathsf{cw}}^{\mathsf{Gtz}}\cap$. All systems are syntax-directed and assign strict types to resource control expressions. Some basic properties like Generation lemma, Substitution lemma, Append lemma[1], Subject reduction, Subject equivalence are claimed.

All sections of this chapter are divided into two subsections - one dealing with the ND-base of the resource control cube, and other dealing with the cube's sequent side.

This chapter represents original contribution of the thesis. It was developed by Pierre Lescanne, Silvia Ghilezan, Silvia Likavec and myself, and published in [34].

6.1 Type-free resource control cube

6.1.1 Resource control lambda calculi $\lambda_{\mathcal{R}}$

In this section, we present the syntax and the operational semantics of four calculi obtained by adding explicit contraction and/or weakening operator to the λ-calculus. We denote these four calculi by $\lambda_{\mathcal{R}}$, where $\mathcal{R} \subseteq \{\mathsf{c}, \mathsf{w}\}$ and c, w denote explicit operators of contraction and weakening, respectively. The operator which is not in the index of the calculi is assumed to be implicit in them. Thus, for instance λ_{w} denotes a calculus with explicit weakening and implicit contraction.

For the convenience of the reader and to avoid repetition, we present all the calculi in a uniform way. This implies that some constructions or features are part of one calculus and not of the others. When a feature occurs in a calculus

[1]only for the sequent systems

associated with the operator $\mathsf{r} \in \mathcal{R}$ and is ignored elsewhere, we put this feature between brackets indexed by the symbol r. For instance, if we write $[x \in Fv(f)]_{\mathsf{w}}$, this means that the condition $x \in Fv(f)$ appears only in the calculus which contains explicit weakening, as seen from the index w.

In order to define the terms of the four $\lambda_{\mathcal{R}}$-calculi, we first introduce the notion of *pre-terms*. A pre-term can be a variable, an abstraction, an application, a contraction or a weakening. The abstract syntax of the $\lambda_{\mathcal{R}}$ pre-terms is given by the following:

$$\text{Pre-terms} \qquad f \ ::= \ x \,|\, \lambda x.f \,|\, ff \,|\, x <^{x_1}_{x_2} f \,|\, x \odot f$$

where x ranges over a denumerable set of term variables.

Definition 6.1

(i) The list *of free variables of a pre-term f, denoted by $Fv[f]$, is defined as follows (list operators are given in Definition 5.1):*

$$\begin{array}{c} Fv[x] = x; \\ Fv[\lambda x.f] = Fv[f] \setminus x; \\ Fv[fg] = Fv[f], Fv[g]; \\ Fv[x \odot f] = x, Fv[f]; \end{array}$$

$$Fv[x <^{x_1}_{x_2} f] = \begin{cases} Fv[f], & \{x_1, x_2\} \cap Fv[f] = \emptyset \\ x, ((Fv[f] \setminus x_1) \setminus x_2), & \{x_1, x_2\} \cap Fv[f] \neq \emptyset \end{cases}$$

(ii) The set *of free variables of a pre-term f, denoted by $Fv(f)$, is extracted out of the list $Fv[f]$, by un-ordering the list and removing multiple occurrences of each variable, if any.*

(iii) The set *of bound variables of a pre-term f, denoted by $Bv(f)$, contains all variables that exist in f, but are not free in it.*

The only difference with respect to Definition 5.1 is in the contraction case. Here we distinguish two possibilities in the case of $x <^{x_1}_{x_2} f$ - the duplication binds the variables x_1 and x_2 in f and introduces a fresh variable x if at least one of x_1, x_2 is free in f; otherwise no new free variable is introduced. This difference plays an important role in the case of λ_{c} calculus, where due to the implicit weakening terms with void contractions exist, and it is important to forbid introduction of a “real" free variable obtained by contracting two “fake" variables. For example, in the term $x <^{x_1}_{x_2} x_1$ we want x to be free, while in $x <^{x_1}_{x_2} y$ we do not want that.

The sets of $\lambda_{\mathcal{R}}$-terms, denoted by $\Lambda_{\mathcal{R}}$, are subsets of the set of pre-terms and are defined by the inference rules given in Figure 6.1. In the reminder of this section, we will use $M, N, P, Q...$ to denote $\lambda_{\mathcal{R}}$-terms.

We also use the abbreviation $X \odot M$ for $x_1 \odot \ldots x_n \odot M$ and $X <^Y_Z M$ for $x_1 <^{y_1}_{z_1} \ldots x_n <^{y_n}_{z_n} M$, where X, Y and Z are lists of size n, consisting of all distinct variables $x_1,\ldots,x_n,y_1,\ldots,y_n,z_1,\ldots,z_n$. If $n = 0$, i.e., if X is an empty list, then $X \odot M = X <^Y_Z M = M$. Note that due to the equivalence relation defined in Figure 6.4, we can use these notations also for sets of variables of the same size.

Some inference rules that define $\lambda_{\mathcal{R}}$-terms (Figure 6.1) include conditions written in square parentheses, thus working in one way if some resource operator is explicit, and in another way if it is implicit. For example, if the weakening is explicit, i.e. $\mathsf{w} \in \mathcal{R}$, in the rules for building abstraction and contraction the bindings $\lambda x.f$ and $x <^{x_1}_{x_2} g$ can be constructed only if x is free in f and x_1 and x_2 are free in g. Similarly, in the application rule, two subterms must not share free variables only in the presence of explicit contraction, i.e. if $\mathsf{c} \in \mathcal{R}$. In that way, Figure 6.1 defines terms of all four calculi of *ND*-base of the cube.

$$\frac{}{x \in \Lambda_{\mathcal{R}}}$$

$$\frac{f \in \Lambda_{\mathcal{R}} \quad [x \in Fv(f)]_{\mathsf{w}}}{\lambda x.f \in \Lambda_{\mathcal{R}}} \qquad \frac{f \in \Lambda_{\mathcal{R}} \quad g \in \Lambda_{\mathcal{R}} \quad [Fv(f) \cap Fv(g) = \emptyset]_{\mathsf{c}}}{fg \in \Lambda_{\mathcal{R}}}$$

$$\frac{f \in \Lambda_{\mathcal{R}} \quad x \notin Fv(f)}{x \odot f \in \Lambda_{\mathcal{R}}} \ (\mathsf{w} \in \mathcal{R})$$

$$\frac{f \in \Lambda_{\mathcal{R}} \quad x_1 \neq x_2 \quad x \notin Fv(f) \setminus \{x_1,x_2\} \quad [x_1,x_2 \in Fv(f)]_{\mathsf{w}}}{x <^{x_1}_{x_2} f \in \Lambda_{\mathcal{R}}} \ (\mathsf{c} \in \mathcal{R})$$

Figure 6.1: $\lambda_{\mathcal{R}}$-terms

Example 6.2 *Pre-terms $\lambda x.y$ and $y <^{y_1}_{y_2} x$ are $\lambda_{\mathcal{R}}$-terms only if weakening is implicit (i.e. $\mathsf{w} \notin \mathcal{R}$). Similarly, pre-terms $\lambda x.xx$ and $x \odot \lambda y.yy$ are $\lambda_{\mathcal{R}}$-terms only if contraction is implicit (i.e. $\mathsf{c} \notin \mathcal{R}$).*

All the operations defined along this chapter are considered modulo alpha-conversion, the congruence generated by renaming of bound variables.

Using alpha-conversion, in what follows we consider Barendregt's convention [3] for variables: in the same context a variable cannot be both free and bound. This applies to binders like $\lambda x.M$ which binds x in M, $x <^{x_1}_{x_2} M$ which binds x_1 and

x_2 in M, and also to the implicit substitution $M[N/x]$ which can be seen as a binder for x in M.

Implicit substitution $M[N/x]$ is defined in Figure 6.2. In this definition, in case $\mathsf{c} \in \mathcal{R}$, the following condition must be satisfied:

$$Fv(M) \cap Fv(N) = \emptyset,$$

otherwise the substitution result would not be a (well-formed) $\lambda_{\mathcal{R}}$-term. In the same definition, terms N_1 and N_2 are obtained from N by renaming all free variables in N by distinct fresh variables, and $M[N_1/x_1, N_2/x_2]$ denotes parallel substitution. Note that the terms N_1 and N_2 do not have any free variables in common hence, it is not a problem to perform the substitution in parallel.

$$\begin{array}{rcl}
x[N/x] & \triangleq & N \\
y[N/x] & \triangleq & y,\ x \neq y \\
(\lambda y.M)[N/x] & \triangleq & \lambda y.M[N/x],\ x \neq y \\
(MP)[N/x] & \triangleq & M[N/x]P[N/x] \\
(y \odot M)[N/x] & \triangleq & \{y\} \setminus Fv(N) \odot M[N/x],\ x \neq y \\
(x \odot M)[N/x] & \triangleq & Fv(N) \setminus Fv(M) \odot M \\
(y <^{y_1}_{y_2} M)[N/x] & \triangleq & y <^{y_1}_{y_2} M[N/x],\ x \neq y \\
(x <^{x_1}_{x_2} M)[N/x] & \triangleq & Fv(N) <^{Fv(N_1)}_{Fv(N_2)} M[N_1/x_1, N_2/x_2]
\end{array}$$

Figure 6.2: Substitution in $\lambda_{\mathcal{R}}$-calculi

The operational semantics for the four calculi that form the "natural deduction base" of the resource control cube are given in Figures 6.3 and 6.4. Reduction rules of $\lambda_{\mathcal{R}}$-calculi are given in Figure 6.3, whereas equivalences are given in Figure 6.4. Reduction rules specific for each calculus are given in Figure 6.5.

As in the case of the $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus, reductions are divided into four groups:

- (β) reduction is the main computational step;
- (γ) reductions perform propagation of contraction into the term;
- (ω) reductions extract weakening out of the term;
- $(\gamma\omega)$ reductions explain the interaction between the two different resource operators.

$$
\begin{array}{lrcl}
(\beta) & (\lambda x.M)N & \to & M[N/x] \\
(\gamma_0) & x <^{x_1}_{x_2} y & \to & y \quad y \neq x_1, x_2 \\
(\gamma_0^{'}) & x <^{x_1}_{x_2} x_1 & \to & x \\
(\gamma_1) & x <^{x_1}_{x_2} (\lambda y.M) & \to & \lambda y.x <^{x_1}_{x_2} M \\
(\gamma_2) & x <^{x_1}_{x_2} (MN) & \to & (x <^{x_1}_{x_2} M)N, \text{ if } x_1, x_2 \notin Fv(N) \\
(\gamma_3) & x <^{x_1}_{x_2} (MN) & \to & M(x <^{x_1}_{x_2} N), \text{ if } x_1, x_2 \notin Fv(M) \\
(\omega_1) & \lambda x.(y \odot M) & \to & y \odot (\lambda x.M), \; x \neq y \\
(\omega_2) & (x \odot M)N & \to & \{x\} \setminus Fv(N) \odot (MN) \\
(\omega_3) & M(x \odot N) & \to & \{x\} \setminus Fv(M) \odot (MN) \\
(\gamma\omega_1) & x <^{x_1}_{x_2} (y \odot M) & \to & y \odot (x <^{x_1}_{x_2} M), \; y \neq x_1, x_2 \\
(\gamma\omega_2) & x <^{x_1}_{x_2} (x_1 \odot M) & \to & M[x/x_2]
\end{array}
$$

