УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Андрија Д. Урошевић

ФОРМАЛИЗАЦИЈА ИНТУИЦИОНИСТИЧКЕ ТЕОРИЈЕ ТИПОВА КАО УВОД У ХОМОТОПНУ ТЕОРИЈУ ТИПОВА

мастер рад

Београд, 2024.

Ментор:

др Сана Стојановић Ђурђевић, доцент
Универзитет у Београду, Математички факултет

Чланови комисије:

проф. др Филип Марић, редовни професор
Универзитет у Београду, Математички факултет

др Иван Чукић, доцент
Универзитет у Београду, Математички факултет

Датум одбране: dd. септембар 2024.

Породици, пријатељима, колегама,
планети Земљи и Универзуму

Наслов мастер рада: Формализација интуиционистичке теорије типова као увод у хомотопну теорију типова

Резиме: *Интуиционистичка теорија типова* (енгл. *Intuitionistic Type Theory*) или *Мартин Луф теорија типова* (МЛТТ) (енгл. *Martin–Löf Type Theory*) је математичка теорија конструкција и представља основу за алтернативни начин заснивања математике. За разлику од класичног заснивања математике преко теорије скупова, МЛТТ не захтева логички оквир, већ поставља основе како за математику тако и за логику. Генерално, МЛТТ добро повезује теорију типова и друге математике области, али уводи изазове за области као што су теорија категорија и хомотопна теорија. *Хомотопна теорија типова* (ХоТТ) (енгл. *Homotopy Type Theory*) проширују МЛТТ са високо индуктивним типовима и аксиомом унивалентности. Због тога, ХоТТ олакшавају рад над комплексим математичким структурама и повезују разне области: логику, теорију скупова, теорију категорија, хомотопну теорију, функционално програмирање, теорију типова и теорију доказа. МЛТТ представља стуб за ХоТТ, и као таква битна је за разумевање ХоТТ-а. Поред тога, МЛТТ пружа формалан систем чије варијанте имплементирају многи интерактивни доказивачи и због тога се примењује у формалној верификацији софтверских система осетљивих на грешке. Са друге стране, МЛТТ поставља многа филозофска питања о природи математике и природи математичких објеката. Овај рад пружа теоријске основе и практичне имплементације концепата МЛТТ-а у типски зависном програмском језику Agda.

Кључне речи: хомотопна теорија типова, интуиционистичка теорија типова, интерактивно доказивање теорема, формално заснивање математике

Садржај

Глава 1

Увод

Хомотопна теорија типова (ХоТТ) (енгл. *Homotopy Type Theory*) је нова област математике која повезује многе друге области. Ослања се на *хомотопну теорију* и *теорију типова*. Хомотопна теорија је област настала из алгебарске топологије и хомолошке алгебре, са идејама више теорије категорија, док теорија типова има корене у математичкој логици и теоријском рачунарству. Сматра се да ХоТТ представља алетернативно заснивање математике, поступком формализације уз помоћу интерактивних доказивача. Програм заснивања математике у ХоТТ се назива *универзалентне основе* (енгл. *Univalent Foundations*) [38].

Хомотопна теорија типова представља надоградњу *Мартин-Луф теорије типова* (МЛТТ) (енгл. *Martin-Löf Type Theory*) са *вишим индуктивним типовима* и *аксиомом универзалентности*. Виши индуктивни типови омогућавају логичко описивање основних простора и конструкција у хомотопној теорији (сфере, цилиндри, итд.). Са друге стране, аксиома универзалентности тврди да $(A = B) \simeq (A \simeq B)$, односно да је једнакост еквивалентна еквиваленцији.

Постоји пуно разлога за изучавање ХоТТ-а и заснивање математике кроз интерактивне доказиваче теорема, али један од главних је дао један од оснивача, Владимир Војводски (*Владимир Воеводский*), увидевши пропусте у туђим и својим радовима [43]:

> A technical argument by a trusted author, which is hard to check and looks similar to arguments known to be correct, is hardly ever checked in detail. (Владимир Војводски, 2014.)

1.1 Филозофија и историја

Теорију типова је оригинално представио Бертранд Расел [35] (*Bertrand Russell*), решавајући парадокс у заснивању математике тог времена. Након њега Алонзо Черч (*Alonzo Church*), развија *просто типизирани ламбда рачун* (ПТЛР) (енгл. *Simply Typed Lambda Calculus*) [11, 12]. Николас де Брујн (*Nicolaas de Bruijn*) инспирисан ПТЛР-ом развија први аутоматски доказивач теорема Automath [16]. Теорију типова даље развија Пер Мартин-Луф (*Per Martin-Löf*) коју данас знамо као МЛТТ, или *интуиционистичка/конструктивистичка/зависна теорија типова* [24, 25, 27, 26, 23]. МЛТТ представља основу за друге теорије које је проширују и које имплементирају разни интерактивни доказивачи теорема: Agda [32, 31], Coq [37], Lean [29, 28], Idris [7], NuPRL [14]. Инспирисани резултатом да теорија типова може да се интерпретира као ∞-групоид [20], Владимир Војводски [41], Стив Ауоди (*Steve Awodey*) и Мајкл Ворен (*Michael Warren*) [2] независно развијају ХоТТ.

МЛТТ, па самим тим и ХоТТ, се заснива на *Брауверовом интуиционистичком приступу* [8, 1] који тврди да се сва математика, укључујући и концепт доказа, изводи из концепта *конструкције* (рачуна/алгоритма/програма класификованог типом). Браувер је сматрао да је математичко резоновање људска активност и да је математика језик у коме се преносе математички концепти. Другим речима, способност извршавања алгоритма у сврху извођења менталне конструкције је фундаментално људска. Због тога, једини начин на који субјекат може доћи до математичке истине је да доживи истинитост, тако што изведе корак-по-корак одговарајућу менталну конструкцију. Слично, једини начин да субјекат дође до математичке неистине је да доживу њену неистиност, тако што схвати да извођење одговарајуће менталне конструкције није могуће. Интуиционистичи приступ даље развијају Колмогоров (*Андрей Николаевич Колмого́ров*) [22] и Хејтинг (*Arend Heyting*) [19] формулисањем интуиционистичке логике.

Интуиционистички приступ глобално искључује *закон искључења трећег* $P \vee \neg P$ [9, 39]. Један од разлог су *слаби контра-примери* (енгл. *weak counterexamples*). Пример једног слабог контра-примера је Голдбахова претпоставка: *Сваки паран број већи од* 2 *се може представити у облику збира два проста броја.* Конструктивистички, не можемо конструисати доказ да је Голдбахова претпоставка тачна, нити да је нетачна, а поред тога немамо

ни процедуру одлучивања. Због тога, не можемо тврдити да је Голдбахова претпоставка тачна или нетачна. Општије, не можемо тврдити да важи закон искључења трећег. Приметимо да закон искључења трећег искључујемо само глобално, односно да уколико је за конструкцију неког доказа потребно искористити неку конкретну инстанцу правила искључења трећег, довољно је навести је у претпоставкама тврђења. Отуда можемо запазити да теореме чији докази не користе закон искључења трећег имају снажнији резултат (јер користе мањи скуп претпоставки). Са друге стране, тврђења за чије је доказе неопходно искористити закон искључења трећег остају доказива. Закључно, све доказиве теореме у класичној математици остају доказиве и у интуиционистичко математици, а да при томе резултат теорема постаје снажнији.

Појам *типског расуђивања* (енгл. *type judgment*), који је у сржи теорије типова, записујемо као

$$t : T$$

и читамо као *t је терм типа T* или *терм t настањује тип T* (енгл. *t inhabits T*). Термови и типови се узимају као примитивни појмови који се не дефинишу. По Брауверу, терм t представља начин на који спроводимо конструкцију T. На пример, уколико желимо да конструишемо терм типа B и ако имамо конструисане термове $a : A$ и $f : A \to B$, онда терм $f(a)$ описује начин на који конструишемо терм типа B, односно $f(a) : B$. Често се у ХоТТ-у, за терм каже *тачка*, а за тип *простор*, па се тако за расуђивање $t : T$ каже да тачка t *припада* простору T.

Термови и типови у теорији типова имају три интерпретације: (1) докази исказа (логичка интерпретација), (2) програми типова (програмерска интерпретација) и (3) пресликавања структура (категоричка интерпретација). На пример, расуђивање $f : A \to B$ се може сматрати као: (1) доказ импликације, (2) функција која за дати улаз типа A враћа излаз типа B и (3) морфизам из објеката A у објекат B. Ову доктрину је поставио Роберт Харпер (Robert Harper) и назвао ју је *рачунарски тринитаризам* (енгл. *computational trinitarianism*) [18]. Рачунарски тринитаризам подразумева да сваки концепт који се јавља у једној интерпретацији треба да има значење у друге два интерпретације.

Теорија типова се заснива на идеји о *доказној релевантности* (енгл. *proof relevance*) која тврди да су математичка тврђења, па и сами докази, грађани првог реда, што значи да се различити докази истог исказа могу међусобно

поредити. Прецизније, ако су $p_0 : P$ и $p_1 : P$ докази исказа P, односно начини на који можемо конструисати тип P, тада се они могу поредити и у том смислу су *релевантни*. Са друге стране, код *доказно ирелевантних* система битно је само постојање доказа.

Доказивање исказа представља конструисање програма одређеног типа (за детаље видети поглавље 2.5). У том смислу, логика представља област математике, која се бави конструкцијама које су докази. Штавише, по Брауверу, математика је област рачунарства, јер су конструкције представљене рачуном/алгоритмом/програмом. Напоменимо да постоје и друге конструкције које нису докази (на пример, природни бројеви).

Још једна карактеристика МЛТТ-а, па самим тим и ХоТТ-а, је да користи *синтетички*, а не *аналитички* приступ. У синтетичком приступу, основни објекти су примитиве чије се особине и релације аксиоматизују, из којих се даље логички дедукују последице. У аналитичком приступу, основни објекти су конструисани од других објеката, а њихове особине и релације су дедуктоване из математичког окружења у ком су дефинисани. Често се за пример синтетичног приступа узима Еуклидска геометрија (где су основни објекти тачка, права и раван), а за аналитички приступ Декартова/аналитичка геометрија (где се основни објекти конструишу помоћу скупова).

1.2 Мотивација и циљ рада

Формализација математике представља једну од примарних делатности математичара. Од самог настанка математике, математичари су покушавали да прецизно заснују математику уводећи прецизно дефинисан дедуктивни систем, као и аксиоме теорије које формализују. Један од првих сачуваних примера су Еуклидови Елементи [17], који су, након Библије, највише проучавано, преписивано и штампано дело у људској историји. Елементи су представљали прво увођење аксиоматизације у област геометрије, накнадно су уочени одређени пропусти које су наредне аксиоматизације исправиле.

Стотинама година касније, многи велики математичари настоје да унапреде поступак формализације у смислу прецизнијег дефинисања дедуктивог система, а и самих метода/алата који се користе при формализацији. Вековима се уводе нови симболи, како би се резоновање у мозгу одвило механички, једноставним погледом. Почетком двадесетог века, математика и разноли-

кост симбола се у тој мери шири да се многи математичари из исте области међусобно не разумеју. Другим речима, користе различите симболе за исте појмове. Из тог разлога, 1934. године, формира се група математичара под колективним псеудонимом Николас Бурбаки (Nicolas Bourbaki). Бурбаки група, оригинално, има за циљ да формализује уџбенике математичке анализе. Након почетног успеха, објављује низ уџбеника из разних области математике као што су теорија скупова, апстрактна алгебра, топологија, и Лијева алгебра [6].

Данас се за било коју формализацију која се изводи помоћу *оловке и папира* сматра да је *неформална*. Због тога, модерни поступак формализације подразумева коришћење интерактивних доказивача теорема. Интерактивни доказивачи омогућавају алгоритамску проверу доказа која је од кључног значаја за исправност доказа. Иако интерактивни доказивачи представљају конкретну програмску имплементацију, па могу садржати грешке при имплементацији, те грешке скоро никада неће утицати на то да интерактивни доказивач прихвати неисправан доказ као исправан. Због тога се формализација унутар интективних доказивача сматра *формалнот*.