Figure 6.3: Reduction rules of $\lambda_{\mathcal{R}}$-calculi

$$
\begin{array}{lrcl}
(\varepsilon_1) & x \odot (y \odot M) & \equiv & y \odot (x \odot M) \\
(\varepsilon_2) & x <^{x_1}_{x_2} M & \equiv & x <^{x_2}_{x_1} M \\
(\varepsilon_3) & x <^{y}_{z} (y <^{u}_{v} M) & \equiv & x <^{y}_{u} (y <^{z}_{v} M) \\
(\varepsilon_4) & x <^{x_1}_{x_2} (y <^{y_1}_{y_2} M) & \equiv & y <^{y_1}_{y_2} (x <^{x_1}_{x_2} M), \; x \neq y_1, y_2, \; y \neq x_1, x_2
\end{array}
$$

Figure 6.4: Equivalences in $\lambda_{\mathcal{R}}$-calculi

$\lambda_{\mathcal{R}}$-calculi	reduction rules	equivalences
$\lambda_{\emptyset}$	β	
λ_{c}	$\beta,\ \gamma_0,\ \gamma_0^{'},\ \gamma_1, \gamma_2, \gamma_3$	$\varepsilon_2, \varepsilon_3, \varepsilon_4$
λ_{w}	$\beta,\ \omega_1, \omega_2, \omega_3$	ε_1
λ_{cw}	$\beta,\ \gamma_1, \gamma_2, \gamma_3,\ \omega_1, \omega_2, \omega_3,\ \gamma\omega_1,\ \gamma\omega_2$	$\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4$

Figure 6.5: ND base of the resource control cube

The group of (γ) reductions exists only in the two calculi that contain explicit contraction (i.e., if $\mathsf{c} \in \mathcal{R}$). Similarly, (ω) reductions belong only to the two calculi containing explicit weakening (i.e., if $\mathsf{w} \in \mathcal{R}$). Finally, the rules in $(\gamma\omega)$ group exist only if $\mathcal{R} = \{\mathsf{c}, \mathsf{w}\}$.

Particularly, the rules (γ_0) and $(\gamma_0^{'})$ exist only if $\mathcal{R} = \{\mathsf{c}\}$ and their role is to erase meaningless contractions. These two rules together correspond to the CGc rule in [46].

Notice the asymmetry between the reduction rules of the calculi λ_{c} and λ_{w}, namely in λ_{w} there is no counterpart of the (γ_0) and $(\gamma_0^{'})$ reductions of λ_{c}. The reason can be tracked back to the definition of $\lambda_{\mathcal{R}}$-terms in Figure 6.1, where the definition of the weakening operator $x \odot f$ does not depend on the presence of the explicit contraction. Alternatively, it would be possible to define weakening with the condition $[x \notin Fv(f)]_{\mathsf{c}}$ instead of unrestricted $x \notin Fv(f)$. In that case, terms like $x \odot x \odot M$ would exist, and the reduction $(\omega_0) : x \odot M \to M$, if $x \in Fv(M)$ would erase this redundant weakening. The typing rule for this weakening would require multiset treatment of the bases Γ, which is out of the scope of this thesis.

6.1.2 Resource control sequent lambda calculi $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$

In this subsection we present the syntax and the operational semantics of the four sequent calculi with explicit or implicit resource control, denoted by $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$, where $\mathcal{R} \subseteq \{\mathsf{c}, \mathsf{w}\}$ and c, w denote explicit contraction and weakening, respectively. These four calculi are sequent counterparts of the four resource control calculi $\lambda_{\mathcal{R}}$ presented above, and represent extensions of Espírito Santo's λ^{Gtz}-calculus.

The abstract syntax of the $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$ pre-expressions is the following:

$$\begin{array}{lrcl}
\text{Pre-terms} & f & ::= & x \,|\, \lambda x.f \,|\, fc \,|\, x <^{x_1}_{x_2} f \,|\, x \odot f \\
\text{Pre-contexts} & c & ::= & \widehat{x}.f \,|\, f :: c \,|\, x \odot c \,|\, x <^{x_1}_{x_2} c
\end{array}$$

where $x, x_1, \ldots, y, \ldots$ range over a denumerable set of term variables.

A pre-term can be a variable, an abstraction, an application (cut), a contraction or a weakening, whereas a pre-context is one of the following: a selection, a context constructor, a weakening on a pre-context or a contraction on a pre-context. Pre-terms and pre-contexts are together referred to as pre-expressions and will be ranged over by E. Pre-contexts $x \odot c$ and $x <^{x_1}_{x_2} c$ behave exactly like the corresponding pre-terms $x \odot f$ and $x <^{x_1}_{x_2} f$ in the untyped calculi, so they will mostly not be treated separately.

Definition 6.3

(i) The list *of free variables of a pre-expression E, denoted by $Fv[E]$, is defined as follows (list operators are defined as in Definition 5.1):*

$$\begin{array}{c}
Fv[x] = x; \\
Fv[\lambda x.f] = Fv[f] \setminus x; \\
Fv[fc] = Fv[f], Fv[c]; \\
Fv[\widehat{x}.f] = Fv[f] \setminus x; \\
Fv[f :: c] = Fv[f], Fv[c]; \\
Fv[x \odot E] = x, Fv[E]; \\
Fv[x <^{x_1}_{x_2} E] = \left\{ \begin{array}{rl} Fv[E], & \{x_1, x_2\} \cap Fv[E] = \emptyset \\ x, ((Fv[E] \setminus x_1) \setminus x_2), & \{x_1, x_2\} \cap Fv[E] \neq \emptyset \end{array} \right.
\end{array}$$

(ii) The set *of free variables of a pre-expression E, denoted by $Fv(E)$, is extracted out of the list $Fv[E]$, by un-ordering the list and removing multiple occurrences of each variable, if any.*

(iii) The set *of bound variables of a pre-expression E, denoted by $Bv(E)$, contains all the variables that exist in E, but are not free in it.*

The sets of $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-expressions $\Lambda^{\mathsf{Gtz}}_{\mathcal{R}} = \mathrm{T}^{\mathsf{Gtz}}_{\mathcal{R}} \cup \mathrm{K}^{\mathsf{Gtz}}_{\mathcal{R}}$ (where $\mathrm{T}^{\mathsf{Gtz}}_{\mathcal{R}}$ are the sets of $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-terms and $\mathrm{K}^{\mathsf{Gtz}}_{\mathcal{R}}$ are the sets of $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-contexts) are the subsets of the set of pre-expressions, defined by the inference rules given in Figure 6.6. We denote $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-terms by $t, u, v...$, $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-contexts by $k, k', ...$ and $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-expressions by e, e'.

Analogously to the definition of $\lambda_{\mathcal{R}}$-terms given in Figure 6.1, Figure 6.6 defines terms and contexts of all four calculi of the *LJ*-base of the cube. This is achieved by presenting some inference rules that define $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-expressions together with conditions depending on the explicit/implicit treatment of the resource operators. For example, if the weakening is explicit, i.e. $\mathsf{w} \in \mathcal{R}$, in the rules for building abstraction, selection and contraction the bindings $\lambda x.f$, $\widehat{y}.g$ and $x <^{x_1}_{x_2} h$ can be constructed only if x is free in f, y is free in g, and x_1 and x_2 are free in h. Similarly, in the application and cons rule, two subexpressions must not share free variables only in the presence of explicit contraction, i.e. if $\mathsf{c} \in \mathcal{R}$.

Example 6.4

$\lambda x.x(y :: \widehat{z}.z)$ belongs to all four sets of terms $\Lambda^{\mathsf{Gtz}}_{\emptyset}$, $\Lambda^{\mathsf{Gtz}}_{\mathsf{c}}$, $\Lambda^{\mathsf{Gtz}}_{\mathsf{w}}$ and $\Lambda^{\mathsf{Gtz}}_{\mathsf{cw}}$;

- *$\lambda x.w(y :: \widehat{z}.z)$ belongs only to $\Lambda^{\mathsf{Gtz}}_{\emptyset}$ and $\Lambda^{\mathsf{Gtz}}_{\mathsf{c}}$;*
- *$\lambda x.x(x :: \widehat{z}.z)$ belongs only to $\Lambda^{\mathsf{Gtz}}_{\emptyset}$ and $\Lambda^{\mathsf{Gtz}}_{\mathsf{w}}$;*
- *$\lambda x.y(y :: \widehat{z}.z)$ belongs only to $\Lambda^{\mathsf{Gtz}}_{\emptyset}$;*

$$\frac{}{x \in \mathsf{T}^{\mathsf{Gtz}}_{\mathcal{R}}}$$

$$\frac{f \in \mathsf{T}^{\mathsf{Gtz}}_{\mathcal{R}} \;\; [x \in Fv(f)]_{\mathsf{w}}}{\lambda x.f \in \mathsf{T}^{\mathsf{Gtz}}_{\mathcal{R}}} \qquad \frac{f \in \mathsf{T}^{\mathsf{Gtz}}_{\mathcal{R}} \;\; c \in \mathsf{K}^{\mathsf{Gtz}}_{\mathcal{R}} \;\; [Fv(f) \cap Fv(c) = \emptyset]_{\mathsf{c}}}{fc \in \mathsf{T}^{\mathsf{Gtz}}_{\mathcal{R}}}$$

$$\frac{f \in \mathsf{T}^{\mathsf{Gtz}}_{\mathcal{R}} \;\; [x \in Fv(f)]_{\mathsf{w}}}{\widehat{x}.f \in \mathsf{K}^{\mathsf{Gtz}}_{\mathcal{R}}} \qquad \frac{f \in \mathsf{T}^{\mathsf{Gtz}}_{\mathcal{R}} \;\; c \in \mathsf{K}^{\mathsf{Gtz}}_{\mathcal{R}} \;\; [Fv(f) \cap Fv(c) = \emptyset]_{\mathsf{c}}}{f :: c \in \mathsf{K}^{\mathsf{Gtz}}_{\mathcal{R}}}$$

$$\frac{f \in \mathsf{T}^{\mathsf{Gtz}}_{\mathcal{R}} \;\; x \notin Fv(f)}{x \odot f \in \mathsf{T}^{\mathsf{Gtz}}_{\mathcal{R}}} \;(\mathsf{w} \in \mathcal{R}) \qquad \frac{c \in \mathsf{K}^{\mathsf{Gtz}}_{\mathcal{R}} \;\; x \notin Fv(c)}{x \odot c \in \mathsf{K}^{\mathsf{Gtz}}_{\mathcal{R}}} \;(\mathsf{w} \in \mathcal{R})$$

$$\frac{f \in \mathsf{T}^{\mathsf{Gtz}}_{\mathcal{R}} \;\; x_1 \neq x_2 \;\; x \notin Fv(f) \setminus \{x_1, x_2\} \;\; [x_1, x_2 \in Fv(f)]_{\mathsf{w}}}{x <^{x_1}_{x_2} f \in \mathsf{T}^{\mathsf{Gtz}}_{\mathcal{R}}} \;(\mathsf{c} \in \mathcal{R})$$

$$\frac{c \in \mathsf{K}^{\mathsf{Gtz}}_{\mathcal{R}} \;\; x_1 \neq x_2 \;\; x \notin Fv(c) \setminus \{x_1, x_2\} \;\; [x_1, x_2 \in Fv(c)]_{\mathsf{w}}}{x <^{x_1}_{x_2} c \in \mathsf{K}^{\mathsf{Gtz}}_{\mathcal{R}}} \;(\mathsf{c} \in \mathcal{R})$$

Figure 6.6: $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-expressions

- $\lambda x.x \odot y(y :: \widehat{z}.z)$ *belongs only to* $\Lambda_{\mathsf{w}}^{\mathsf{Gtz}}$;
- $\lambda x.y <_{y_2}^{y_1} y_1(y_2 :: \widehat{z}.z)$ *belongs only to* $\Lambda_{\mathsf{c}}^{\mathsf{Gtz}}$;
- $\lambda x.x \odot y <_{y_2}^{y_1} y_1(y_2 :: \widehat{z}.z)$ *belongs only to* $\Lambda_{\mathsf{cw}}^{\mathsf{Gtz}}$.

The inductive definition of the meta operator of implicit substitution $e[t/x]$, representing the substitution of free variables, is given in Figure 6.7. In this definition, in case $\mathsf{c} \in \mathcal{R}$, the following condition must be satisfied:

$$Fv(e) \cap Fv(t) = \emptyset,$$

otherwise the substitution result would not be a (well-formed) $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$-expression. In the same definition, terms t_1 and t_2 are obtained from t by renaming all free variables in t by fresh distinct variables.

$$\begin{array}{rcl}
x[t/x] & \triangleq & t \\
y[t/x] & \triangleq & y \\
(\lambda y.v)[t/x] & \triangleq & \lambda y.v[t/x], \;\; x \neq y \\
(vk)[t/x] & \triangleq & v[t/x]\,k[t/x] \\
(v :: k)[t/x] & \triangleq & v[t/x] :: k[t/x] \\
(\widehat{y}.v)[t/x] & \triangleq & \widehat{y}.v[t/x] \\
(y \odot e)[t/x] & \triangleq & \{y\} \setminus Fv(t) \odot e[t/x], \;\; x \neq y \\
(x \odot e)[t/x] & \triangleq & Fv(t) \setminus Fv(e) \odot e \\
(y <_{y_2}^{y_1} e)[t/x] & \triangleq & y <_{y_2}^{y_1} e[t/x], \;\; x \neq y \\
(x <_{x_2}^{x_1} e)[t/x] & \triangleq & Fv(t) <_{Fv(t_2)}^{Fv(t_1)} e[t_1/x_1][t_2/x_2]
\end{array}$$

Figure 6.7: Substitution in $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$-calculi

The computation over the set of $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$-expressions reflects the cut-elimination process, and manages the explicit resource control operators. Four groups of reductions in $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$-calculi are given in Figure 6.8, whereas the equivalences are given in Figure 6.9. Reduction rules and equivalences specific to each of the four term calculi forming the *LJ* base of the resource control cube are given in Figure 6.10.

The first group consists of (β), (π), (σ) and (μ) reductions that exist in all four $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$-calculi. (β) reduction represents the main computational step. It creates a substitution, but it is the rule (σ) that executes it. In that sense, substitution can be controlled (i.e. delayed) although it is implicit. Note that this feature is not present

$$
\begin{array}{lrcl}
(\beta) & (\lambda x.t)(u :: k) & \rightarrow & u(\widehat{x}.tk) \\
(\sigma) & t(\widehat{x}.v) & \rightarrow & v[t/x] \\
(\pi) & (tk)k' & \rightarrow & t(k@k') \\
(\mu) & \widehat{x}.xk & \rightarrow & k \\
(\gamma_0) & x <^{x_1}_{x_2} y & \rightarrow & y \quad y \neq x_1, x_2 \\
(\gamma_0{}') & x <^{x_1}_{x_2} x_1 & \rightarrow & x \\
(\gamma_1) & x <^{x_1}_{x_2} (\lambda y.t) & \rightarrow & \lambda y.x <^{x_1}_{x_2} t \\
(\gamma_2) & x <^{x_1}_{x_2} (tk) & \rightarrow & (x <^{x_1}_{x_2} t)k,\ \text{if } x_1, x_2 \notin Fv(k) \\
(\gamma_3) & x <^{x_1}_{x_2} (tk) & \rightarrow & t(x <^{x_1}_{x_2} k),\ \text{if } x_1, x_2 \notin Fv(t) \\
(\gamma_4) & x <^{x_1}_{x_2} (\widehat{y}.t) & \rightarrow & \widehat{y}.(x <^{x_1}_{x_2} t) \\
(\gamma_5) & x <^{x_1}_{x_2} (t :: k) & \rightarrow & (x <^{x_1}_{x_2} t) :: k,\ \text{if } x_1, x_2 \notin Fv(k) \\
(\gamma_6) & x <^{x_1}_{x_2} (t :: k) & \rightarrow & t :: (x <^{x_1}_{x_2} k),\ \text{if } x_1, x_2 \notin Fv(t) \\
(\omega_1) & \lambda x.(y \odot t) & \rightarrow & y \odot (\lambda x.t),\ x \neq y \\
(\omega_2) & (x \odot t)k & \rightarrow & \{x\} \setminus Fv(k) \odot (tk) \\
(\omega_3) & t(x \odot k) & \rightarrow & \{x\} \setminus Fv(t) \odot (tk) \\
(\omega_4) & \widehat{x}.(y \odot t) & \rightarrow & y \odot (\widehat{x}.t),\ x \neq y \\
(\omega_5) & (x \odot t) :: k & \rightarrow & \{x\} \setminus Fv(k) \odot (t :: k) \\
(\omega_6) & t :: (x \odot k) & \rightarrow & \{x\} \setminus Fv(t) \odot (t :: k) \\
(\gamma\omega_1) & x <^{x_1}_{x_2} (y \odot e) & \rightarrow & y \odot (x <^{x_1}_{x_2} e) \qquad x_1 \neq y \neq x_2 \\
(\gamma\omega_2) & x <^{x_1}_{x_2} (x_1 \odot e) & \rightarrow & e[x/x_2]
\end{array}
$$

Figure 6.8: Reduction rules of $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-calculi

$$
\begin{array}{lrcl}
(\varepsilon_1) & x \odot (y \odot e) & \equiv & y \odot (x \odot e) \\
(\varepsilon_2) & x <^{x_1}_{x_2} e & \equiv & x <^{x_2}_{x_1} e \\
(\varepsilon_3) & x <^{y}_{z} (y <^{u}_{v} e) & \equiv & x <^{y}_{u} (y <^{z}_{v} e) \\
(\varepsilon_4) & x <^{x_1}_{x_2} (y <^{y_1}_{y_2} e) & \equiv & y <^{y_1}_{y_2} (x <^{x_1}_{x_2} e),\ x \neq y_1, y_2,\ y \neq x_1, x_2
\end{array}
$$

Figure 6.9: Equivalences in $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-calculi

$\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$-calculi	reduction rules	equivalences
$\lambda_{\emptyset}^{\mathsf{Gtz}}$	β, π, σ, μ	
$\lambda_{\mathsf{c}}^{\mathsf{Gtz}}$	β, π, σ, μ, γ_0 - γ_6	$\varepsilon_2, \varepsilon_3, \varepsilon_4$
$\lambda_{\mathsf{w}}^{\mathsf{Gtz}}$	β, π, σ, μ, ω_1 - ω_6	ε_1
$\lambda_{\mathsf{cw}}^{\mathsf{Gtz}}$	β, π, σ, μ, γ_1 - γ_6, ω_1 - ω_6, $\gamma\omega_1$, $\gamma\omega_2$	ε_1 - ε_4

Figure 6.10: *LJ* base of the resource control cube

in $\lambda_{\mathcal{R}}$-calculi since it is a consequence of the existence of contexts. Combination of $(\beta) + (\sigma)$ rules corresponds to the traditional (β) reduction in the λ-calculus. The rule (π) simplifies the head of a cut (t is the head of tk). The rule (μ) erases the sequence made of a trivial cut (a cut is trivial if its head is a variable) followed by the selection of the same variable. In that sense, it represents a kind of garbage collection.

In the (π) rule, the meta-operator @, called append, joins two contexts. The definition of the append operator completely corresponds to the definition of the same operator for the $\lambda_{®}^{\mathsf{Gtz}}$-calculus (Figure 5.3) but it is repeated here in Figure 6.11 for the sake of completeness.

$$\boxed{\begin{array}{lcl} (\widehat{x}.t)@k' & \triangleq & \widehat{x}.tk'; \\ (t :: k)@k' & \triangleq & t :: (k@k'); \\ (x \odot k)@k' & \triangleq & x \odot (k@k'); \\ (x <^{x_1}_{x_2} k)@k' & \triangleq & x <^{x_1}_{x_2} (k@k'). \end{array}}$$

Figure 6.11: Meta-operator @ in $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$-calculi

If $\mathsf{c} \in \mathcal{R}$, the group of (γ) reductions has the additional three reductions (comparing to (γ) reductions in $\lambda_{\mathcal{R}}$-calculi) which manage the interaction of contraction with selection and context construction. Also if $\mathsf{w} \in \mathcal{R}$, the group of (ω) reductions has additional three reductions which manage the interaction of weakening with selection and context construction. Finally, the group of $(\gamma\omega)$ reductions has additional two rules which manage the interaction between explicit resource operators of different nature in contexts.

We have presented the syntax and the reduction rules of the $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$, $\mathcal{R} \subseteq \{\mathsf{c}, \mathsf{w}\}$, the family of intuitionistic sequent term calculi. By instantiating $\mathcal{R}$, we obtain the following particular calculi:

- $\mathcal{R} = \emptyset$ gives us the well-known *lambda Gentzen calculus* λ^{Gtz}, proposed by Espírito Santo [27], whose simply typed version corresponds to the intuitionistic sequent calculus with the cut and implicit structural rules, according to the Curry-Howard correspondence.

- By letting $\mathcal{R} = \{\mathsf{c}, \mathsf{w}\}$, we obtain the *resource control lambda Gentzen calculus*, $\lambda_{®}^{\mathsf{Gtz}}$, whose call-by-value version was proposed and investigated in [35]. The simply typed $\lambda_{®}^{\mathsf{Gtz}}$ expands the Curry-Howard correspondence to the intuitionistic sequent calculus with the cut and explicit structural rules of weakening and contraction.
- Finally, in case $\mathcal{R} = \{\mathsf{c}\}$ and $\mathcal{R} = \{\mathsf{w}\}$ we get two new calculi, namely $\lambda_{\mathsf{c}}^{\mathsf{Gtz}}$ and $\lambda_{\mathsf{w}}^{\mathsf{Gtz}}$. These calculi could be related to the substructural logics, as will be elaborated in the sequel.

6.2 Simple types for resource control cube

In this section we summarize the type assignment systems that assign *simple types* to all eight calculi of the resource control cube. Simple types for λ^{Gtz}-calculus were introduced by Espírito Santo in [27]. As far as resource control calculi are concerned, simple types were introduced to the $\lambda\mathsf{lxr}$-calculus by Kesner and Lengrand in [45] and to resource control lambda Gentzen calculus $\lambda_{®}^{\mathsf{Gtz}}$ by Ghilezan et al. in [35]. Introductory notions, joint for all eight type assignment systems are the following.

Definition 6.5 *The syntax of simple types is defined as follows:*

$$\alpha \ ::= \ p \mid \alpha \to \alpha$$

where p ranges over a denumerable set of type atoms.