ХоТТ, тренутно, представља најбољи начин формалног заснивања математике. Такође, сматра се да ће имати велики утицај и бити део неких нових покушаја заснивања у будућности. Као такав, вредан је изучавања. Са друге стране, како је МЛТТ у основи ХоТТ-а, циљ овог рада се састоји у:

1. Детаљном теоријском и практичном изучавању МЛТТ;
2. Формализацији основних објеката и конструкција МЛТТ-а у типски зависном програмском језику Agda.

1.3 Друге формализације

Тренутно постоји неколицина библиотека које за циљ имају да формализују математику у оквиру ХоТТ-а. Прву и основну библиотеку, написану у интерактивном доказивачу Coq, објавио је Владимир Војводски, 2010. године, под називом Foundations [42]. Ова библиотека постала је део проширене библиотеке UniMath [44], коју је оригинално написало неколико аутора, такође у интерактивном доказивачу Coq. Након успеха ове две библиотеке, настаје пројекат унивалентног заснивања у коме се паралелно развијају две

нове библиотеке HoTT-Coq [4] и HoTT-Agda [10]. Година која је имала највећи утицај на област и заједницу је 2013. Те године, у Институту за напредне студије у Принстону, пише се и објављује прва неформална књига *Homotopy Type Theory: Univalent Foundations of Mathematics*, или краће *The HoTT Book* [38]. По узору на UniMath, од 2021. године, развија се и библиотека Agda-UniMath [34]. Такође, од 2021. године, на основу библиотеке UniMath, пише се књига *Symmetry* [5]. Све четири библиотеке се, и данас, активно развијају.

Једна мана XoTT-а је недостатак конструкције унивалентности. Унивалентност се аксиоматизује, те нема правило израчунавања. Због тога, није могуће конструисати типове за чију се конструкцију користи унивалентност. Другим речима, када систем наиђе на унивалентност неће знати како да је редукује. Као решење овог проблема настаје *коцкаста теорија типова* (енгл. *cubical type theory*) [13, 15]. Коцкаста теорија типова је имплементирана проширењем програмског језика Agda који се назива Cubical Agda. То је омогућило да настане библиотека cubical [40] која се, такође, активно развија.

Глава 2

Интуиционистичка теорија типова

Интуиционистичка теорија типова или Пер Мартин-Луф теорија типова је математичка теорија конструкција. Тип представља врсту конструкције. Елемент, терм или тачка представља резултат конструкције неког типа. Прецизније, елемент a типа A записујемо као $a : A$, и кажемо да елемент a *настањује* тип A. Битно је напоменути да терм не може да "живи самостално" тј. терм увек мора да настањује неки тип.

Конструкција типова се састоји из низа дедуктивних *правила закључивања*. Правило закључивања записујемо као

$$\frac{\mathcal{H}_1 \qquad \mathcal{H}_2 \qquad \dots \qquad \mathcal{H}_n}{\mathcal{C}}$$

где расуђивања $\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_n$ називамо *премисе* или *хипотезе*, а расуђивање $\mathcal{C}$ називамо *закључак*.

Дефиниција 2.0.1. Свако *расуђивање* је облика $\Gamma \vdash \mathcal{J}$, где је Γ *контекст* и $\mathcal{J}$ *теза* расуђивања.

Дефиниција 2.0.2. *Контекст расуђивања* је коначна листа узајамно зависних промењивих декларисаних на следећи начин

$$x_1 : A_1, x_2 : A_2(x_1), \dots, x_n : A_n(x_1, \dots, x_{n-1}),$$

под условом да за свако $1 \leq k \leq n$ можемо да изведемо расуђивање

$$x_1 : A_1, x_2 : A_2(x_1), \dots, x_{k-1} : A_{k-1}(x_1, \dots, x_{k-2}) \vdash A_k(x_1, x_2, \dots, x_{k-1}).$$

Дефиниција 2.0.3. *Теза расуђивања* може имати четири врсте расуђивања и то су:

(i) A је *(добро-формиран) тип* у контексту Γ

$$\Gamma \vdash A \text{ type}$$

(ii) A и B су *расуђивачки једнаки типови* у контексту Γ

$$\Gamma \vdash A \equiv B \text{ type}$$

(iii) a је *елемент* типа A у контексту Γ

$$\Gamma \vdash a : A$$

(iv) a и b су *расуђивачки једнаки елементи* типа A у контексту Γ

$$\Gamma \vdash a \equiv_A b : A$$

2.1 Правила закључивања

Интуиционистичка теорија типова, као и други математички формализми, захтева скуп правила закључивања на којима ће се формализам заснивати. Та правила називамо *структурна правила*.

Пример структурних правила закључивања која описују да је расуђивачка једнакост релација еквиваленције:

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash A \equiv A \text{ type}} \qquad \frac{\Gamma \vdash A \equiv A' \text{ type}}{\Gamma \vdash A' \equiv A \text{ type}} \qquad \frac{\Gamma \vdash A \equiv A' \text{ type} \qquad \Gamma \vdash A' \equiv A'' \text{ type}}{\Gamma \vdash A \equiv A'' \text{ type}}$$

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a \equiv_A a : A} \qquad \frac{\Gamma \vdash a \equiv_A a' : A}{\Gamma \vdash a' \equiv_A a : A} \qquad \frac{\Gamma \vdash a \equiv_A a' : A \qquad \Gamma \vdash a' \equiv_A a'' : A}{\Gamma \vdash a \equiv_A a'' : A}$$

Исцрпна листа структурних правила закључивања у интуиционистичкој теорији типова се може наћи у [33].

2.2 Зависни типови

Из дефиниције контекста можемо видети да неки типови могу зависити од неких термова. На пример, тип $A_2(x_1)$ зависи од терма $x_1 : A_1$, тј. за разне термове $x_1 : A_1$ имамо разне типове $A_2(x_1)$. Ову идеју можемо уопштити помоћу следећих дефиниција.

Дефиниција 2.2.1. Нека је тип A у контексту Γ. *Фамилија типова* над A у контексту Γ је тип $B(x)$ у контексту $\Gamma, x : A$, тј.

$$\Gamma, x : A \vdash B(x) \text{ type}.$$

Кажемо да је B *фамилија типова* над A у контексту Γ. Алтернативно, кажемо да је $B(x)$ тип *индексиран* са $x : A$ у контексту Γ.

Дефиниција 2.2.2. Нека је B фамилија типова над A у контексту Γ. *Секција фамилије* B над типом A у контексту Γ је елемент типа $B(x)$ у контексту $\Gamma, x : A$, тј.

$$\Gamma, x : A \vdash b(x) : B(x).$$

Кажемо да је b *секција фамилије* B над A у контексту Γ. Алтернативно, кажемо да је $b(x)$ елемент типа $B(x)$ *индексиран* са $x : A$ у контексту $\Gamma, x : A$.

Дефиниција 2.2.3. Нека је B фамилија типова над A у контексту Γ, и нека је $a : A$. Кажемо да је $B[a/x]$ *нит* (енгл. *fiber*) од B за параметар a, где $B[a/x]$ представља замену свих појављивања x у B са a. Нит од B за параметар a краће записујемо као $B(a)$.

Дефиниција 2.2.4. Нека је b секција фамилије типова B над A у контексту Γ. Кажемо да је $b[a/x]$ *вредност* од b за параметар a, где $b[a/x]$ представља замену свих појављивања x у b са a. Такође, вредност од b за параметар a краће записујемо као $b(a)$.

2.3 Типови зависних функција

У математици заснованој на теорији скупова функција $f : A \to B$ дефинисана је над одређеним доменом A и кодоменом B. У теорији типова то не мора да буде случај, тј. кодомен може зависити од елемента над којим се функција примењује. Прецизније, посматрајмо секцију b фамилије типова B

над A у контексту Γ. Један начин је да b посматрамо као функцију $x \mapsto b(x)$. Тада $b(x)$ настањује тип $B(x)$ који зависи од $x : A$. Због тога за разне елементе $x : A$ домена имамо разне кодомене, те има смисла говорити о типу *зависних функција* $\prod_{(x:A)} B(x)$.

Спецификација типа зависних функција $\prod_{(x:A)} B(x)$ је дата следећим правилима закључивања:

$$\frac{\Gamma, x : A \vdash B(x)\ \text{type}}{\Gamma \vdash \prod_{(x:A)} B(x)\ \text{type}}\ [\textstyle\prod\text{-form}] \qquad \frac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma \vdash \lambda x.b(x) : \prod_{(x:A)} B(x)}\ [\textstyle\prod\text{-intro}] \qquad \frac{\Gamma \vdash f : \prod_{(x:A)} B(x)}{\Gamma, x : A \vdash f(x) : B(x)}\ [\textstyle\prod\text{-elim}]$$

$$\frac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma \vdash (\lambda y.b(y))(x) \equiv b(x) : B(x)}\ [\textstyle\prod\text{-comp}_1] \qquad \frac{\Gamma \vdash f : \prod_{(x:A)} B(x)}{\Gamma \vdash \lambda x.f(x) \equiv f : \prod_{(x:A)} B(x)}\ [\textstyle\prod\text{-comp}_2]$$

Специјалан случај типа зависних функција је тип (уобичајених) *функција* $A \to B$. Уколико су типови A и B у контексту Γ, тј. тип B не зависи од елемената типа A, тада $\prod_{(x:A)} B$ представља тип (уобичајених) функција.

Дефиниција 2.3.1. Тип (уобичајених) *функција* $A \to B$ дефинишемо као:

$$A \to B := \prod_{(x:A)} B.$$

Ако је $f : A \to B$ функција, тада је A *домен*, а B *кодомен* функције f.

Дефиниција 2.3.2. За сваки тип A дефинишемо *функцију идентитета* $\mathsf{id}_A : A \to A$ као $\mathsf{id}_A :\equiv \lambda x.x$.

Дефиниција 2.3.3. За свака два типа A и B, и за сваку фамилију типова C над типом B дефинишемо *композицију* $\mathsf{comp} : (\prod_{(y:B)} C(y)) \to (A \to B) \to \prod_{(x:A)} C(f(x))$ као $\mathsf{comp} :\equiv \lambda g.\lambda f.\lambda x.g(f(x))$.

Може се показати да је композиција асоцијативна, као и да је функција идентитета неутрал за композицију функција. Због сагласности типова имамо леви неутрал id_B и десни неутрал id_A.

2.4 Индуктивни типови

Поред типова зависних функција постоји и класа *индуктивних типова*. Сваки индуктивни тип се дефинише помоћу следеће спецификације:

(i) *Формирање* типа описује начин на који се дати тип формира. Прецизније, дефинише хипотезе на основу којих је могуће формирати дати тип.

(ii) *Конструисање* описује на који начин се уводе нови канонични термови датог типа. Прецизније, дефинише функције које конструишу каноничне термове датог типа, као и потребне хипотезе за њихово постојање. Те функције се често називају *конструктори*.

(iii) *Индуктивни принцип* описује податке који су потребни да би се конструисала секција произвољне фамилије типова над датим типом. Другим речима, дефинише хипотезе за постојање функције $\mathsf{ind}_A : \prod_{(x:A)} P(x)$.

(iv) *Правила израчунавања* захтевају да се индуктивно дефинисана секција произвољне фамилије типова над датим типом поклапа по конструкторима који уводе нове каноничне термове. Другим речима, за сваки конструктор $c : A$, мора да важи $\mathsf{ind}_A(c) \equiv \square : P(c)$, где $\square$ представља одговарајући израз/рачун.

Обично се, поред ових спецификација, уводи и *правило рекурзије* које је специјални случај правила индукције. Код правила рекурзије не конструишемо секцију произвољне фамилије типова над датим типом, већ само константну фамилију над датим типом. Другим речима, дефинишемо хипотезе за постојање функције $\mathsf{rec}_A : A \to P$, као и одговарајућа правила израчунавања.

У наставку су наведене спецификације за уобичајене индуктивне типове: тип природних бројева $\mathbb{N}$, празан тип $\mathbb{0}$, јединични тип $\mathbb{1}$, типови копроизвода $A + B$, тип зависних парова $\sum_{(x:A)} B(x)$, као и специјални случајеви ових типова. Поред њих, у засебном поглављу ће бити представљени типови идентитета $x =_A y$.

Тип природних бројева

Тип природних бројева $\mathbb{N}$ представља тип кога настањују природни бројеви $0_{\mathbb{N}}, 1_{\mathbb{N}}, 2_{\mathbb{N}}, \ldots$. Тип природних бројева $\mathbb{N}$ и његове каноничне елементе уводимо помоћу следећих правила:

$$\frac{}{\vdash \mathbb{N}\ \text{type}}\ [\mathbb{N}\text{-form}] \qquad \frac{}{\vdash 0_{\mathbb{N}} : \mathbb{N}}\ [\mathbb{N}\text{-intro}_{0_{\mathbb{N}}}] \qquad \frac{}{\vdash \mathsf{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}\ [\mathbb{N}\text{-intro}_{\mathsf{succ}_{\mathbb{N}}}]$$

По правилу $\mathbb{N}$-form, тип природних бројева $\mathbb{N}$ може да се формира из празног контекста. Другим речима, постојање типа природних бројева $\mathbb{N}$ не зависи од постојања других типова. Даље, имамо два конструктора помоћу којих конструишемо све каноничке термове типа $\mathbb{N}$. Први конструктор је константа $0_{\mathbb{N}} : \mathbb{N}$ и он говори да је $0_{\mathbb{N}}$ канонични терм типа $\mathbb{N}$. Други конструктор је функција $\mathsf{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ и она говори да ће $\mathsf{succ}_{\mathbb{N}}(n)$ бити канонични терм типа $\mathbb{N}$ ако је $n : \mathbb{N}$ канонични терм. Због тога су $0_{\mathbb{N}}, \mathsf{succ}_{\mathbb{N}}(0_{\mathbb{N}}), \mathsf{succ}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(0_{\mathbb{N}})), \ldots$ канонични термови који настањују тип $\mathbb{N}$.

Правила формирања и конструкције нам говоре под којим условима се може формирати тип, и како конструисати каноничне термове тог типа. Потребно је још дефинисати и начин на који се тип и елементи тог типа користе. Због тога се уводи индуктивно правило и правила израчунавања. Да би конструисали елемент $\mathsf{ind}_{\mathbb{N}}(p_{0_{\mathbb{N}}}, p_{\mathsf{succ}_{\mathbb{N}}}) : \prod_{(n:\mathbb{N})} P(n)$ потребно је конструисати елемент $p_{0_{\mathbb{N}}} : P(0_{\mathbb{N}})$ (*база индукције*) и $p_{\mathsf{succ}_{\mathbb{N}}} : \prod_{n:\mathbb{N}} P(n) \to P(\mathsf{succ}_{\mathbb{N}}(n))$ (*индуктивни корак*).

$$[\mathbb{N}\text{-ind}]\ \frac{\begin{array}{l}\Gamma, n : \mathbb{N} \vdash P(n)\ \text{type}\\ \Gamma \vdash p_{0_{\mathbb{N}}} : P(0_{\mathbb{N}})\\ \Gamma \vdash p_{\mathsf{succ}_{\mathbb{N}}} : \prod_{(n:\mathbb{N})} P(n) \to P(\mathsf{succ}_{\mathbb{N}}(n))\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(p_{0_{\mathbb{N}}}, p_{\mathsf{succ}_{\mathbb{N}}}) : \prod_{(n:\mathbb{N})} P(n)}$$

За сваки од конструктора треба увести правило израчунавања у складу са зависном функцијом $\mathsf{ind}_{\mathbb{N}}(p_{0_{\mathbb{N}}}, p_{\mathsf{succ}_{\mathbb{N}}}) : \prod_{(n:\mathbb{N})} P(n)$. Због тога имамо два правила израчунавања $\mathbb{N}\text{-comp}_{0_{\mathbb{N}}}$ и $\mathbb{N}\text{-comp}_{\mathsf{succ}_{\mathbb{N}}}$. За правило израчунавања $\mathbb{N}\text{-comp}_{0_{\mathbb{N}}}$ потребно је дефинисати вредност функције $\mathsf{ind}_{\mathbb{N}}(p_{0_{\mathbb{N}}}, p_{\mathsf{succ}_{\mathbb{N}}}) : \prod_{(n:\mathbb{N})} P(n)$ за аргумент $0_{\mathbb{N}} : N$. Односно, у изразу $\mathsf{ind}_{\mathbb{N}}(p_{0_{\mathbb{N}}}, p_{\mathsf{succ}_{\mathbb{N}}}, 0_{\mathbb{N}}) \equiv \square : P(0_{\mathbb{N}})$ треба заменити $\square$ одговарајућом конструкцијом. Како је база индукције $p_{0_{\mathbb{N}}}$ типа $P(0_{\mathbb{N}})$, онда смо пронашли одговарајућу конструкцију. Наиме, $\mathsf{ind}_{\mathbb{N}}(p_{0_{\mathbb{N}}}, p_{\mathsf{succ}_{\mathbb{N}}}, 0_{\mathbb{N}}) \equiv$

$p_{0_\mathbb{N}} : P(0_\mathbb{N})$ је тражено правило израчунавања. Слично можемо извршити одговарајућу конструкцију за правило $\mathbb{N}\text{-comp}_{\mathsf{succ}_\mathbb{N}}$. Формално правила израчунавања дефинишемо као:

$$[\mathbb{N}\text{-comp}^{\mathsf{ind}_\mathbb{N}}_{0_\mathbb{N}}] \quad \frac{\begin{array}{l}\Gamma, n : \mathbb{N} \vdash P(n) \text{ type} \\ \Gamma \vdash p_{0_\mathbb{N}} : P(0_\mathbb{N}) \\ \Gamma \vdash p_{\mathsf{succ}_\mathbb{N}} : \prod_{(n:\mathbb{N})} P(n) \to P(\mathsf{succ}_\mathbb{N}(n))\end{array}}{\Gamma \vdash \mathsf{ind}_\mathbb{N}(p_{0_\mathbb{N}}, p_{\mathsf{succ}_\mathbb{N}}, 0_\mathbb{N}) \equiv p_{0_\mathbb{N}} : P(0_\mathbb{N})}$$

$$[\mathbb{N}\text{-comp}^{\mathsf{ind}_\mathbb{N}}_{\mathsf{succ}_\mathbb{N}}]$$

$$\frac{\begin{array}{l}\Gamma, n : \mathbb{N} \vdash P(n) \text{ type} \\ \Gamma \vdash p_{0_\mathbb{N}} : P(0_\mathbb{N}) \\ \Gamma \vdash p_{\mathsf{succ}_\mathbb{N}} : \prod_{(n:\mathbb{N})} P(n) \to P(\mathsf{succ}_\mathbb{N}(n))\end{array}}{\Gamma,\ n : \mathbb{N} \vdash \mathsf{ind}_\mathbb{N}(p_{0_\mathbb{N}}, p_{\mathsf{succ}_\mathbb{N}}, \mathsf{succ}_\mathbb{N}(n)) \equiv p_{\mathsf{succ}_\mathbb{N}}(n, \mathsf{ind}_\mathbb{N}(p_{0_\mathbb{N}}, p_{\mathsf{succ}_\mathbb{N}}, n)) : P(\mathsf{succ}_\mathbb{N}(n))}$$

Специјални случај индукције типа природних бројева је рекурзија типа природних бројева, у којој тип P не зависи од $\mathbb{N}$. Тада добијамо функцију $\mathsf{rec}_\mathbb{N}(a_{0_\mathbb{N}}, a_{\mathsf{succ}_\mathbb{N}}) : \mathbb{N} \to A$, под условом да имамо елементе $a_{0_\mathbb{N}} : A$ и $a_{\mathsf{succ}_\mathbb{N}} : \mathbb{N} \to A \to A$.

$$[\mathbb{N}\text{-rec}_\mathbb{N}] \quad \frac{\begin{array}{l}\Gamma \vdash A \text{ type} \\ \Gamma \vdash a_{0_\mathbb{N}} : A \\ \Gamma \vdash a_{\mathsf{succ}_\mathbb{N}} : \mathbb{N} \to A \to A\end{array}}{\Gamma \vdash \mathsf{rec}_\mathbb{N}(a_{0_\mathbb{N}}, a_{\mathsf{succ}_\mathbb{N}}) : \mathbb{N} \to A}$$

$$[\mathbb{N}\text{-comp}^{\mathsf{rec}_\mathbb{N}}_{0_\mathbb{N}}] \quad \frac{\begin{array}{l}\Gamma \vdash A \text{ type} \\ \Gamma \vdash a_{0_\mathbb{N}} : A \\ \Gamma \vdash a_{\mathsf{succ}_\mathbb{N}} : \mathbb{N} \to A \to A\end{array}}{\Gamma \vdash \mathsf{rec}_\mathbb{N}(a_{0_\mathbb{N}}, a_{\mathsf{succ}_\mathbb{N}}, 0_\mathbb{N}) \equiv a_{0_\mathbb{N}} : A}$$

$$[\mathbb{N}\text{-comp}^{\mathsf{rec}_\mathbb{N}}_{\mathsf{succ}_\mathbb{N}}]$$

$$\frac{\begin{array}{l}\Gamma \vdash A \text{ type} \\ \Gamma \vdash a_{0_\mathbb{N}} : A \\ \Gamma \vdash a_{\mathsf{succ}_\mathbb{N}} : \mathbb{N} \to A \to A\end{array}}{\Gamma, n : \mathbb{N} \vdash \mathsf{rec}_\mathbb{N}(a_{0_\mathbb{N}}, a_{\mathsf{succ}_\mathbb{N}}, \mathsf{succ}_\mathbb{N}(n)) \equiv a_{\mathsf{succ}_\mathbb{N}}(n, \mathsf{rec}_\mathbb{N}(a_{0_\mathbb{N}}, a_{\mathsf{succ}_\mathbb{N}}, n)) : A}$$

Правило индукције, заједно са правилом рекурзије, омогућава дефинисање разних функција над природним бројевима. Да би дефинисали операцију сабирања природних бројева $+_\mathbb{N} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ можемо искористити правило рекурзије, тј. функцију $\mathsf{rec}_\mathbb{N} : A \to (\mathbb{N} \to A \to A) \to \mathbb{N} \to A$. За тип A узећемо $\mathbb{N}$. Због тога, сабирање природних бројева дефинишемо као:

$$m +_\mathbb{N} n :\equiv \mathsf{rec}_\mathbb{N}(m, \lambda n.\lambda r.\mathsf{succ}_\mathbb{N}(r), n).$$

Заиста, за овако дефинисану операцију сабирања важи:

$$m +_{\mathbb{N}} 0_{\mathbb{N}} \equiv m;$$
$$m +_{\mathbb{N}} \mathsf{succ}_{\mathbb{N}}(n) \equiv \mathsf{succ}_{\mathbb{N}}(m +_{\mathbb{N}} n).$$

Слично, множење природних бројева $\times_{\mathbb{N}} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ можемо дефинисати као:

$$m \times_{\mathbb{N}} n :\equiv \mathsf{rec}_{\mathbb{N}}(0_{\mathbb{N}}, \lambda n.\lambda r.m +_{\mathbb{N}} r, n).$$

Такође, за овако дефинисану операцију множења важи:

$$m \times_{\mathbb{N}} 0_{\mathbb{N}} \equiv 0_{\mathbb{N}};$$
$$m \times_{\mathbb{N}} \mathsf{succ}_{\mathbb{N}}(n) \equiv (m +_{\mathbb{N}} (m \times_{N} n)).$$

Можемо приметити шаблон између дефинисања операција преко рекурзивног правила и правила која захтевамо да важе по конструкторима. Наиме, уколико желимо да дефинишемо функцију $f : \mathbb{N} \to A$ за коју важи:

$$f(0_{\mathbb{N}}) \equiv \Phi_{0_{\mathbb{N}}};$$
$$f(\mathsf{succ}_{\mathbb{N}}(n)) \equiv \Phi_{\mathsf{succ}_{\mathbb{N}}},$$

где је $\Phi_{0_{\mathbb{N}}}$ израз типа A, и $\Phi_{\mathsf{succ}_{\mathbb{N}}}$ израз типа A који може садржати n и $f(n)$. Тада функцију $f : \mathbb{N} \to A$ дефинишемо као:

$$f :\equiv \mathsf{rec}_{\mathbb{N}}(\Phi_{0_{\mathbb{N}}}, \lambda n.\lambda r.\Phi'_{\mathsf{succ}_{\mathbb{N}}}),$$

где $\Phi'_{\mathsf{succ}_{\mathbb{N}}}$ добијемо из $\Phi_{\mathsf{succ}_{\mathbb{N}}}$ тако што сва појављивања $f(n)$ заменимо са r. Овај поступак дефинисања можемо уопштити и на индуктивно правило, и тада се он назива *упаривање шаблона* (енгл. *pattern matching*).