Types will be denoted by $\alpha, \beta, \gamma, \alpha_1, ...$ and the set of all simple types will be denoted by $\mathrm{T}_{\to}$.

Definition 6.6

(i) A basic type assignment is an expression of the form $x : \alpha$, where x is a term variable and α is a simple type.

(ii) A basis Γ is a set $\{x_1 : \alpha_1, \dots, x_n : \alpha_n\}$ of basic type assignments, where all term variables are different. $Dom(\Gamma) = \{x_1, \dots, x_n\}$.

(iii) A basis extension $\Gamma, x : \alpha$ denotes the set $\Gamma \cup \{x : \alpha\}$, where $x \notin Dom(\Gamma)$. Γ, Δ represents the disjoint union of the two bases.

(iv) $\Gamma \cup_{\mathsf{c}} \Delta$ denotes the standard union of the bases, if $\mathsf{c} \notin \mathcal{R}$, and the disjoint union, if $\mathsf{c} \in \mathcal{R}$.

6.2.1 Simply typed $\lambda_{\mathcal{R}}$-calculi

The type assignment systems $\lambda_{\mathcal{R}}\to$ for the *ND*-base of the resource control cube are given in Figure 6.12.

$$\frac{\mathsf{w} \notin \mathcal{R}}{\Gamma, x:\alpha \vdash_{\mathcal{R}} x:\alpha}\ (Ax_{iw}) \qquad \frac{\mathsf{w} \in \mathcal{R}}{x:\alpha \vdash_{\mathcal{R}} x:\alpha}\ (Ax_{ew})$$

$$\frac{\Gamma, x:\alpha \vdash_{\mathcal{R}} M:\beta}{\Gamma \vdash_{\mathcal{R}} \lambda x.M:\alpha\to\beta}\ (\to_I) \qquad \frac{\Gamma \vdash_{\mathcal{R}} M:\alpha\to\beta \quad \Delta \vdash_{\mathcal{R}} N:\beta}{\Gamma \cup_{\mathsf{c}} \Delta \vdash_{\mathcal{R}} MN:\beta}\ (\to_E)$$

$$\frac{\Gamma, x:\alpha, y:\alpha \vdash_{\mathcal{R}} M:\beta \quad \mathsf{c}\in\mathcal{R}}{\Gamma, z:\alpha \vdash_{\mathcal{R}} z <^x_y M:\beta}\ (Cont) \qquad \frac{\Gamma \vdash_{\mathcal{R}} M:\beta \quad \mathsf{w}\in\mathcal{R}}{\Gamma, x:\alpha \vdash_{\mathcal{R}} x\odot M:\beta}\ (Weak)$$

Figure 6.12: $\lambda_{\mathcal{R}}\to$: Simply typed $\lambda_{\mathcal{R}}$-calculi

Figure 6.12 includes four simple type assignment systems, each providing Curry-Howard correspondence with particular intuitionistic natural deduction system, with implicit/explicit structural rules of weakening and contraction. These systems are obtained by instantiating the set $\mathcal{R}$ in the turn-style symbol $\vdash_{\mathcal{R}}$.

- The system $\lambda_{\emptyset}\to$ consists of rules (Ax_{iw}), $(\to_I)$ and $(\to_E)$, where in $(\to_E)$ rule $\Gamma \cup_{\mathsf{c}} \Delta$ denotes standard bases union.
- The system $\lambda_{\mathsf{c}}\to$ consists of rules (Ax_{iw}), $(\to_I)$, $(\to_E)$ and $(Cont)$. In this case $\Gamma \cup_{\mathsf{c}} \Delta$ denotes disjoint bases union.
- The system $\lambda_{\mathsf{w}}\to$ consists of rules (Ax_{ew}), $(\to_I)$, $(\to_E)$ (with standard bases union) and $(Weak)$.
- Finally, the system $\lambda_{\mathsf{cw}}\to$ consists of rules (Ax_{ew}), $(\to_I)$, $(\to_E)$ (with disjoint bases union), $(Cont)$ and $(Weak)$.

All systems are syntax directed.

6.2.2 Simply typed $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-calculi

The type assignment systems $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}\to$ for the *LJ*-base of the resource control cube are given in Figure 6.13. In this case, we distinguish two sorts of type assignments:

- $\Gamma \vdash_{\mathcal{R}} t:\alpha$ for typing a $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-term and

- $\Gamma;\beta \vdash_{\mathcal{R}} k:\alpha$, a type assignment with a *stoup*, for typing a $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-context.

The role of the stoup is the same as in the case of other sequent term calculi presented in Chapter 4 and Chapter 5 of this thesis.

$$\frac{\mathsf{w}\notin\mathcal{R}}{\Gamma,x:\alpha\vdash_{\mathcal{R}} x:\alpha}\,(Ax_{iw}) \qquad \frac{\mathsf{w}\in\mathcal{R}}{x:\alpha\vdash_{\mathcal{R}} x:\alpha}\,(Ax_{ew})$$

$$\frac{\Gamma,x:\alpha\vdash_{\mathcal{R}} t:\beta}{\Gamma\vdash_{\mathcal{R}} \lambda x.t:\alpha\to\beta}\,(\to_R) \qquad \frac{\Gamma\vdash_{\mathcal{R}} t:\alpha \quad \Delta;\beta\vdash_{\mathcal{R}} k:\gamma}{\Gamma\cup_{\mathsf{c}}\Delta;\alpha\to\beta\vdash_{\mathcal{R}} t::k:\gamma}\,(\to_L)$$

$$\frac{\Gamma,x:\alpha\vdash_{\mathcal{R}} t:\beta}{\Gamma;\alpha\vdash_{\mathcal{R}} \widehat{x}.t:\beta}\,(Sel) \qquad \frac{\Gamma\vdash_{\mathcal{R}} t:\alpha \quad \Delta;\alpha\vdash_{\mathcal{R}} k:\beta}{\Gamma\cup_{\mathsf{c}}\Delta\vdash_{\mathcal{R}} tk:\beta}\,(Cut)$$

$$\frac{\Gamma,x:\alpha,y:\alpha\vdash_{\mathcal{R}} t:\beta \quad \mathsf{c}\in\mathcal{R}}{\Gamma,z:\alpha\vdash_{\mathcal{R}} z<^{x}_{y} t:\beta}\,(Cont_t) \qquad \frac{\Gamma\vdash_{\mathcal{R}} t:\beta \quad \mathsf{w}\in\mathcal{R}}{\Gamma,x:\alpha\vdash_{\mathcal{R}} x\odot t:\beta}\,(Weak_t)$$

$$\frac{\Gamma,x:\alpha,y:\alpha;B\vdash_{\mathcal{R}} k:\gamma \quad \mathsf{c}\in\mathcal{R}}{\Gamma,z:\alpha;\beta\vdash_{\mathcal{R}} z<^{x}_{y} k:\gamma}\,(Cont_k) \qquad \frac{\Gamma;\beta\vdash_{\mathcal{R}} k:\gamma \quad \mathsf{w}\in\mathcal{R}}{\Gamma,x:\alpha;\beta\vdash_{\mathcal{R}} x\odot k:\gamma}\,(Weak_k)$$

Figure 6.13: $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}\to$: Simply typed $\lambda^{\mathsf{Gtz}}_{\mathcal{R}}$-calculi

Particular simple type assignment systems for the four calculi of the *LJ* base of the resource control cube are the subsets of the system presented in Figure 6.13. The choice between two axioms depends on the treatment of weakening in the particular system, while reading of the bases union $\Gamma\cup_{\mathsf{c}}\Delta$ in the rules with two premises depends on the treatment of contraction. More precisely, the four syntax directed systems are the following:

- the system $\lambda^{\mathsf{Gtz}}_{\emptyset}\to$ consists of rules (Ax_{iw}), $(\to_R)$, $(\to_L)$, (Sel) and (Cut) ($\Gamma\cup_{\mathsf{c}}\Delta$ denotes the standard bases union in rules $(\to_L)$ and (Cut));
- the system $\lambda^{\mathsf{Gtz}}_{\mathsf{c}}\to$ consists of rules (Ax_{iw}), $(\to_R)$, $(\to_L)$, (Sel), (Cut), $(Cont_t)$ and $(Cont_k)$ (in this case $\Gamma\cup_{\mathsf{c}}\Delta$ denotes disjoint bases union);
- the system $\lambda^{\mathsf{Gtz}}_{\mathsf{w}}\to$ consists of rules (Ax_{ew}), $(\to_R)$, $(\to_L)$, (Sel), (Cut), $(Weak_t)$ and $(Weak_k)$ (bases union is standard);
- finally, the system $\lambda^{\mathsf{Gtz}}_{\mathsf{cw}}\to$ consists of rules (Ax_{ew}), $(\to_R)$, $(\to_L)$, (Sel), (Cut), $(Cont_t)$, $(Cont_k)$, $(Weak_t)$ and $(Weak_k)$ (bases union is disjoint).

Simply typed $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$-calculi correspond to intuitionistic sequent calculus with implicit/explicit structural rules *à la* Curry-Howard. Particularly, $\lambda_{\emptyset}^{\mathsf{Gtz}}\!\!\to$ and $\lambda_{\mathsf{cw}}^{\mathsf{Gtz}}\!\!\to$ calculi correspond to the intuitionistic implicative fragments of Kleene's systems G3 and G1 from [48], respectively, except for the fact that the exchange rule is implicit here. The exchange rule could be made explicit by considering the bases as lists instead of sets. The system $\lambda_{\mathsf{c}}^{\mathsf{Gtz}}\!\!\to$ corresponds to the intuitionistic sequent calculus with explicit contraction and implicit weakening, whereas the system $\lambda_{\mathsf{w}}^{\mathsf{Gtz}}\!\!\to$ corresponds to the intuitionistic sequent calculus with explicit weakening and implicit contraction.

Modifications of $\lambda_{\mathcal{R}}\!\!\to$ and $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}\!\!\to$ systems can provide the computational interpretation of substructural logics, different from the usual approach via linear logic. For instance, if one combines (Ax_{ew}) axiom and the other rules in $\mathsf{w} \notin \mathcal{R}$ modality, the resulting system would correspond to the logic without weakening, i.e. to the variant of implicative fragment of relevance logic. In turn, if we use $\cup_{\mathsf{c}}$ as a disjoint union together with the $\mathsf{c} \notin \mathcal{R}$ modality of the rest of the system, the correspondence with the variant of the logic without contraction, i.e. the implicative fragment of affine logic is obtained. These systems, although very interesting from both theoretical and practical point of view, will not be investigated in this thesis.

Although the systems $\lambda_{\mathcal{R}}\!\!\to$ and $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}\!\!\to$ enjoy the subject reduction and the strong normalisation, they as expected do not assign types to all strongly normalising expressions. For example, the normal form $\lambda x.x <^{x_1}_{x_2} x_1x_2$ is not typeable in the system $\lambda_{\mathcal{R}}\!\!\to$. This is the motivation for introducing intersection types to the resource control cube in the next section.

6.3 Intersection types for resource control cube

In this section we introduce intersection type assignment systems which assign strict types, a specific subset of intersection types, to $\lambda_{\mathcal{R}}$-terms and $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$-expressions. The proposed type system scheme integrates the strict type systems already used in [25] and [32] for the characterisation of strong normalisation in three calculi of the cube, namely the λ^{Gtz}-calculus, the $\lambda_{®}$-calculus and the $\lambda_{®}^{\mathsf{Gtz}}$-calculus. Therefore, it represents a generalisation of the results presented in Chapter 4 and Chapter 5 of this thesis.