Празан тип

Празан тип $\mathbb{0}$ је дегенерисани пример индуктивног типа кога не настањује ни један елемент. Прецизније, празан тип $\mathbb{0}$ дефинишемо следећом спецификацијом:

$$[\mathbb{0}\text{-form}]\ \frac{}{\vdash \mathbb{0}\ \text{type}} \quad [\mathbb{0}\text{-ind}]\ \frac{\Gamma, 0 \vdash P(x)\ \text{type}}{\Gamma \vdash \mathsf{ind}_{\mathbb{0}} : \prod_{(x:\mathbb{0})} P(x)} \quad [\mathbb{0}\text{-rec}]\ \frac{\Gamma \vdash A\ \text{type}}{\Gamma \vdash \mathsf{rec}_{\mathbb{0}} : \mathbb{0} \to A}$$

Како празан тип $\mathbb{0}$ не настањује ни један елемент, за њега не постоји ни један конструктор, и самим тим нема ни једно правило израчунавања. Може

да се формира из празног контекста, а његово правило индукције тврди да за било коју фамилију типова P над $\mathbb{0}$ постоји елемент $\mathsf{ind}_{\mathbb{0}} : \prod_{(x:\mathbb{0})} P(x)$. Чешће се користи правило рекурзије које тврди да уколико конструишемо елемент $x : \mathbb{0}$, онда можемо да конструишемо елемент $\mathsf{rec}_{\mathbb{0}}(x) : A$ било ког типа A. Правило рекурзије за празан тип $\mathbb{0}$ се обично назива и *правило контрадикције* или *правило противречности*.

Дефиниција 2.4.1. За сваки тип A дефинишемо тип *негације* od A као $\neg A := A \to \mathbb{0}$. Поред тога, кажемо да је тип A *празан* ако његову негацију настањује неки елемент, тј. $\mathsf{empty}(A) := A \to \mathbb{0}$.

Приметимо да је *двострука негација* од A дефинисана као $\neg\neg A := (A \to \mathbb{0}) \to \mathbb{0}$. Због тога, не мора да важи $\neg\neg A \to A$, те у општем случају није могуће изводити доказе контрадикцијом.

Јединични тип

Јединични тип $\mathbb{1}$ је индуктивни тип кога настањује само елемент $\star$. Прецизније, јединични тип $\mathbb{1}$ дефинишемо следећом спецификацијом:

$$[\mathbb{1}\text{-form}]\ \frac{}{\vdash \mathbb{1}\ \text{type}} \qquad [\mathbb{1}\text{-intro}_\star]\ \frac{}{\vdash \star : \mathbb{1}}$$

$$[\mathbb{1}\text{-ind}]\ \frac{\Gamma, x : \mathbb{1} \vdash P(x)\ \text{type} \quad \Gamma \vdash p_\star : P(\star)}{\Gamma \vdash \mathsf{ind}_{\mathbb{1}}(p_\star) : \prod_{(x:\mathbb{1})} P(x)} \qquad [\mathbb{1}\text{-comp}]\ \frac{\Gamma, 1 \vdash P(x)\ \text{type} \quad \Gamma \vdash p_\star : P(\star)}{\Gamma \vdash \mathsf{ind}_{\mathbb{1}}(p_\star, \star) \equiv p_\star : P(\star)}$$

$$[\mathbb{1}\text{-rec}]\ \frac{\Gamma \vdash A\ \text{type} \quad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{rec}_{\mathbb{1}}(a) : \mathbb{1} \to A}$$

Јединични тип $\mathbb{1}$ може да се формира из празног контекста, а његово правило индукције тврди да за било коју фамилију типова P над $\mathbb{1}$ постоји елемент $\mathsf{ind}_{\mathbb{1}}(p_\star) : \prod_{(x:\mathbb{1})} P(x)$ уколико постоји елемент $p_\star : P(\star)$. Како постоји само један конструктор $\star : \mathbb{1}$, имамо једно правило израчунавања које треба да се поклопи са индуктивним правилом. Због тога, $\mathsf{ind}_{\mathbb{1}}(p_\star, \star) \equiv p_\star : P(\star)$.

Специјални случај правила индукције типа $\mathbb{1}$ је правило рекурзије типа $\mathbb{1}$, које добијамо када фамилија типова P над $\mathbb{1}$ не зависи од $x : \mathbb{1}$. Тада за сваки елемент $a : A$ имамо функцију $\mathsf{rec}_{\mathbb{1}}(a) : \mathbb{1} \to A$.

Дефиниција 2.4.2. За сваки тип A дефинишемо тип *јединствене функције* од A као $!\mathbb{1}(A) := A \to \mathbb{1}$. Специјално, јединствена функција од $\mathbb{0}$, тј. $\mathbb{0} \to \mathbb{1}$, се назива *вакумска функција*.

У хомотопној теоријити типова за вакумску функцију важи да је јединствена.

Типови копроизвода

За типове A и B из контекста Γ можемо дефинисати тип копроизвода $A+B$ кога ће настањивати елементи или из типа A (ако $a : A$, онда $\mathsf{inl}(a) : A + B$) или из типа B (ако $b : B$, онда $\mathsf{inr}(b) : A + B$). Прецизније, тип копроизвода $A + B$ дефинишемо следећом спецификацијом:

$$\frac{\Gamma \vdash A \text{ type}, B \text{ type}}{\Gamma \vdash A + B \text{ type}}\ [+\text{-form}] \qquad \frac{}{\Gamma \vdash \mathsf{inl} : A \to A + B}\ [+\text{-intro}_{\mathsf{inl}}] \qquad \frac{}{\Gamma \vdash \mathsf{inr} : B \to A + B}\ [+\text{-intro}_{\mathsf{inr}}]$$

$$[+\text{-ind}]\ \frac{\begin{array}{l}\Gamma, z : A + B \vdash P(z) \text{ type} \\ \Gamma \vdash p_{\mathsf{inl}} : \prod_{(a:A)} P(\mathsf{inl}(a)) \\ \Gamma \vdash p_{\mathsf{inr}} : \prod_{(b:B)} P(\mathsf{inr}(b))\end{array}}{\Gamma \vdash \mathsf{ind}_+(p_{\mathsf{inl}}, p_{\mathsf{inr}}) : \prod_{(z:A+B)} P(z)}$$

$$[+\text{-comp}]\ \frac{\begin{array}{l}\Gamma, z : A + B \vdash P(z) \text{ type} \\ \Gamma \vdash p_{\mathsf{inl}} : \prod_{(a:A)} P(\mathsf{inl}(a)) \\ \Gamma \vdash p_{\mathsf{inr}} : \prod_{(b:B)} P(\mathsf{inr}(b))\end{array}}{\begin{array}{l}\Gamma, a : A \vdash \mathsf{ind}_+(p_{\mathsf{inl}}, p_{\mathsf{inr}}, \mathsf{inl}(a)) \equiv p_{\mathsf{inl}}(a) : P(\mathsf{inl}(a)) \\ \Gamma, b : B \vdash \mathsf{ind}_+(p_{\mathsf{inl}}, p_{\mathsf{inr}}, \mathsf{inr}(b)) \equiv p_{\mathsf{inr}}(b) : P(\mathsf{inr}(b))\end{array}}$$

$$[+\text{-rec}]\ \frac{\begin{array}{l}\Gamma, z : A + B \vdash X \text{ type} \\ \Gamma \vdash f : A \to X \\ \Gamma \vdash g : B \to X\end{array}}{\Gamma \vdash \mathsf{rec}_+(f, g) : A + B \to X}$$

Тип копроизвода $A + B$ због своје природе има два конструктора $\mathsf{inl} : A \to A + B$ і $\mathsf{inr} : B \to A + B$. Правило индукције тврди да за било коју фамилију типова P над $A + B$ постоји елемент $\mathsf{ind}_+(p_{\mathsf{inl}}, p_{\mathsf{inr}}) : \prod_{(z:A+B)} P(z)$ уколико постоје елементи $p_{\mathsf{inl}} : \prod_{(a:A)} P(\mathsf{inl}(a))$ и $p_{\mathsf{inr}} : \prod_{(b:B)} P(\mathsf{inr}(b))$. Како постоје два конструктора, имамо два правила израчунавања која треба да се поклопе

са правилом индукције. Због тога, $\mathsf{ind}_+(p_{\mathsf{inl}}, p_{\mathsf{inr}}, \mathsf{inl}(a)) \equiv p_{\mathsf{inl}}(a) : P(\mathsf{inl}(a))$ и $\mathsf{ind}_+(p_{\mathsf{inl}}, p_{\mathsf{inr}}, \mathsf{inr}(b)) \equiv p_{\mathsf{inr}}(b) : P(\mathsf{inr}(b))$.

Специјални случај правила индукције типа $A + B$ је правило рекурзије типа $A + B$, које добијамо када фамилија типова P над $A + B$ не зависи од $z : A + B$. Тада за сваку функцију $f : A \to X$ и за сваку функцију $g : B \to X$ имамо функцију $\mathsf{rec}_+(f, g) : A + B \to X$. Из правила индукције, за свако $f : A \to X$ и за свако $g : B \to Y$, имамо функцију $f + g : A + B \to X + Y$.