The syntax of types is defined as follows:

$$\begin{array}{llll} \text{Strict Types} & \sigma & ::= & p \mid \alpha \to \sigma \\ \text{Types} & \alpha & ::= & \cap_i^n \sigma_i \end{array}$$

where p ranges over a denumerable set of type atoms and $\cap_i^n \sigma_i = \sigma_1 \cap \ldots \cap \sigma_n$, $n \geq 0$. Particularly, $\cap_i^0 \sigma_i$, abbreviated by $\top$, represents the neutral element for the intersection.

We denote types by α, β, γ..., strict types by $\sigma, \tau, \rho, \upsilon$... and the set of all types by Types. We assume that the intersection operator is idempotent, commutative and associative, and that it has priority over the arrow operator. Hence, we will omit parenthesis in expressions like $(\cap_i^n \tau_i) \to \sigma$.

The definitions of a basic type assignment, a basis, a domain of the basis, a basis extension and the basis $\Gamma^\top = \{x : \top \mid x \in Dom(\Gamma)\}$ are the same as in Definition 6.6, except for the fact that the type assignments are of the form

$$x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash M : \sigma$$

so that only strict types are assigned to terms.

The following operator on bases, denoted by $\sqcup_{\mathsf{c}}$, is specific for intersection type assignment systems, and defined dependently on the explicit/implicit control of the contraction in the particular calculi of the cube.

Definition 6.7

(i) A union of bases with intersection types is defined in the standard way:

$$\begin{array}{rcl} \Gamma \sqcup \Delta & = & \{x : \alpha \mid x : \alpha \in \Gamma \;\&\; x \notin Dom(\Delta)\} \\ & \cup & \{x : \alpha \mid x : \alpha \in \Delta \;\&\; x \notin Dom(\Gamma)\} \\ & \cup & \{x : \alpha \cap \beta \mid x : \alpha \in \Gamma \;\&\; x : \beta \in \Delta\}. \end{array}$$

(ii) $\Gamma \sqcup_{\mathsf{c}} \Delta$ is $\Gamma \sqcup \Delta$, if $\mathsf{c} \notin \mathcal{R}$, otherwise it is the disjoint union Γ, Δ.

Notice that in this case, contrary to the definition of bases intersection $\Gamma \sqcap \Delta$ (Definition 5.43), it is not required that the domains of bases Γ and Δ coincide in order to perform bases union $\Gamma \sqcup \Delta$. However, the operator $\sqcap$ can be considered as a restriction of the operator $\sqcup$ in the case of equal domains:

$$\Gamma \sqcap \Delta = \Gamma \sqcup \Delta|_{Dom(\Gamma) = Dom(\Delta)}.$$

As a consequence, the basis $\Gamma^\top$ is not the neutral element for the $\sqcup$ operator in general, but it is irrelevant since it only appears in those systems in which $\mathsf{w} \in \mathcal{R}$, and in those systems the equality of domains holds.

6.3.1 Intersection types for $\lambda_{\mathcal{R}}$

The type assignment systems $\lambda_{\mathcal{R}}\cap$ for the natural deduction base of the resource control cube are given in Figure 6.14. The rules that correspond to each of the four $\lambda_{\mathcal{R}}\cap$-systems are specified in Figure 6.15.

All four systems are syntax-directed, i.e. the intersection operator is incorporated into already existing rules of the simply-typed systems, thus one of the most important nice properties of the systems from Chapters 4 and 5 is preserved in this generalised setting.

$$\frac{\mathsf{w} \notin \mathcal{R}}{\Gamma, x:\sigma \vdash_{\mathcal{R}} x:\sigma}\ (Ax_{iw}) \qquad \frac{\mathsf{w} \in \mathcal{R}}{x:\sigma \vdash_{\mathcal{R}} x:\sigma}\ (Ax_{ew})$$

$$\frac{\Gamma, x:\alpha \vdash_{\mathcal{R}} M:\sigma}{\Gamma \vdash_{\mathcal{R}} \lambda x.M : \alpha \to \sigma}\ (\to_I)$$

$$\frac{\Gamma \vdash_{\mathcal{R}} M : \cap_i^n \tau_i \to \sigma \quad [\Delta_0 \vdash_{\mathcal{R}} N:\tau_0]_{\mathsf{w}} \quad \Delta_1 \vdash_{\mathcal{R}} N:\tau_1 \quad \ldots \quad \Delta_n \vdash_{\mathcal{R}} N:\tau_n}{\Gamma \sqcup_{\mathsf{c}} ([\Delta_0^{\top} \sqcup]_{\mathsf{w}} \Delta_1 \sqcup \ldots \sqcup \Delta_n) \vdash_{\mathcal{R}} MN:\sigma}\ (\to_E)$$

$$\frac{\Gamma, x:\alpha, y:\beta \vdash_{\mathcal{R}} M:\sigma \quad \mathsf{c} \in \mathcal{R}}{\Gamma, z:\alpha\cap\beta \vdash_{\mathcal{R}} z <_y^x M:\sigma}\ (Cont) \qquad \frac{\Gamma \vdash_{\mathcal{R}} M:\sigma \quad \mathsf{w} \in \mathcal{R}}{\Gamma, x:\top \vdash_{\mathcal{R}} x \odot M:\sigma}\ (Weak)$$

Figure 6.14: $\lambda_{\mathcal{R}}\cap$: $\lambda_{\mathcal{R}}$-calculi with intersection types

$\lambda_{\mathcal{R}}\cap$-systems	type assignment rules
$\lambda_{\emptyset}\cap$	(Ax_{iw}), $(\to_I)$, $(\to_E)$
$\lambda_{\mathsf{c}}\cap$	(Ax_{iw}), $(\to_I)$, $(\to_E)$, $(Cont)$
$\lambda_{\mathsf{w}}\cap$	(Ax_{ew}), $(\to_I)$, $(\to_E)$, $(Weak)$
$\lambda_{\mathsf{cw}}\cap$	(Ax_{ew}), $(\to_I)$, $(\to_E)$, $(Cont)$, $(Weak)$

Figure 6.15: Four ND intersection type systems

The role of the assignment $\Delta_0 \vdash_{\mathcal{R}} N:\sigma_0$ in the rule $(\to_E)$ is to ensure that N has a type in case that $n = 0$. In that case $\Gamma \vdash_{\mathcal{R}} M : \top \to \sigma$, so it appears only in the systems with explicit control of weakening, as indicated by $[\ ]_{\mathsf{w}}$. Since the type $\top$ denotes a "useless" variable (the one introduced by weakening) free variables of N will also become useless (it will be obvious after β-reduction). In order to satisfy type preservation during computation, their types are also turned to $\top$ in the basis

$\Delta_0^\top$. If $n \neq 0$, then $\Delta_0 \vdash_{\mathcal{R}} N : \sigma_0$ can be any of the already existing assignments $\Delta_i \vdash_{\mathcal{R}} N : \sigma_i$. Moreover, due to the assumed idempotency of intersection, it can be even forgotten.

Example 6.8 *Consider the term $(\lambda x.x \odot y)z$ in the $\lambda_{\mathsf{cw}}\cap$ system* [2]*. This term reduces in the following way:*

$$(\lambda x.x \odot y)z \;\to_\beta\; (x \odot y)[z/x] \;\triangleq\; z \odot y.$$

The normal form $z \odot y$ is typeable in $\lambda_{\mathsf{cw}}\cap$ by $z : \top, y : \sigma \vdash_{\mathcal{R}} z \odot y : \sigma$. The same type assignment holds for the term $(\lambda x.x \odot y)z$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y:\sigma \vdash_{\mathsf{cw}} y:\sigma}\,(Ax_{ew})}{x:\top, y:\sigma \vdash_{\mathsf{cw}} x \odot y : \sigma}\,(Weak)}{y:\sigma \vdash_{\mathsf{cw}} \lambda x.x \odot y : \top \to \sigma}\,(\to_I) \qquad \dfrac{}{z:\rho \vdash_{\mathsf{cw}} z:\rho}\,(Ax_{ew})}{y:\sigma, z:\top \vdash_{\mathsf{cw}} (\lambda x.x \odot y)z : \sigma}\,(\to_E).$$

In the previous derivation, $N \equiv z$, $\Delta_0 = \{z : \rho\}$ and $\Delta_0^\top = \{z : \top\}$.

6.3.2 Intersection types for $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$

The type assignment systems $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}\cap$ for the sequent LJ-base of the resource control cube are given in Figure 6.16. The rules that correspond to each of the four $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}\cap$-systems are given in Figure 6.17.

As in $\lambda_{\mathcal{R}}\cap$, no new rules are added compared to $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}\!\to$ in order to manage intersection. Specificity of the rules $(\to_L)$ and (Cut) comparing them to their instances in the system $\lambda_{\circledR}^{\mathsf{Gtz}}\cap$ (Figure 5.8) is only in the fact that in the case when $\mathsf{w} \notin \mathcal{R}$ domains of the bases Γ_i do not necessarily coincide. The role of an additional assignment $\Gamma_0 \vdash_{\mathcal{R}} t : \sigma_0$ in these two rules, in the presence of explicit control of weakening is already explained in the previous subsection.

Since the role of this chapter was only to show the joint underlying ground of the calculi presented in the previous chapters and some possibilities of their joint foundation, we will not go further into details and claim their properties in this framework. It is possible to prove in the modular way that both *ND* and *LJ* bases of the resource cube, equipped with simple types, satisfy preservation of types during computation (Subject reduction and Subject equivalence) and that all typeable terms are terminating (Strong normalisation). For the systems with intersection types, it is possible to prove even a stronger property, namely that terms

[2]The other possibility is to consider it in the system $\lambda_{\mathsf{w}}\cap$

$$\frac{\mathsf{w} \notin \mathcal{R}}{\Gamma, x : \sigma \vdash_{\mathcal{R}} x : \sigma}\ (Ax_{iw}) \qquad \frac{\mathsf{w} \in \mathcal{R}}{x : \sigma \vdash_{\mathcal{R}} x : \sigma}\ (Ax_{ew})$$

$$\frac{\Gamma, x : \alpha \vdash_{\mathcal{R}} t : \sigma}{\Gamma \vdash_{\mathcal{R}} \lambda x.t : \alpha \to \sigma}\ (\to_R) \qquad \frac{\Gamma, x : \alpha \vdash_{\mathcal{R}} t : \sigma}{\Gamma; \alpha \vdash_{\mathcal{R}} \widehat{x}.t : \sigma}\ (Sel)$$

$$\frac{[\Gamma_0 \vdash_{\mathcal{R}} t : \sigma_0]_{\mathsf{w}} \quad \Gamma_1 \vdash_{\mathcal{R}} t : \sigma_1 \quad \ldots \quad \Gamma_n \vdash_{\mathcal{R}} t : \sigma_n \quad \Delta; \cap_j^m \tau_j \vdash_{\mathcal{R}} k : \rho}{([\Gamma_0^{\top} \sqcup]_{\mathsf{w}} \Gamma_1 \sqcup \ldots \sqcup \Gamma_n) \sqcup_{\mathsf{c}} \Delta; \cap_j^m (\cap_i^n \sigma_i \to \tau_j) \vdash_{\mathcal{R}} t :: k : \rho}\ (\to_L)$$

$$\frac{[\Gamma_0 \vdash_{\mathcal{R}} t : \sigma_0]]_{\mathsf{w}} \quad \Gamma_1 \vdash_{\mathcal{R}} t : \sigma_1 \quad \ldots \quad \Gamma_n \vdash_{\mathcal{R}} t : \sigma_n \quad \Delta; \cap_i^n \sigma_i \vdash_{\mathcal{R}} k : \tau}{([\Gamma_0^{\top} \sqcup]_{\mathsf{w}} \Gamma_1 \sqcup \ldots \sqcup \Gamma_n) \sqcup_{\mathsf{c}} \Delta \vdash_{\mathcal{R}} tk : \tau}\ (Cut)$$

$$\frac{\Gamma, x : \alpha, y : \beta \vdash_{\mathcal{R}} t : \sigma \quad \mathsf{c} \in \mathcal{R}}{\Gamma, z : \alpha \cap \beta \vdash_{\mathcal{R}} z <_y^x t : \sigma}\ (Cont_t) \qquad \frac{\Gamma \vdash_{\mathcal{R}} t : \sigma \quad \mathsf{w} \in \mathcal{R}}{\Gamma, x : \top \vdash_{\mathcal{R}} x \odot t : \sigma}\ (Weak_t)$$

$$\frac{\Gamma, x : \alpha, y : \beta; \gamma \vdash_{\mathcal{R}} k : \sigma \quad \mathsf{c} \in \mathcal{R}}{\Gamma, z : \alpha \cap \beta; \gamma \vdash_{\mathcal{R}} z <_y^x k : \sigma}\ (Cont_k) \qquad \frac{\Gamma; \gamma \vdash_{\mathcal{R}} k : \sigma \quad \mathsf{w} \in \mathcal{R}}{\Gamma, x : \top; \gamma \vdash_{\mathcal{R}} x \odot k : \sigma}\ (Weak_k)$$

Figure 6.16: $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}\cap$: $\lambda_{\mathcal{R}}^{\mathsf{Gtz}}$-calculi with intersection types

$\lambda_{\mathcal{R}}\cap$-systems	type assignment rules
$\lambda_{\emptyset}\cap$	(Ax_{iw}), $(\to_R)$, $(\to_L)$, (Sel), (Cut)
$\lambda_{\mathsf{c}}\cap$	(Ax_{iw}), $(\to_R)$, $(\to_L)$, (Sel), (Cut), $(Cont_t)$, $(Cont_k)$
$\lambda_{\mathsf{w}}\cap$	(Ax_{ew}), $(\to_R)$, $(\to_L)$, (Sel), (Cut), $(Weak_t)$, $(Weak_k)$
$\lambda_{\mathsf{cw}}\cap$	(Ax_{ew}), $(\to_R)$, $(\to_L)$, (Sel), (Cut), $(Cont_t)$, $(Cont_k)$, $(Weak_t)$, $(Weak_k)$

Figure 6.17: Four LJ intersection type systems

are terminating if and only if they are typeable with intersection types. However, the proofs are quite technical because of the uniform approach to both explicit and implicit treatment of the resource control operators, so one may ask whether it pays off to investigate the structure together as a whole.

We conclude by giving an example of type assignment of an operationally equivalent expression in the different calculi of the resource control cube.

Example 6.9 *The well-known example of the term typeable only with intersection types is $\lambda x.xx$. In the resource control cube, this term exists only in the two calculi of the ND-base in which the contraction is implicit, while in the other six calculi, we can find operationally equivalent expressions:*

- *$\lambda x.xx$ belongs to $\lambda_{\emptyset}$ and λ_{w};*
- *$\lambda x.x <^{x_1}_{x_2} x_1x_2$ belongs to λ_{c} and λ_{cw};*
- *$\lambda x.x(x :: \widehat{y}.y)$ belongs to $\lambda^{\mathsf{Gtz}}_{\emptyset}$ and $\lambda^{\mathsf{Gtz}}_{\mathsf{w}}$;*
- *$\lambda x.x <^{x_1}_{x_2} x_1(x_2 :: \widehat{y}.y)$ belongs to $\lambda^{\mathsf{Gtz}}_{\mathsf{c}}$ and $\lambda^{\mathsf{Gtz}}_{\mathsf{cw}}$.*

All these expressions receive the same strict type $(\sigma\to\tau)\cap\sigma\to\tau$ in all eight calculi of the cube.

- *In the $\lambda_{\emptyset}\cap$ system:*

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x:\sigma\to\tau\vdash_{\emptyset} x:\sigma\to\tau}\,(Ax_{iw})\qquad \dfrac{}{\Delta, x:\sigma\vdash_{\emptyset} x:\sigma}\,(Ax_{iw})}{\Gamma\sqcup\Delta\sqcup x:(\sigma\to\tau)\cap\sigma\vdash_{\emptyset} xx:\tau}\,(\to_E)}{\Gamma\sqcup\Delta\vdash_{\emptyset}\lambda x.xx:(\sigma\to\tau)\cap\sigma\to\tau}\,(\to_I).$$

- *In the $\lambda_{\mathsf{w}}\cap$ system:*

$$\dfrac{\dfrac{\dfrac{}{x:\sigma\to\tau\vdash_{\mathsf{w}} x:\sigma\to\tau}\,(Ax_{ew})\qquad \dfrac{}{x:\sigma\vdash_{\mathsf{w}} x:\sigma}\,(Ax_{ew})}{x:(\sigma\to\tau)\cap\sigma\vdash_{\mathsf{w}} xx:\tau}\,(\to_E)}{\vdash_{\mathsf{w}}\lambda x.xx:(\sigma\to\tau)\cap\sigma\to\tau}\,(\to_I).$$

- *In the $\lambda_{\mathsf{c}}\cap$ system:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, x_1:\sigma\to\tau\vdash_{\mathsf{c}} x_1:\sigma\to\tau}\,(Ax_{iw})\qquad \dfrac{}{\Delta, x_2:\sigma\vdash_{\mathsf{c}} x_2:\sigma}\,(Ax_{iw})}{\Gamma,\Delta, x_1:\sigma\to\tau, x_2:\sigma\vdash_{\mathsf{c}} x_1x_2:\tau}\,(\to_E)}{\Gamma,\Delta, x:(\sigma\to\tau)\cap\sigma\vdash_{\mathsf{c}} x <^{x_1}_{x_2} x_1x_2:\tau}\,(Cont)}{\Gamma,\Delta\vdash_{\mathsf{c}}\lambda x.x <^{x_1}_{x_2} x_1x_2:(\sigma\to\tau)\cap\sigma\to\tau}\,(\to_I).$$

- *In the* $\lambda_{\mathsf{cw}}\cap$ *system:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x_1:\sigma\to\tau\vdash_{\mathsf{cw}} x_1:\sigma\to\tau}\,(Ax_{ew}) \qquad \dfrac{}{x_2:\sigma\vdash_{\mathsf{cw}} x_2:\sigma}\,(Ax_{ew})}{x_1:\sigma\to\tau, x_2:\sigma\vdash_{\mathsf{cw}} x_1x_2:\tau}\,(\to_E)}{x:(\sigma\to\tau)\cap\sigma\vdash_{\mathsf{cw}} x<^{x_1}_{x_2} x_1x_2:\tau}\,(Cont)}{\vdash_{\mathsf{cw}}\lambda x.x<^{x_1}_{x_2}x_1x_2:(\sigma\to\tau)\cap\sigma\to\tau}\,(\to_I).$$

- *In the* $\lambda_{\emptyset}^{\mathsf{Gtz}}\cap$ *system:*

$$\dfrac{\dfrac{\dfrac{}{\Gamma_1,x:\sigma\to\tau\vdash_{\emptyset} x:\sigma\to\tau}\,(Ax_{iw}) \qquad \dfrac{\dfrac{}{\Gamma_2,x:\sigma\vdash_{\emptyset} x:\sigma}\,(Ax_{iw}) \quad \dfrac{\dfrac{}{\Gamma_3,y:\tau\vdash_{\emptyset} y:\tau}\,(Ax_{iw})}{\Gamma_3;\tau\vdash_{\emptyset}\widehat{y}.y:\tau}\,(Sel)}{\Gamma_2\sqcup\Gamma_3,x:\sigma;\sigma\to\tau\vdash_{\emptyset} x::\widehat{y}.y:\tau}\,(\to_L)}{\Gamma_1\sqcup\Gamma_2\sqcup\Gamma_3,x:(\sigma\to\tau)\cap\sigma\vdash_{\emptyset} x(x::\widehat{y}.y):\tau}\,(Cut)}{\Gamma_1\sqcup\Gamma_2\sqcup\Gamma_3\vdash_{\emptyset}\lambda x.x(x::\widehat{y}.y):(\sigma\to\tau)\cap\sigma\to\tau}\,(\to_R).$$

- *In the* $\lambda_{\mathsf{w}}^{\mathsf{Gtz}}\cap$ *system:*

$$\dfrac{\dfrac{\dfrac{}{x:\sigma\to\tau\vdash_{\mathsf{w}} x:\sigma\to\tau}\,(Ax_{ew}) \qquad \dfrac{\dfrac{}{x:\sigma\vdash_{\mathsf{w}} x:\sigma}\,(Ax_{ew}) \quad \dfrac{\dfrac{}{y:\tau\vdash_{\mathsf{w}} y:\tau}\,(Ax_{ew})}{;\tau\vdash_{\mathsf{w}}\widehat{y}.y:\tau}\,(Sel)}{x:\sigma;\sigma\to\tau\vdash_{\mathsf{w}} x::\widehat{y}.y:\tau}\,(\to_L)}{x:(\sigma\to\tau)\cap\sigma\vdash_{\mathsf{w}} x(x::\widehat{y}.y):\tau}\,(Cut)}{\vdash_{\mathsf{w}}\lambda x.x(x::\widehat{y}.y):(\sigma\to\tau)\cap\sigma\to\tau}\,(\to_R).$$

- *In the* $\lambda_{\mathsf{c}}^{\mathsf{Gtz}}\cap$ *system:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_1,x_1:\sigma\to\tau\vdash_{\mathsf{c}} x_1:\sigma\to\tau}\,(Ax_{iw}) \qquad \dfrac{\dfrac{}{\Gamma_2,x_2:\sigma\vdash_{\mathsf{c}} x_2:\sigma}\,(Ax_{iw}) \quad \dfrac{\dfrac{}{\Gamma_3,y:\tau\vdash_{\mathsf{c}} y:\tau}\,(Ax_{iw})}{\Gamma_3;\tau\vdash_{\mathsf{c}}\widehat{y}.y:\tau}\,(Sel)}{\Gamma_2,\Gamma_3,x_2:\sigma;\sigma\to\tau\vdash_{\mathsf{c}} x_2::\widehat{y}.y:\tau}\,(\to_L)}{\Gamma_1,\Gamma_2,\Gamma_3,x_1:\sigma\to\tau,x_2:\sigma\vdash_{\mathsf{c}} x_1(x_2::\widehat{y}.y):\tau}\,(Cut)}{\Gamma_1,\Gamma_2,\Gamma_3,x:(\sigma\to\tau)\cap\sigma\vdash_{\mathsf{c}} x<^{x_1}_{x_2}x_1(x_2::\widehat{y}.y):\tau}\,(Cont)}{\Gamma_1,\Gamma_2,\Gamma_3\vdash_{\mathsf{c}}\lambda x.x<^{x_1}_{x_2}x_1(x_2::\widehat{y}.y):(\sigma\to\tau)\cap\sigma\to\tau}\,(\to_R).$$