Специјални случај типа копроизвода је *Булов тип* $2 := \mathbb{1} + \mathbb{1}$, чије једине елементе дефинишемо као $\mathsf{true} :\equiv \mathsf{inl}(\star)$ и $\mathsf{false} :\equiv \mathsf{inr}(\star)$. Из спецификације типа копроизвода можемо извући правило индукције и правило израчунавања, за буловски тип 2. Правило индукције 2-ind се назива и *if-then-else*.

$$[2\text{-ind}]\quad \frac{\begin{array}{l}\Gamma, x : 2 \vdash P(x)\ \text{type}\\ \Gamma \vdash p_{\mathsf{true}} : P(\mathsf{true})\\ \Gamma \vdash p_{\mathsf{false}} : P(\mathsf{false})\end{array}}{\Gamma \vdash \mathsf{ind}_2(p_{\mathsf{true}}, p_{\mathsf{false}}) : \prod_{(x:2)} P(x)}$$

$$[2\text{-comp}]\quad \frac{\begin{array}{l}\Gamma, x : 2 \vdash P(x)\ \text{type}\\ \Gamma \vdash p_{\mathsf{true}} : P(\mathsf{true})\\ \Gamma \vdash p_{\mathsf{false}} : P(\mathsf{false})\end{array}}{\begin{array}{l}\Gamma \vdash \mathsf{ind}_2(p_{\mathsf{true}}, p_{\mathsf{false}}, \mathsf{true}) \equiv p_{\mathsf{true}} : P(\mathsf{true})\\ \Gamma \vdash \mathsf{ind}_2(p_{\mathsf{true}}, p_{\mathsf{false}}, \mathsf{false}) \equiv p_{\mathsf{false}} : P(\mathsf{true})\end{array}}$$

Типови зависних парова

Ако је B фамилија типова над A из контекста Γ, онда можемо формирати тип зависних парова $\sum_{(x:A)} B(x)$ кога ће настањивати *парови* $(x, y(x))$, где је $x : A$ и $y(x) : B(x)$. Прецизније, тип зависних парова $\sum_{(x:A)} B(x)$ дефинишемо следећом спецификацијом.

$$\frac{[\textstyle\sum\text{-form}]\quad \Gamma, x : A \vdash B(x)\ \text{type}}{\Gamma \vdash \sum_{(x:A)} B(x)\ \text{type}} \qquad \frac{[\textstyle\sum\text{-intro}]\quad \Gamma, x : A \vdash y(x) : B(x)}{\Gamma \vdash (x, y(x)) : \sum_{(x:A)} B(x)}$$

$$[\textstyle\sum\text{-ind}]\quad \frac{\begin{array}{l}\Gamma, (x, y) : \sum_{(x:A)} B(x) \vdash P((x, y))\ \text{type}\\ \Gamma \vdash f : \prod_{(x:A)} \prod_{(y:B(x))} P((x, y))\end{array}}{\Gamma \vdash \mathsf{ind}_\sum(f) : \prod_{(p:\sum_{(x:A)} B(x))} P(p)}$$

$$[\textstyle\sum\text{-comp}]\quad \frac{\begin{array}{l}\Gamma, (x,y) : \sum_{(x:A)} B(x) \vdash P((x,y)) \text{ type} \\ \Gamma \vdash f : \prod_{(x:A)} \prod_{(y:B(x))} P((x,y))\end{array}}{\Gamma, (x,y) : \sum_{(x:A)} B(x) \vdash \mathsf{ind}_{\sum}(f,(x,y)) \equiv f(x,y) : P((x,y))}$$

Тип зависних парова $\sum_{(x:A)} B(x)$ има један конструктор помоћу кога се могу формирати елементи који га настањују, и то једноставним упаривањем елемената $x : A$ и $y(x) : B(x)$. Правило индукције тврди да за било коју фамилију типова P над $\sum_{(x:A)} B(x)$ постоји елемент $\mathsf{ind}_{\sum}(f) : \prod_{p:\sum_{(x:A)} B(x)} P(p)$ уколико постоји елемент $f : \prod_{(x:A)} \prod_{(y:B(x))} P((x,y))$. Како постоји само један конструктор, имамо само једно правило израчунавања које треба да се поклопи са правилом индукције. Због тога, важи $\mathsf{ind}_{\sum}(f,(x,y)) \equiv f(x,y) : P((x,y))$.

Правило индукције нам омогућава да дефинишемо функције у наставку.

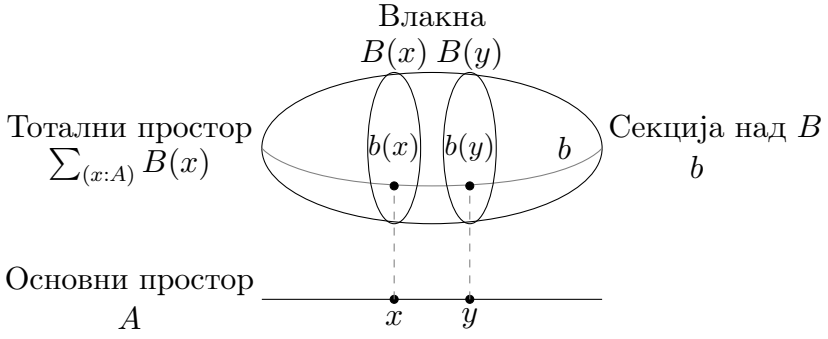
Дефиниција 2.4.3. Нека је B фамилија типова над A. Тада елемент $\mathsf{pr}_1 : \sum_{(x:A)} B(x) \to A$ *пројекције на први елемент* дефинишемо као:

$$\mathsf{pr}_1((a,b)) :\equiv\ a, \tag{2.1}$$

а елемент $\mathsf{pr}_2 : \prod_{p:\sum_{(x:A)} B(x)} B(\mathsf{pr}_1(p))$ *пројекције на други елемент* дефинишемо као:

$$\mathsf{pr}_2((a,b)) :\equiv\ b. \tag{2.2}$$

Ако претпоставимо да имамо елемент $f : \prod_{((x,y):\sum_{(x:A)} B(x))} P((x,y))$ тада конструишемо елемент типа $\prod_{(x:A)} \prod_{(y:B(x))} P((x,y))$ као $\lambda x.\lambda y.f((x,y))$. Ова конструкција се назива *каријевање* (енгл. *curry*), и како је супротна правилу $\sum$-ind, правило $\sum$-ind често називамо *инверзно каријевање* (енгл. *uncurry*).



Слика 2.1: Геометријска репрезентација типа зависних парова.

Специјални случај типа зависних парова је тип (независних) *парова* или *(Декартов) производ* $A \times B$. Уколико су типови A и B у контексту Γ, тј. тип

B не зависи од елемената типа A, тада $\sum_{(x:A)} B$ представља тип (независних) парова.

Дефиниција 2.4.4. Тип (независних) *парова* $A \times B$ дефинишемо као:

$$A \times B := \sum_{(x:A)} B.$$

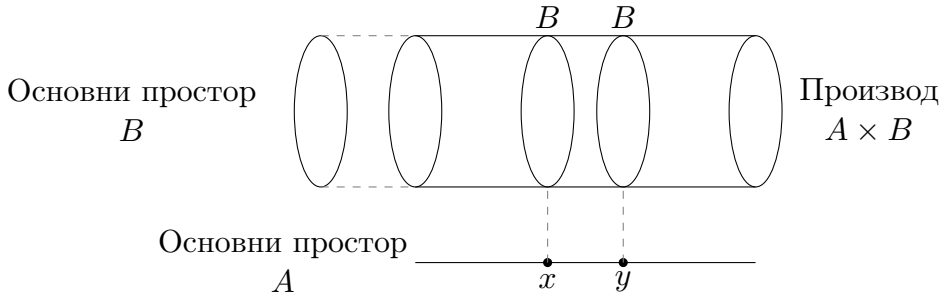
Такође, *пројекцију на први елемент* $\mathsf{fst} : A \times B \to A$ и *пројекцију на други елемент* $\mathsf{snd} : A \times B \to B$ дефинишемо као:

$$\mathsf{fst}((a,b)) :\equiv a, \quad \mathsf{snd}((a,b)) :\equiv b.$$

Правило индукције и израчунавања за тип (независних) парова $A \times B$ директно добијамо из правила индукције и израчунавања за тип зависних парова $\sum_{(x:A)} B(x)$.

$$[\times\text{-ind}] \quad \frac{\begin{array}{l}\Gamma, (x,y) : A \times B \vdash P((x,y)) \text{ type} \\ \Gamma \vdash f : \prod_{(x:A)} \prod_{(y:B)} P((x,y))\end{array}}{\Gamma \vdash \mathsf{ind}_\times(f) : \prod_{(p:A\times B)} P(p)}$$

$$[\times\text{-comp}] \quad \frac{\begin{array}{l}\Gamma, (x,y) : A \times B \vdash P((x,y)) \text{ type} \\ \Gamma \vdash f : \prod_{(x:A)} \prod_{(y:B)} P((x,y))\end{array}}{\Gamma, (x,y) : A \times B \vdash \mathsf{ind}_\times(f, (x,y)) \equiv f(x,y) : P((x,y))}$$



Слика 2.2: Геометријска репрезентација типа независних парова.

Тип независних парова можемо уопштити на тип *k-торки* $A_1 \times A_2 \times \cdots \times A_k$.

Искази	Типови
$\bot$	$\mathbb{0}$
$\top$	$\mathbb{1}$
$\neg A$	$A \to \mathbb{0}$
$A \implies B$	$A \to B$
$A \wedge B$	$A \times B$
$A \vee B$	$\|\|A + B\|\|$
$\forall x.P(x)$	$\prod_{(x:A)} P(x)$
$\exists x.P(x)$	$\|\| \sum_{(x:A)} P(x)\|\|$

Табела 2.1: Интерпретација логике првог реда.

2.5 Искази као типови

Интерпретација *искази као типови* (енгл. *propositions as types*) (често се назива и *Кари-Хауардова коресподенција* [21] или прецизније *Брауверу-Хејтингу-Колмогоров интерпретација* (БХК интепретација)) неформално посматра исказе као типове, доказе као елементе типова, и предикате као фамилије типова [45]. Да би показали да је исказ тачан у теорији типова треба конструисати елемент који настањује одговарајући тип. Прецизније, за дати исказ A (добро-формирани тип) уколико конструишемо елемент $x : A$ (кога често називамо и *сведок* за A) тада сматрамо да је исказ A тачан. Приметимо да исказ није тачан или нетачан, већ да представља колекцију својих сведока који могу да потрврде његову истинитост. Због тога су и сами докази математички објекти. У табели 2.1 приказани су искази заједно са њиховом одговарајућом интерпретацијом у теорији типова.

Прокоментаришимо неке интерпретације из табеле 2.1. Да би показали да важи $A \implies B$ треба претпоставити да важи A и доказати да важи B. У теорији типова треба конструисати елемент типа $A \to B$, тј. треба конструисати елемент типа B који користи претпоставку дату постојањем елемента типа A. Остали типови имају сличне интерпретације сем типа копроизвода $A + B$ и типа зависних парова $\sum_{(x:A)} B(x)$.

Да би показали $A \vee B$ треба показати да важи бар један од A и B. У теорији типова треба конструисати елемент типа $A + B$, помоћу једног од конструктора inl или inr. Због тога тип $A + B$, у односу на $A \vee B$, носи информацију о исказу који је тачан (да ли је тачно A или је тачно B). Слично, да би показали $\exists x.P(x)$ у теорији типова треба конструисати елемент типа

$\sum_{(x:A)} P(x)$. У овом случају теорија типова нам даје и више од тога. Наиме, P је фамилија типова, што значи да $P(x)$ не мора да буде типа $\mathbb{2}$, тј. P не мора да буде предикат. Поред тога, тип $\sum_{(x:A)} P(x)$ можемо схватити као тип свих елемената $x : A$ за које $P(x)$.

Како ова два типа дају више информација у односу на традиционално значење исказа $A \vee B$ и $\exists x.P(X)$, користе се *окрњени искази* (енгл. *propositional truncation*) $||A + B||$ и $||\sum_{(x:A)} B(x)||$ који заборављају све информације о својим сведоцима сем да они постоје. Окрњени искази су ван опсега овог рада, тако да се неће детаљно описивати.

2.6 Хијерархија универзума и универзум типови

Универзум типови се могу посматрати као типови које настањују други типови. Универзум тип $\mathcal{U}$ омогућава да се исказ „A type" запише формално као $A : \mathcal{U}$. Поред тога, омогућава да се фамилија типова B над типом A дефинише као функција $B : A \to \mathcal{U}$.

Желимо да типови који могу да се формирају из празног контекста настањују универзум $\mathcal{U}$ (то су, на пример, $\mathbb{0}$, $\mathbb{1}$, и $\mathbb{N}$). Штавише, како универзум $\mathcal{U}$ настањују и други типови, желимо да универзум $\mathcal{U}$ буде затворен по свим начинима формирања нових типова помоћу типова универзума $\mathcal{U}$. На пример, ако $A : \mathcal{U}$ и $B : A \to \mathcal{U}$, онда $\prod_{(x:A)} B(x) : \mathcal{U}$. Међутим, не сме доћи то тога да универзум настањује сам себе, тј. не сме да важи $\mathcal{U} : \mathcal{U}$. Другим речима, не смемо обезбедити услове настанка *Раселовог парадокса*.