- *In the* $\lambda_{\mathsf{cw}}^{\mathsf{Gtz}}\cap$ *system:*

$$\dfrac{\dfrac{\dfrac{}{x_1 : \sigma \to \tau \vdash_{\mathsf{cw}} x_1 : \sigma \to \tau}\,(Ax_{ew}) \qquad \dfrac{\dfrac{}{x_2 : \sigma \vdash_{\mathsf{cw}} x_2 : \sigma}\,(Ax_{ew}) \quad \dfrac{\dfrac{}{y : \tau \vdash_{\mathsf{cw}} y : \tau}\,(Ax_{ew})}{;\tau \vdash_{\mathsf{cw}} \widehat{y}.y : \tau}\,(Sel)}{x_2 : \sigma ; \sigma \to \tau \vdash_{\mathsf{cw}} x_2 :: \widehat{y}.y : \tau}\,(\to_L)}{\dfrac{x_1 : \sigma \to \tau, x_2 : \sigma \vdash_{\mathsf{cw}} x_1(x_2 :: \widehat{y}.y) : \tau}{x : (\sigma \to \tau) \cap \sigma \vdash_{\mathsf{cw}} x <^{x_1}_{x_2} x_1(x_2 :: \widehat{y}.y) : \tau}\,(Cont)}\,(Cut)}{\vdash_{\mathsf{cw}} \lambda x.x <^{x_1}_{x_2} x_1(x_2 :: \widehat{y}.y) : (\sigma \to \tau) \cap \sigma \to \tau}\,(\to_R).$$

Chapter 7

Conclusion

This dissertation presents an investigation of the formal calculi that correspond to two different levels of sequent-style computation in the framework of intuitionistic logic. It started from the λ^{Gtz}-calculus of Espírito Santo, and developed it by extending both the syntax and the type assignment system.

The extension of the type assignment system was motivated by the intention to obtain a system that assigns types exactly to those expressions whose computation always terminates, i.e. that satisfies the characterisation of strong normalisation. As already done for the λ-calculus, the desired property was obtained by introducing intersection types. The novelty of the proposed type system $\lambda^{\mathsf{Gtz}}\cap$ is its syntax directness that enabled to prove the characterisation of strong normalisation in a direct way, without using evaluation strategies.

In the domain of syntax extension, the main motivation was to obtain the control over quantitative transformations of the term evaluation, i.e. duplication and erasure of the variables. Following the ideas of the $\lambda\mathsf{lxr}$-calculus of Kesner and Lengrand, the solution was found in explicit operators for contraction and weakening of variables, called the resource control operators. In comparison with the λ^{Gtz}-calculus, the proposed $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus enables finer-grained computation, thus providing a mean to control and optimize resource management. The contribution in this part was in adapting these operators and their computational properties to the sequent calculus setting. Another contribution was an extension of the Curry-Howard correspondence to the intuitionistic sequent calculus with explicit structural rules, since it is evident that the simply typed $\lambda^{\mathsf{Gtz}}_{\circledR}$-calculus represents the computational interpretation of a variant of the system $G1$ for LJ.

The two extensions, namely intersection types and the resource control, were then combined and the system $\lambda^{\mathsf{Gtz}}_{\circledR}\cap$ was proposed. It turned out that the two concepts worked very well together. Our type assignment system is sensitive to a role of a variable in the expression; it assigns a strict type to a variable introduced by the axiom (which serves as a placeholder), an intersection type to a variable introduced by the contraction (whose role is to denote duplication) and finally, it assigns a specific type constant to a variable introduced by weakening (the one that can be erased). Moreover, it is syntax directed and is proved to characterise all strongly normalising $\lambda_{®}^{\mathsf{Gtz}}$-expressions.

Finally, all calculi were incorporated into a structure called the resource control cube, consisting of eight intuitionistic term calculi with implicit/explicit resource management and with natural deduction/sequent calculus logical setting. The λ^{Gtz}-calculus and the $\lambda_{®}^{\mathsf{Gtz}}$-calculus, as well as the λ-calculus, represent vertices of the cube. More precisely, the λ^{Gtz}-calculus and the $\lambda_{®}^{\mathsf{Gtz}}$-calculus belong to the so-called *LJ*-base of the cube, while the λ-calculus belongs to the *ND*-base. The four calculi of each base are treated in the uniform way, due to the presentation parameterized by the set of explicitly controlled resource operations. In that way, a system with different levels of the resource control is obtained, since one can impose total, partial or no resource control.

Future work

The investigation presented in this thesis can be broadened and continued in several directions.

Resource control term calculi, both in the natural deduction and sequent style, are good candidates to investigate the computational content of some substructural logics [65], a wide family of logics characterised by the absence of some of the structural rules. The most well-known substructural logics are the relevant logic (logic without weakening), the affine logic (without contraction) and particularly, the linear logic (without both weakening and contraction). The usual approach to the affine and relevant logic is via linear logic, thus their computational interpretations rely on the linear lambda calculus. Our point of view is that an other approach, starting from the $\lambda_{®}^{\mathsf{Gtz}}$ and $\lambda_{®}$ calculi is also possible and that it could be useful as a simple and neat logical foundation for the construction of specific relevant and affine programming languages, despite the fact that it could be considered naive because it only treats implicative fragments of the relevant and affine logics.

This thesis does not contain the research of the semantical properties of the proposed calculi. Therefore, it would be interesting to continue in that direction, particularly to investigate the use of intersection types in constructing models for sequent lambda calculi, since intersection types are known to be powerful means for building models of lambda calculus, as showed in [18].

The other appealing direction of research would be to compare and relate two different approaches to the resource control - the one enabled via explicit operators used here and the approach used in [55], where the resources are managed by introducing applicative structures called bags with multiplicities, and where terms are equipped with non-idempotent intersection types.

The proposed intersection type assignment systems exhibit the syntax directness property. More over, in the system with the resource control, a type of an expression contains detailed information about the structure of that expression, due to the presence of different kinds of types assigned to variables with different roles. Those features make the proposed system very convenient for the type reconstruction. It is expected that the type reconstruction algorithm based on the strict type assignment in the system with the control of resources would be significantly simpler than other similar procedures.

From a more pragmatic perspective, resources need to be controlled tightly in different applications. For instance, while working on description of compilers by rules with binders, Rose [61] noticed that the implementation of substitutions of linear variables is efficient, whereas substitutions of duplicated variables require a cumbersome and time consuming mechanism. It is therefore important to precisely control duplications and to minimize the necessity of duplication of variables, which coincides with the policy of propagating contraction into the expression of the $\lambda_{®}^{\mathsf{Gtz}}$-calculus. On the other hand, strong control of erasing (performed by the explicit weakening operator in $\lambda_{®}^{\mathsf{Gtz}}$) eliminates the need for a garbage collector and prevents memory leaking.

In the domain of object-oriented languages, Mycroft [54] presented resource aware type-systems for multi-core program efficiency, which represents another line of application of resource control. The type assignment system proposed in this thesis, which differs variables according to their role and assigns to them different kinds of types, could find application in this field.

Bibliography

[1] Amadio, R.M. and Curien, P.L.: Domains and lambda calculi. Cambridge University Press, Cambridge (1998).

[2] Baader, F. and Nipkow, T.: Term Rewriting and All That. Cambridge University Press, UK, (1998).

[3] Barendregt, H.P.: The Lambda Calculus - Its Syntax and Semantics. North-Holland, Amsterdam (1984).

[4] Barendregt, H.P.: Lambda calculi with types. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.): *Handbook of Logic in Computer Science,* Vol. 2, pages 117–309. Oxford University Press, Oxford (1992).

[5] Barendregt, H.P., Dekkers, W. and Statman, R.: Lambda Calculus With Types. Cambridge University Press, Cambridge (2013).

[6] Barendregt H.P., Coppo, M. and Dezani, M.: A filter lambda model and the completeness of type assignment. In: *Journal of Symbolic Logic*, Vol. 48, pages 931–940 (1983).

[7] Barendregt, H. and Ghilezan, S.: Lambda terms for natural deduction, sequent calculus and cut elimination. In: *Journal of Functional Programming*, Vol. 10, pages 121–134. Cambridge University Press (2000).

[8] Bloo, R. and Rose, K.H.: Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In: *Computer Science in the Netherlands, CSN '95*, pages 62–72 (1995).

[9] Boudol, G.: The lambda-calculus with multiplicities (abstract). In: E. Best, (ed.), *4th International Conference on Concurrency Theory - CONCUR'93,* Lecture Notes in Computer Science, Vol. 715, pages 1–6. Springer (1993).

[10] Church, A.: A set of postulates for the foundation of logic. *Annals of Mathematics*, Series 2, Vol. 33, pages 346–366 (1932).

[11] Church, A.: A formulation of the simple theory of types. *Journal of Symbolic Logic*, Vol. 5, pages 56–68 (1940).

[12] Church, A.: The Calculi of Lambda-Conversion. Princeton University Press, Princeton (1941).

[13] Coppo, M. and Dezani-Ciancaglini, M.: A new type-assignment for lambda terms. In: *Archiv für Mathematische Logik*, Vol. 19, pages 139–156, (1978).

[14] Curien, P.-L. and Herbelin, H.: The duality of computation. In: *5th International Conference on Functional Programming - ICFP'00*, pages 233–243. ACM Press (2000).

[15] Danos, V., Joinet, J-B. and Schellinx, H.: LKQ and LKT: Sequent calculi for second order logic based upon dual linear decomposition of classical implication. In: Girard, J-Y., Regnier, L. and Lafont, Y. (eds.):*The Workshop on Linear Logic*. Cornell (1993).

[16] David, R. and Guillaume, B.: A calculus with explicit weakening and explicit substitution. In: *Mathematical Structures in Computer Science*, Vol. 11, pages 169-206, (2001).

[17] Dershowitz, N.: Termination of rewriting. In: *Journal of Symbolic Computation*, Vol. 3(1/2), pages 69–116, (1987).

[18] Dezani-Ciancaglini, M., Ghilezan, S. and Likavec, S.: Behavioural Inverse Limit Models. In: *Theoretical Computer Science*. Vol. 316(1–3), pages 49–74. Elsevier (2004).

[19] Dougherty, D., Ghilezan, S. and Lescanne, P.: Characterizing strong normalization in the Curien-Herbelin symmetric lambda calculus: extending the Coppo-Dezani heritage. In: *Theoretical Computer Science*. Vol. 398, pages 114–128. Elsevier (2008).

[20] Espírito Santo, J. and Pinto, L.: Permutative Conversions in Intuitionistic Multiary Sequent Calculi with Cuts. In:*6th International Conference on Typed Lambda Calculi and Applications -TLCA'03*, Lecture Notes in Computer Science, Vol. 2071, pages 286–300. Springer-Verlag (2003).

[21] Espírito Santo, J.: An isomorphism between a fragment of sequent calculus and an extension of natural deduction. In: *9th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning - LPAR'02*, Lecture Notes in Artifitial Intelligence, Vol. 2514, pages 354–366. Springer-Verlag (2002).