У многим случајевима довољно је постојање једног универзума $\mathcal{U}$, међутим, некада желимо да универзум настањује неки други универзум. Како би избегли Раселов парадокс захтевамо постојање *хијерархије универзума*

$$\mathcal{U}_0, \quad \mathcal{U}_1, \quad \mathcal{U}_2, \quad \ldots \tag{2.3}$$

за коју важе следећа правила:

$$[\mathcal{U}\text{-intro}] \ \frac{}{\Gamma \vdash \mathcal{U}_i : \mathcal{U}_{i+1}} \qquad [\mathcal{U}\text{-cumul}] \ \frac{\Gamma \vdash A : \mathcal{U}_i}{\Gamma \vdash A : \mathcal{U}_{i+1}}$$

Универзум $\mathcal{U}_0$ називамо *базни универзум*. Базни универзум настањују типови који могу да се формирају из празног контекста, као и сви типови за

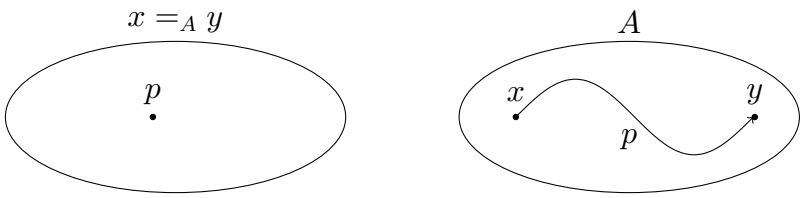
чије се формирање користе типови који се већ налазе у базном универзуму. За универзум $\mathcal{U}_i$ има смисла посматрати и $\mathcal{U}_{i+1}$ кога називамо и *универзум следбеник*. Често није битно знати редни број универзума у хијерархији, те се следбеник универзума $\mathcal{U}$ обележава са $\mathcal{U}^+$. За два универзума $\mathcal{U}$ и $\mathcal{V}$ можемо дефинисати њихову најмању горњу границу $\mathcal{U} \sqcup \mathcal{V}$. На пример, за $\mathcal{U}_0$ і $\mathcal{U}_1$, најмања горња граница $\mathcal{U}_0 \sqcup \mathcal{U}_1$ је $\mathcal{U}_1$.

2.7 Типови идентитета

Подсетимо се да из дефиниције операције $+_{\mathbb{N}}$ важи $m +_{\mathbb{N}} 0_{\mathbb{N}} \equiv m$. Природно се намеће питање: Да ли важи $0_{\mathbb{N}} +_{\mathbb{N}} m \equiv m$? Јасно је да одговор на ово питање треба да буде позитиван, али то није случај у интуиционистичкој теорији типова. Тиме долазимо до фундаменталног проблема интуиционистичке теорије типова: *Шта значи да су елементи неког типа једнаки?* Одговор на ово питање нам дају *типови идентитета* (енгл. *identity types*).

Како расуђивачка једнакост не може описати све врсте једнакости, потребно је дефинисати *исказну једнакост* (енгл. *propositional equality*) која тврди да ће два елемента $x, y : A$ бити исказно једнака. Исказна једнакост је исказ, и по Кари–Хауардовој интерпретацији представља неки тип, а како зависи од два елемента типа A мора бити фамилија типова. Исказне једнакости другачије називамо *типови идентитета*, и обележавамо као $\mathsf{Id}_A : A \to A \to \mathcal{U}$. За два конкретна елемента $x, y : A$, тип $\mathsf{Id}_A(x, y)$ обележавамо и као $x =_A y$ и кажемо да су x и y *једнаки* или *исказно једнаки*.

У хомотопној теорији типова, уколико интерпретирамо тип као простор, и елементе типа као тачке тог простора, онда елементе типа $x =_A y$ можемо интерпретирати као *путање* или *еквиваленције* између тачака x и y у простору A. Као што је могуће да између две тачке у простору постоји више различитих путања, тако је могуће да постоји више од једног сведока једнакости $x =_A y$. Другим речима, $x =_A y$ се може посматрати као тип *идентификације* елемената x и y, и може постојати више начина на који x и y могу да се *идентификују*. На слици 2.7, приказана је геометријска репрезентација ова два тумачења елемената типа идентитета.



Слика 2.3: Геометријска репрезентација типова идентитета. Лево је представљена класична интерпретација, у којој $x =_A y$ представља исказ, са много различитих сведока $p : x =_A y$ који оправдавају његову истинитост. Док је десно представљена хомотопна интерпретација, у којој између тачака x и y постоји много различитих путања p у простору A.

Ако је A тип и ако су дати елементи $x, y : A$ у контексту Γ, онда можемо формирати тип идентитета $x =_A y$ кога ће настањивати путање, еквиваленције или идентификације. Основна идентификација коју можемо да конструишемо је *рефлексија*:

$$\mathsf{refl}_x : x =_A x.$$

Рефлексија refl_x тврди да је било који елемент $x : A$ једнак самом себи. Рефлексију refl_x, у хомотопном смислу, можемо посматрати као константну путању у тачки $x : A$. Формално, начин формирања и конструисања типова идентитета дат је следећом спецификацијом:

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash x : A \qquad \Gamma \vdash y : A}{\Gamma \vdash x =_A y \text{ type}}\ [=\text{-form}] \qquad \frac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash x : A}{\Gamma \vdash \mathsf{refl}_x : x =_A x}\ [=\text{-intro}]$$

Уколико су два елемента $x, y : A$ расуђивачки једнака, тј. важи $x \equiv_A y$, онда су и исказно једнака и важи $\mathsf{refl}_x : x =_A y$. Оправдање проналазимо у $\mathsf{refl}_x : (x =_A x) \equiv (x =_A y)$, јер важи $x \equiv_A y$. Обратно, у општем случају, не важи. Другим речима, уколико су два елемента $x, y : A$ исказно једнака, тј. важи $x =_A y$, у општем случају, неће бити расуђивачки једнака, тј. не важи $x \equiv_A y$. Због тога, расуђивачку једнакост често можемо посматрати као *дефиницону једнакост*.

Индукција путање

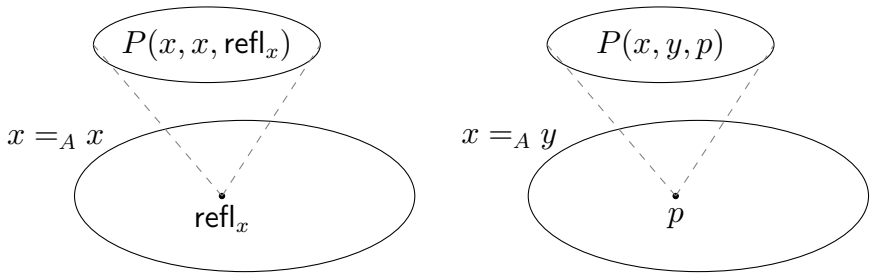
Правило индукције за типове идентитета називамо *индукција путање*. Индукција путање тврди да за било коју фамилију типова P над типовима

A и $x =_A y$ постоји функција $\mathsf{ind}_= : \prod_{(x,y:A)} \prod_{(p:x=_A y)} P(x,y,p)$ уколико постоји функција $f : \prod_{(x:A)} P(x,x,\mathsf{refl}_x)$. Како постоји само један конструктор $\mathsf{refl}_x : x =_A x$, постоји само једно правило израчунавања које треба да се поклопи са правилом индукције. Због тога је правило израчунавања $\mathsf{ind}_=(x,x,\mathsf{refl}_x) \equiv f(x) : P(x,x,\mathsf{refl}_x)$. Формално, правило индукције и правило израчунавања је дато следећом спецификацијом:

$$[=\text{-ind}]\quad \frac{\begin{array}{l}\Gamma, x:A, y:A, p:x =_A y \vdash P(x,y,p)\ \text{type}\\ \Gamma \vdash f : \prod_{(x:A)} P(x,x,\mathsf{refl}_x)\end{array}}{\Gamma \vdash \mathsf{ind}_= : \prod_{(x,y:A)} \prod_{(p:x=_A y)} P(x,y,p)}$$

$$[=\text{-comp}]\quad \frac{\begin{array}{l}\Gamma, x:A, y:A, p:x =_A y \vdash P(x,y,p)\ \text{type}\\ \Gamma \vdash f : \prod_{(x:A)} P(x,x,\mathsf{refl}_x)\end{array}}{\Gamma, x:A \vdash \mathsf{ind}_=(x,x,\mathsf{refl}_x) \equiv f(x) : P(x,x,\mathsf{refl}_x)}$$

Једно од кључних питања је шта оправдава индукцију путањом? Другим речима, зашто ће $P(x,y,p)$ важити за било које тачке $x,y : A$ и било коју путању $p : x =_A y$ уколико важи $P(x,x,\mathsf{refl}_x)$ за било коју тачку $x : A$? Кључно запажање лежи у томе да типови идентитета нису индуктивни тип, већ да су индуктивна фамилија типова. То значи да индукција путањом тврди да је фамилија типова $x =_A y$, где су x и y слободне тачке простора A, индуктивно дефинисана константном путањом refl_x. Односно, $\sum_{(x,y:A)}(x =_A y)$ је индуктивно генерисан константним путањама у свакој тачки $x : A$. Битно је напоменути да су обе тачке слободне. Такође, можемо фиксирати једну тачку (у том случају долазимо до другачије еквивалентне индукције путањом, која се назива и *базна индукција путањом*).



Слика 2.4: Геометријска репрезентација индукције путањом.

Неформално, оправдање индукције путањом је осликано следећим примером: Посматрајмо петљу на пробушеном диску која полази из једне тачке, обилази једном рупу, и враћа се у исту тачку. Уколико су почетна тачка

и крајња тачка фиксиране, није могуће непрекидно деформисати петљу у константну путању. Када допустимо један крај петље да буде слободан, он може обићи рупу, те је могуће непрекидно деформисати петљу у константну путању. Слично је могуће ако допустимо да обе краја петље буду слободна.