[22] Espírito Santo, J.: A note on preservation of strong normalisation in the λ-calculus. In: *Theoretical Computer Science*, Vol. 412(11), pages 1027–1032. Elsevier (2011).

[23] Espírito Santo, J. and Pinto, L.: Confluence and Strong Normalisation of the Generalised Multiary Lambda Calculus. In: *International Workshop on Types for Proofs and Programs - TYPES'03*, Lecture Notes in Computer Science, Vol. 3085, pages 194–209. Springer-Verlag (2003).

[24] Espírito Santo, J., Ghilezan, S. and Ivetić, J.: Characterising strongly normalising intuitionistic sequent terms. In: *- International Workshop on Types for Proofs and Programs TYPES 2007 (Selected Papers)*. Lecture Notes in Computer Science, Vol. 4941, pages 85–99. Springer-Verlag (2007).

[25] Espírito Santo, J., Ivetić, J. and Likavec, S.: Characterising strongly normalising intuitionistic terms. In: *Fundamenta informaticae*, Vol. 121, pages 87–124. IOS Press, Nederlands (2012).

[26] Espírito Santo, J.: Delayed substitutions. In: Baader, F.(ed): *18th International Conference on Term Rewriting and Applications - RTA 2007*, Lecture Notes in Computer Science, Vol. 4533, pages 169–183. Springer-Verlag (2007).

[27] Espírito Santo, J.: Completing Herbelin's programme. In: S. Ronchi Della Rocca (ed.): *6th International Conference on Typed Lambda Calculi and Applications - TLCA 2007*, Lecture Notes in Computer Science, Vol. 4583, pages 118–132. Springer-Verlag (2007).

[28] Espírito Santo, J.: The lambda-calculus and the unity of structural proof theory. In: *Theory of Computing Systems*, Vol. 45, pages 963–994. Springer (2009).

[29] Gentzen, G.: Unterschungen über das logische Schliessen. In: M.E. Szabo (ed.): *Collected papers of Gerhard Gentzen*. North-Holland (1969).

[30] Ghilezan, S.: Strong normalization and typability with intersection types. In: *Notre Dame Journal of Formal Logic*, Vol. 37, pages 44–53, (1996).

[31] Ghilezan, S. and Ivetić, J.: Intersection types for λ^{Gtz}-calculus. In: *Publications de l'Institute Mathematique*. Vol. 82, pages 85–92. SANU, Srbija (2007).

[32] Ghilezan, S., Ivetić, J., Lescanne, P., and Likavec, S.: Intersection types for the resource control lambda calculi. In: A. Cerone and P. Pihlajasaari, (eds.), *8th International Colloquium on Theoretical Aspects of Computing, ICTAC '11*, Lecture Notes in Computer Science, Vol. 6916, pages 116–134. Springer (2011).

[33] Ghilezan, S., Ivetić, J., Lescanne, P., and Likavec, S.: A journey through resource control lambda calculi and explicit substitution using intersection types. Available at http:// arxiv. org/ abs/ 1306. 2283 (2013).

[34] Ghilezan, S., Ivetić, J., Lescanne, P., and Likavec, S.: Resource control and strong normalisation. Available at http:// arxiv. org/ abs/ 1112. 3455 (2011).

[35] Ghilezan, S., Ivetić, J., Lescanne, P., and Žunić, D.: Intuitionistic sequent-style calculus with explicit structural rules. In: *8th International Tbilisi Symposium on Language, Logic and Computation*, Lecture Notes in Artificial Intelligence, Vol. 6618, pages 101–124, Springer (2011).

[36] Ghilezan, S. and Likavec, S.: Computational interpretations of logics. In: *Collection of papers, Mathematical Institute SANU*, (special issue Logic in Computer Science, ed Z. Ognjanovic, invited paper). Vol. 12, pages 159–215. SANU, Srbija (2009).

[37] Girard, J-Y., Lafont, Y. and Taylor, P.: Proofs and Types. In: *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press (1989).

[38] Herbelin, H.: A lambda calculus structure isomorphic to Gentzen-style sequent calculus structure. In: *Computer Science Logic, CSL 1994*, Vol. 933, pages 61–75. Springer-Verlag (1995).

[39] Hindley, J.R. and Cardone, F.: History of lambda-calculus and combinatory logic. In: *Handbook of the History of Logic*, Volume 5 - Logic from Russell to Church (edited by D. Gabbay and J. Woods), pages 723–817, Elsevier (2009).

[40] Howard, W.A.: The formulae-as-types notion of construction. In: J.R. Hindley and J.P. Seldin, (eds.) *To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, London (1980).

[41] Ivetić, J.: Formalni računi za intuicionističku logiku (Formal calculi for intuitionistic logic). Master thesis, Faculty of technical sciences, University of Novi Sad (2008).

[42] Ivetić, J.: Regaining confluence in lambda-Gentzen calculus. In: *Algebra and Coalgenbra in Computer Science Young Researcher Workshop -CALCOjnr 2009*. Technical report, University of Udine (2009).

[43] Janičić, P.: Matematička logika u računarstvu. Matematički fakultet, Beograd (2009).

[44] Joachimski, F. and Matthes, R.: Standardization and confluence for a lambda calculus with generalized applications. In: Bachmair, L. (ed): *11th International Conference on Term Rewriting and Applications - RTA 2000*, Lecture Notes in Computer Science, Vol. 1833, pages 141–155. Springer-Verlag (2000).

[45] Kesner, D. and Lengrand, S.: Resource operators for lambda-calculus. In: *Information and Computation*, Vol. 205(4):419–473, (2007).

[46] Kesner, D. and Renaud, F.: The prismoid of resources. In: R. Královič and D. Niwiński, (eds.), *34th International Symposium on Mathematical Foundations of Computer Science - MFCS '09*, Lecture Notes in Computer Science, Vol. 5734, pages 464–476. Springer (2009).

[47] Kesner, D. and Renaud, F.: A prismoid framework for languages with resources. In: *Theoretical Computer Science*, Vol. 412(37), pages 4867–4892. Elsevier (2011).

[48] Kleene, S.C.: Introduction to Metamathematics. North-Holland, Amsterdam (1952).

[49] Krivine, J.L.: Lambda-calcul: types et modèles. Masson, Paris (1990).

[50] Landin, J. P.: Correspondence between ALGOL 60 and Church's Lambda-notation: part I. In: *Commun. ACM* Vol. 8(2), pages 89-101 (1965).

[51] Lengrand, S., Lescanne, P., Dougherty, D.J., Dezani-Ciancaglini, M. and van Bakel, S.: Intersection types for explicit substitutions. In: Longo, G. (ed): *Information and Computation,* Vol. 189, pages 17–42. Elsevier (2003).

[52] Lescanne, P. and Žunić, D.: Classical proofs' essence and diagrammatic computation. In: *International Conference on Numerical Analysis and Applied Mathematics - ICNAAM 2011*, AIP Conf. Proc., Vol. 1389, pages 792–797 (2011).

[53] Likavec, S.: Metod redukcije u lambda računu sa tipovima sa presekom (Reducibility method in the lambda calculus with intersection types). Master thesis. Faculty of Technical Sciences, University of Novi Sad (2005).

[54] Mycroft, A.: Using Kilim's Isolation Types for Multicore Efficiency. Invited talk at: *2nd International Conference on Formal Verification of Object-Oriented Software - FoVeOOS 2011*, (2011).

[55] Pagani, M. and Ronchi Della Rocca, S.: Solvability in resource lambda-calculus. In: C.-H.L. Ong, (ed.), *13th International Conference on Foundations of Software Science and Computational Structures - FOSSACS 2010*, Lecture Notes in Computer Science, Vol. 6014, pages 358–373. Springer (2010).

[56] Pottinger, G.: Normalization as Homomorphic Image of Cut-Elimination. In: *Annals of Mathematical Logic* Vol. 12, pages 323–357 (1977).

[57] Pottinger, G.: A type assignment for the strongly normalizable $\lambda-$terms. In: Seldin, J.P., Hindley, J.R. (Eds.), *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577, Academic Press, London (1980).

[58] Prawitz, D.: Natural deduction: a proof theoretical study (1965).

[59] Regnier, L.: Une équivalence sur les lambda-termes. In: *Theoretical Computer Science* Vol. 126, pages 281–292. Elsevier (1994).

[60] Ronchi Della Rocca, S.: Principal Type Scheme and Unification for Intersection Type Discipline. In: *Theoretical Computer Science* Vol. 59, pages 181–209. Elsevier (1988).

[61] Rose, K. H.: Implementation Tricks That Make CRSX Tick. In: *IFIP 1.6 workshop, Federated Conference on Rewriting, Deduction, and Programming - RDP'11*, (2011).

[62] Rose, K. H., Bloo, R and Lang, F.: On Explicit Substitution with Names. In: *Journal of Automated Reasoning* Vol. 49(2), pages 275–300, (2012).

[63] Sallé, P.: Une extension de la theorie des types en lambda-calcul. In: Ausiello, G., Böhm, C. (Eds.) *5th International Conference on Automata, Languages and Programming, ICALP'78*, Lecture Notes in Computer Science, Vol. 62, pages 398–410. Springer (1978).

[64] Schönfinkel, M.: Uber die Bausteine der mathematischen Logik [1924] (On the Building Blocks of Mathematical Logic). In: *J. van Heijenoort (ed.): From Frege to Gödel: a source book in mathematical logic, 1879-1931*. Harvard University Press (1967).

[65] Schroeder-Heister, P. and Došen, K.: Substructural Logics. Oxford University Press (1993).

[66] Schwichtenberg, H.: Termination of permutative conversions in intuitionistic Gentzen calculi. In: *Theoretical Computer Science* Vol. 212, pages 247–260. Elsevier (1999).

[67] Seldin, J.P. and Hindley, J.R. (eds.): To H.B. Curry: Essays on Combinatory Logic, Typed Lambda Calculus and Formalism. Academic Press, London (1980).

[68] Takahashi, M.: Parallel reduction in lambda calculus. In: *Information and Computation,* Vol. 118, pages 120–127. Academic press (1995).

[69] Troelstra, A. S. and Schwichtenberg, H.: Basic Proof Theory. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambrige, U.K. (1996).

[70] van Bakel, S.: Complete restrictions of the intersection type discipline. In: *Theoretical Computer Science*, Vol. 102, pages 136–163. Elsevier (1992).

[71] van Bakel, S., Lengrand, S. and Lescanne, P.: The Language chi: Circuits, Computations and Classical Logic. In: *9th Italian Conference on Theoretical Computer Science - ICTCS 2005,* Lecture Notes in Computer Science, Vol. 3701, pages 81–96. Springer (2005).

[72] van Oostrom, V.: Net-calculus. Course notes, `http://www.phil.uu.nl/ oostrom/oudonderwijs/cmitt/00-01/net.ps` (2001).

[73] Wadler, Ph.: Proofs are Programs: 19th Century Logic and 21st Century Computing. In: *DR. Dobbs Journal* (2000).

[74] Zucker, J.: The correspondence between cut-elimination and normalization. In: *Annals of Mathematical Logic*, Vol.7, pages 1–112 (1974).

[75] Žunić, D.: Computing with sequents and diagrams in classical logic - calculi ${}^*\mathcal{X}$, ${}^d\mathcal{X}$ and ${}^{©}\mathcal{X}$. Phd thesis, École Normale Supérieure de Lyon (2007).

List of Figures