Особине типова идентитета

У хомотопној теорији, над тополошким простором A, *путања* између две тачке x и y представља непрекидно пресликавање $p : [0,1] \to A$, за које важи $p(0) = x$ и $p(1) = y$. Путања $p : [0,1] \to A$ је *петља* уколико почиње и завршава се у тачки a_0, тј. важи $a_0 = p(0) = p(1)$. Релевантни појмови над путањама које ћемо користити су инверз путање и надовезивање путања. *Инверз путање* $p : [0,1] \to A$ је путања $p^{-1} : [0,1] \to A$ за коју важи $p^{-1}(t) = p(1-t)$ за свако $t \in [0,1]$. *Надовезивање путања* $p : [0,1] \to A$ и $q : [0,1] \to A$ којима се крај и почетак поклапају, тј. за које $p(1) = q(0)$, представља путању $p \cdot q : [0,1] \to A$ за коју важи $(p \cdot q)(t) = p(2t)$ за свако $t \in [0,1/2]$, и $(p \cdot q)(t) = q(2t-1)$ за свако $t \in [1/2,1]$. Инверз путање и надовезивање путања, у теорији типова, се карактеришу преко типова идентитета и дате су у следећим лемама:

Лема 1. *Нека је A тип у контексту Γ. Тада можемо конструисати функцију*

$$\mathsf{inv}_A : \prod_{(x,y:A)} (x =_A y) \to (y =_A x)$$

индукцијом путање $p : x =_A y$ као $\mathsf{inv}_A(x,x,\mathsf{refl}_x) :\equiv \mathsf{refl}_x$. Функцију inv_A називамо инверз путање. *Често, за дату путању $p : x =_A y$, њен инверз означавамо са $p^{-1} :\equiv \mathsf{inv}_A(x,y,p)$.*

Доказ. Да би констурисали елемент типа $\prod_{(x,y:A)}(x =_A y) \to (y =_A x)$, конструишемо функцију

$$f(x) : \prod_{(y:A)} (x =_A y) \to (y =_A x)$$

за било који елемент $x : A$. По индукцији путање $p : x =_A y$ довољно је конструисати путању

$$f(x,x,\mathsf{refl}_x) : x =_A x$$

за било који елемент $x : A$. Конструкција ове путање је тривијална и због тога узимамо да је $f(x, x, \mathsf{refl}_x) :\equiv \mathsf{refl}_x$. Коначно, тражена конструкција је

$$\mathsf{inv}_A(x, x, \mathsf{refl}_x) :\equiv \mathsf{refl}_x.$$

□

Лема 2. *Нека је A тип у контексту Γ. Тада можемо конструисати функцију*

$$\mathsf{conc}_A : \prod_{(x,y,z:A)} (x =_A y) \to (y =_A z) \to (x =_A z)$$

индукцијом путање $p : x =_A y$ као $\mathsf{conc}_A(x, x, z, \mathsf{refl}_x, q) :\equiv q$. Функцију conc_A називамо надовезивање путања. *Често, за дате путање $p : x =_A y$ и $q : y =_A z$, надовезану путању означавамо са $p \cdot q :\equiv \mathsf{conc}_A(x, y, z, p, q)$.*

Доказ. Прво конструишемо функцију

$$f(x) : \prod_{(y:A)} (x =_A y) \to \prod_{(z:A)} (y =_A z) \to (x =_A z)$$

за било који елемент $x : A$. По индукцији путање $p : (x =_A y)$ довољно је конструисати функцију

$$f(x, x, \mathsf{refl}_x) : \prod_{(z:A)} (x =_A z) \to (x =_A z)$$

за било који елемент $x : A$. Даље, довољно је конструисати функцију

$$f(x, x, \mathsf{refl}_x, z) : (x =_A z) \to (x =_A z)$$

за било које елементе $x, z : A$. Конструисање ове функције је тривијално и због тога имамо да је $f(x, x, \mathsf{refl}_x, z, q) :\equiv q$. Коначно, тражена конструкција је

$$\mathsf{conc}_A(x, x, z, \mathsf{refl}_x, q) :\equiv f(x, x, \mathsf{refl}_x, z, q) :\equiv q.$$

□

Код једнакосне логике, инверз путање представља особину симетрије једнакости, док надовезивање путања представља особину транзитивности једнакости. Транзитивност једнакости се користи када доказујемо неку једнакост ланцем једнакости. На пример, уколико желимо да пронађемо сведока за

$a = d$, и знамо да су $p : a = b$, $q : b = c$ и $r : c = d$, једноставним надовезивањем путања добијамо сведока $(p \cdot q) \cdot r : a = d$. Ово може бити тешко за тумачење па се често користи следећа синтакса:

$$\begin{aligned} a &= b && (p) \\ &= c && (q) \\ &= d && (r). \end{aligned}$$

Строге једнакости над путањама често нису корисне. Због тога, дефинишемо појам хомотопије. *Хомотопија* између две путање $p : [0,1] \to A$ и $q : [0,1] \to A$ је непрекидно пресликавање $H : [0,1] \times [0,1] \to A$, за које важи $H(s,0) = p(s)$ и $H(s,1) = q(s)$ за свако $s \in [0,1]$, и $H(0,t) = x$ и $H(1,t) = y$ за свако $t \in [0,1]$. За две путање кажемо да су *хомотопне*, уколико постоји хомотопија између њих.

Релација хомотопности између две путање је релација еквиваленције, и операције инверз путање и надовезивање путања одржавају еквиваленцију. Над простором A, класа еквиваленција хомотопних петљи у тачки a_0 формира групу коју називамо *фундаментална група* и обележавамо са $\pi_1(A, a_0)$. За разлику од простора свих петљи у тачки a_0, фундаментална група $\pi_1(A, a_0)$ нам омогућава да лакше испитујемо простор A. Прецизније, фундаментална група је алгебарска инваријанта простора, која може да се искористи за испитивање *хомотопне еквиваленције* два простора.

Хомотопију можемо посматрати као дводимензионалну путању, што природно можемо уопштити на тродимензиону путању, четвородимензиону путању, итд. Ово нам даје бесконачни низ појмова: тачка, путања, хомотопија, хомотопија између хомотопија, хомотопија између хомотопија између хомотопија, итд. Заједно са операцијама фундаменталне групе (константна путања, инверз путање, надовезивање путања) добијамо инстанцу алгебарске структуре која се зове (*слаби*) ∞-*групоид*.

∞-групоид представља алгебарску структуру која садржи колекцију *објеката* и колекцију *морфизама* између објеката, *морфизама између морфизама*, итд. Морфизме на нивоу k зовемо k-морфизми. Морфизми сваког нивоа имају комплексну алгебарску структуру, па тако у случају слабог ∞-групоида, сваки морфизам датог нивоа има идентитет, композицију, и инверз који задовољавају закон асоцијативности, идентитета и инверза, до на морфизме следећег нивоа.

Једнакости	Хомотопија	∞-Групоид
рефлексивност	константна путања	идентички морфизам
симетричност	обртање путања	инверз морфизма
транзитивност	надовезивање путања	компоизиција морфизама

Табела 2.2: Разне интерпретације особина типова идентитета

Ову идеју можемо интерпретирати у теорији типова преко типова идентитета. На основном нивоу имамо елементе неког типа које можемо да идентификујемо. На пример, за $x, y : A$, разматрамо идентификацију $x =_A y$. Како је теорија типова доказно релевантна, могуће је да постоји више различитих сведока за $x =_A y$. Нека су то путање $p, q : x =_A y$. Типови идентитета нам даље омогућавају да разматрамо једнакост између p и q, односно омогућавају нам да формирамо тип $p =_{x=_A y} q$. Како су p и q путање, сведока $r : p =_{x=_A y} q$ можемо сматрати хомотопијом између путања p и q, или као дводимензионом путањом. Можемо даље итерирати и добити тип тродимензионих путања $r =_{p=_{x=_A y} q} s$, итд. У табели 2.2 приказана је веза између једнакосне логике, хомотопне теорије и структуре ∞-групоид.

На слици 2.7 приказана је групоидна структура типова, док следећа лема карактерише да тип идентитета на датом нивоу има својства слабог ∞-групоида:

Лема 3. *Нека је A тип, нека су елементи $x, y, z, w : A$ и нека су путање $p : x =_A y$, $q : y =_A z$ и $r : z =_A w$ у контексту Γ. Тада важи:*

(i) $\mathsf{refl}_x \cdot p = p$; *и* $p \cdot \mathsf{refl}_y = p$

(ii) $p^{-1} \cdot p = \mathsf{refl}_y$; *и* $p \cdot p^{-1} = \mathsf{refl}_x$

(iii) $(p^{-1})^{-1} = p$

(iv) $(p \cdot q) \cdot r = p \cdot (q \cdot r)$

(i) *Доказ.* Желимо да конструишемо путању

$$\mathsf{unit_l}(p) : \mathsf{refl}_x \cdot p = p,$$
$$\mathsf{unit_r}(p) : p \cdot \mathsf{refl}_y = p.$$

Индукцијом по путањи $p : x =_A y$ довољно је конструисати

$$\mathsf{unit_l}(\mathsf{refl}_x) : \mathsf{refl}_x \cdot \mathsf{refl}_x = \mathsf{refl}_x,$$
$$\mathsf{unit_r}(\mathsf{refl}_x) : \mathsf{refl}_x \cdot \mathsf{refl}_x = \mathsf{refl}_x.$$

Обе путање је тривијално конструисати као $\mathsf{refl}_{\mathsf{refl}_x}$. □

(ii) *Доказ.* Желимо да конструишемо путању

$$\mathsf{inv_l}(p) : p^{-1} \cdot p = \mathsf{refl}_y,$$
$$\mathsf{inv_r}(p) : p \cdot p^{-1} = \mathsf{refl}_x.$$

Индукцијом по путањи $p : x =_A y$ довољно је конструисати путању

$$\mathsf{inv_l}(\mathsf{refl}_x) : \mathsf{refl}_x^{-1} \cdot \mathsf{refl}_x = \mathsf{refl}_x,$$
$$\mathsf{inv_r}(\mathsf{refl}_x) : \mathsf{refl}_x \cdot \mathsf{refl}_x^{-1} = \mathsf{refl}_x.$$

Али како је $\mathsf{refl}_x^{-1} \equiv \mathsf{refl}_x$ претходне путање се своде на оне као и у претходном доказу. Због тога обе путање тривијално конструишемо као $\mathsf{refl}_{\mathsf{refl}_x}$. □

(iii) *Доказ.* Желимо да конструишемо путању

$$\mathsf{doubleInv}(p) : (p^{-1})^{-1} = p.$$

Индукцијом по путањи $p : x =_A y$ довољно је конструисати путању

$$\mathsf{doubleInv}(\mathsf{refl}_x) : (\mathsf{refl}_x^{-1})^{-1} = \mathsf{refl}_x.$$

Али како је $(\mathsf{refl}_x^{-1})^{-1} \equiv \mathsf{refl}_x^{-1} \equiv \mathsf{refl}_x$ претходна путања се своди на $\mathsf{refl}_x = \mathsf{refl}_x$. Због тога путању тривијално конструишемо као $\mathsf{refl}_{\mathsf{refl}_x}$. □

(iv) *Доказ.* Желимо да конструишемо путању

$$\mathsf{assoc}_A(p, q, r) : (p \cdot q) \cdot r = p \cdot (q \cdot r).$$

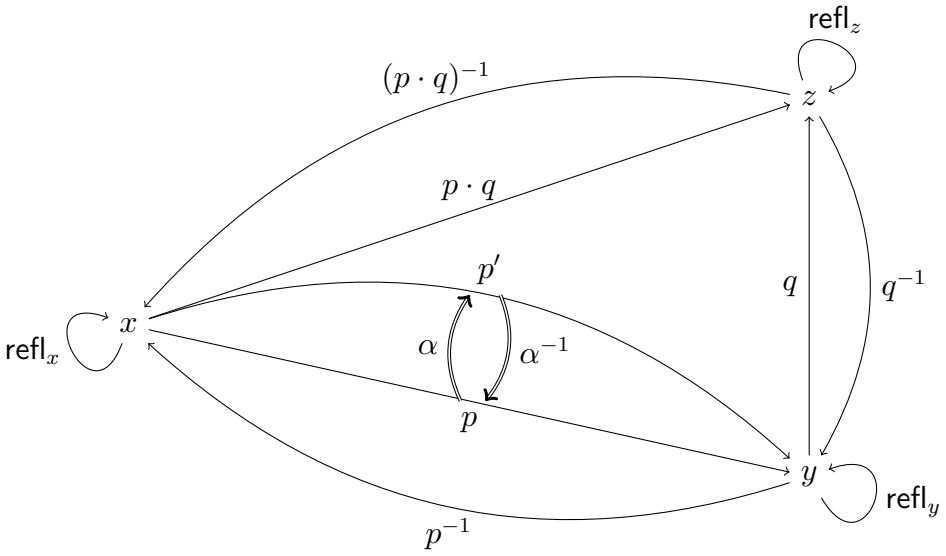
Индукцијом по путањи $p : x =_A y$ довољно је конструисати путању

$$\mathsf{assoc}_A(\mathsf{refl}_x, q, r) : (\mathsf{refl}_x \cdot q) \cdot r = \mathsf{refl}_x \cdot (q \cdot r)$$

Али како је $\mathsf{refl}_x \cdot q \equiv q$ и $\mathsf{refl}_x \cdot (q \cdot r) \equiv q \cdot r$ претходна путања се своди на

$$\mathsf{assoc}_A(\mathsf{refl}_x, q, r) : q \cdot r = q \cdot r.$$

Због тога путању тривијално конструишемо као $\mathsf{assoc}_A(\mathsf{refl}_x, q, r) :\equiv \mathsf{refl}_{q \cdot r}$. □



Слика 2.5: Групоидна структура типова.

Акције над путањама

Функција $f : A \to B$ се понаша као функтор у односу на путање. Другим речима, функције поштују једнакости, или функције одржавају путање. Ова особина дата је следећом лемом:

Лема 4. *Нека су A и B типови, и нека је $f : A \to B$ функција у контексту Γ. Тада можемо конструисати функцију*

$$\mathsf{ap}_f : \prod_{(x,y:A)} (x =_A y) \to (f(x) =_B f(y))$$

индукцијом путање $p : x =_A y$ као $\mathsf{ap}_f(\mathsf{refl}_x) = \mathsf{refl}_{f(x)}$. Функцију ap_f називамо акција над путањама *функције $f : A \to B$.*

Доказ. Индукцијом по путањи $p : x =_A y$ треба конструисати путању

$$\mathsf{ap}_f(x, x, \mathsf{refl}_x) : f(x) =_B f(x).$$

Тривијално конструишемо ову путању као $\mathsf{ap}_f(x, x, \mathsf{refl}_x) :\equiv \mathsf{refl}_{f(x)}$. □

Важе уобичајене особине за функторе:

Лема 5. *Нека су A, B и C типови, нека су елементи $x, y, z : A$ и нека су путање $p : x =_A y$ и $q : y =_A z$ у контексту Γ. Тада важи:*

(i) $\mathsf{ap}_f(p \cdot q) = \mathsf{ap}_f(p) \cdot \mathsf{ap}_f(q)$

(ii) $\mathsf{ap}_f(p^{-1}) = \mathsf{ap}_f(p)^{-1}$

(iii) $\mathsf{ap}_g(\mathsf{ap}_f(p)) = \mathsf{ap}_{g \circ f}(p)$

(iv) $\mathsf{ap}_{\mathsf{id}_A}(p) = p$

Доказ. Доказ изостављамо како је сличан претходним. □

Транспорт

Уколико посматрамо функцију $f : \prod_{(x:A)} B(x)$ и путању $p : x =_A y$, онда не можемо разматрати једнакост елемената $f(x) : B(x)$ и $f(y) : B(y)$, јер нису истог типа. Али можемо разматрати функцију $B(x) \to B(y)$, коју карактерише следећа лема:

Лема 6. *Нека је A тип и B фамилија типова над A у контексту Γ. Тада можемо конструисати функцију*

$$\mathsf{tr}_B : \prod_{(x,y:A)} (x =_A y) \to B(x) \to B(y)$$

индукцијом по путање $p : x =_A y$ као $\mathsf{tr}_B(\mathsf{refl}_x) :\equiv \mathsf{id}_{B(x)}$. Функцију tr_B називамо транспорт *над B.*

Доказ. Индукцијом по путањи $p : x =_A y$ треба конструисати функцију

$$\mathsf{tr}_B(x, x, \mathsf{refl}_x) \to B(x) \to B(x).$$

Тривијално конструишемо ову путању као $\mathsf{tr}_B(x, x, \mathsf{refl}_x) :\equiv \mathsf{id}_{B(x)}$. □

Фамилија типова B над типом A, може се сматрати као својство елемената типа A. Другим речима, $B(x)$ је својство елемента $x : A$. Због тога и претходне леме, можемо тврдити да својства поштују једнакости, у смислу да ако је $x =_A y$, онда $B(x)$ акко $B(y)$.

Друге врсте једнакости

Некада расуђивачка и исказна једнакост нису довољне да би се показале све тврдње. На пример, не можемо пронаћи сведоке за $\neg(\mathsf{succ}_{\mathbb{N}}(n) =_{\mathbb{N}} 0_{\mathbb{N}})$ и $(\mathsf{succ}_{\mathbb{N}}(m) =_{\mathbb{N}} \mathsf{succ}_{\mathbb{N}}(n)) \to m =_{\mathbb{N}} n$ без следеће карактеризације:

Дефиниција 2.7.1. *Простор кодова* над природним бројевима $\mathbb{N}$ се може дефинисати као бинарна релација $\mathsf{code}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N} \to \mathcal{U}_0$ тако да задовољава следеће расуђивачке једнакости:

$$\begin{aligned}
\mathsf{code}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) &\equiv \mathbb{1} \\
\mathsf{code}_{\mathbb{N}}(0_{\mathbb{N}}, \mathsf{succ}_{\mathbb{N}}(m)) &\equiv \mathbb{0} \\
\mathsf{code}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(n), 0_{\mathbb{N}}) &\equiv \mathbb{0} \\
\mathsf{code}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(n), \mathsf{succ}_{\mathbb{N}}(m)) &\equiv \mathsf{code}_{\mathbb{N}}(n, m)
\end{aligned}$$

Лема 7. *Простор кодова је рефлексивна релација, тј. можемо конструисати функцију*

$$\mathsf{reflcode}_{\mathbb{N}} : \prod_{(n:\mathbb{N})} \mathsf{code}_{\mathbb{N}}(n, n).$$

Доказ. Функцију конструишемо индукцијом по $n : \mathbb{N}$ као

$$\begin{aligned}
\mathsf{reflcode}_{\mathbb{N}}(0_{\mathbb{N}}) &:\equiv \star \\
\mathsf{reflcode}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(n)) &:\equiv \mathsf{reflcode}_{\mathbb{N}}(n).
\end{aligned}$$

□

Лема 8. *За било које природне бројеве $n, m : \mathbb{N}$ важи $m =_{\mathbb{N}} n \to \mathsf{code}_{\mathbb{N}}(m, n)$ и $\mathsf{code}_{\mathbb{N}}(m, n) \to m =_{\mathbb{N}} n$.*

Доказ. Прво конструишемо

$$\mathsf{encode}_{\mathbb{N}} : \prod_{(m,n:\mathbb{N})} m =_{\mathbb{N}} n \to \mathsf{code}_{\mathbb{N}}(m, n).$$

Индукцијом по путањи $p : m =_{\mathbb{N}} n$ треба конструисати

$$\mathsf{encode}_{\mathbb{N}}(m, m, \mathsf{refl}_m) : \mathsf{code}_{\mathbb{N}}(m, m).$$

Што смо конструисали у претходној леми, тако да $\mathsf{encode}_{\mathbb{N}}(m, m, \mathsf{refl}_m) :\equiv \mathsf{reflcode}_{\mathbb{N}}(m)$.

Други начин да конструишемо функцију $\mathsf{encode}_{\mathbb{N}}$ је преко транспорта, као $\mathsf{encode}_{\mathbb{N}}(m, n, p) :\equiv \mathsf{tr}_{\mathsf{code}_{\mathbb{N}}(m)}(p, \mathsf{reflcode}_{\mathbb{N}}(m))$.

Даље конструишемо

$$\mathsf{decode}_{\mathbb{N}} : \prod_{(m,n:\mathbb{N})} \mathsf{code}_{\mathbb{N}}(m, n) \to m =_{\mathbb{N}} n$$

индукцијом по $m : \mathbb{N}$ и $n : \mathbb{N}$. У случају када су оба природна броја нуле, онда $\mathsf{decode}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}, c) : 0_{\mathbb{N}} =_{\mathbb{N}} 0_{\mathbb{N}}$ конструишемо као $\mathsf{decode}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}, c) :\equiv \mathsf{refl}_{0_{\mathbb{N}}}$. У случају када је тачно један од њих нула, тада конструишемо елемент типа $\mathbb{0} \to m =_{\mathbb{N}} n$. Овај елемент је тривијално конструисати правилом индукције празног типа. На крају, у случају када су оба различита од нуле, треба конструисати

$$\mathsf{code}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m), \mathsf{succ}_{\mathbb{N}}(n)) \to \mathsf{succ}_{\mathbb{N}}(m) =_{\mathbb{N}} \mathsf{succ}_{\mathbb{N}}(n).$$

Ову конструкцију изводимо на следећи начин:

$$\begin{aligned}
\mathsf{code}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m), \mathsf{succ}_{\mathbb{N}}(n)) &\equiv \mathsf{code}_{\mathbb{N}}(m, n) && (\text{деф. } 2.7.1)\\
&\to m =_{\mathbb{N}} n && (\mathsf{decode}_{\mathbb{N}}(m, n))\\
&\to \mathsf{succ}_{\mathbb{N}}(m) =_{\mathbb{N}} \mathsf{succ}_{\mathbb{N}}(n) && (\mathsf{ap}_{\mathsf{succ}_{\mathbb{N}}}).
\end{aligned}$$

Коначно, завршавамо конструкцију са

$$\mathsf{decode}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(m), \mathsf{succ}_{\mathbb{N}}(n), c) :\equiv \mathsf{ap}_{\mathsf{succ}_{\mathbb{N}}}(\mathsf{decode}_{\mathbb{N}}(m, n, c)).$$

□

Глава 3

Agda

Типски зависни програмски језици представљају фамилију (углавном) чисто функционалних програмских језика чији систем типова подржава концепт зависних типова. Неки од њих су: Agda [32, 31], Lean4 [28] и Idris [7]. Са друге стране, постоје и функционални програмски језици, на пример Haskell, чији систем типова на подржава концепт зависних типова. Типски зависни програмски језици се могу користити као интерактивни доказивачи теорема. Такође, постоје интерактивни доказивачи теорема који подржавају концепт зависних типова, али се не сматрају за функционалне програмске језике, као што су Coq [37] и NuPRL [14]. Наравно, постоји и интерактивни доказивачи теорема, као што је Isabell/HOL [30], чији систем типова не подржава зависне типове. Комплетна класификација система типова дата је кроз *ламбда коцку* [3].

Agda је типски завистан програмски језик који је базиран на МЛТТ-у. Језик Agda подсећа на функционални језик Haskell, али га проширује зависним типовима. Штавише, језик Agda је развијен у језику Haskell, и подржава трансформисање Agda изворног кода у Haskell изворни кôд. Са друге стране, Agda је интерактивни доказивач теорема. Предност језика Agda у односу на друге интерактивне доказиваче теорема је минималност, и високи степен модуларности и изражајности. Минималности језика Agda се огледа у недостатку напредних могућности које пружају други интерактивни доказивачи теорема. Наиме, други интерактивни доказивачи теорема су богати *тактикама*, алгоритмима за аутоматско доказивање. Тактике чини интерактивне доказиваче теорема моћним, али у исто време скривају делове доказа. Високи степен модуларности и изражајности омогућава јасно дефи-

нисање математичких објеката описивањем синтаксе не само за дефинисане објекте, већ и за структуре доказа. Agda изворни кöд подржава UNICODE карактере, што омогућава лакше записивање математичких симбола и конструкције. Све то чини језик Agda идеалним за формализовање МЛТТ-а, па самим тим и XoTT-а.

3.1 Основе језика Agda

Agda програми се развијају интерактивно, што значи да је могуће верификовати ограничења типова (енгл. *type check*) иако кöд није комплетно написан. Програм који извршава верификацију ограничења типова назива се *ūпроверавач ūшūова* (енгл. *typechecker*). Верификовање ограничења типова, се може интерпретирати као провера исправности доказа. Agda изворни кöд може садржати *ūразнине* које се могу допунити касније. Agda празнине интерпретира као изразе одговарајућих типова. Односно, празнине ће успешно проћи верификацију проверавача типова. Многи текстуални едитори (помоћу одговарајуће екстензије за језик Agda) подржавају интерактивни развој Agda програма: Emacs, Vim, Atom, Visual Studio Code.

Када се интерактивно развија Agda програм, у неком од наведених текстуалних едитора, користе се *команде* које пружају информације од проверивача типова, као и начине на које можемо обрађивати празнине. Неке од корисних команди су:

`C-c C-l`: Учитава фајл и покреће проверавач типова. Свако појављивање знака `?` замењује са новом празнином.

`C-c C-d`: Одређује тип датог израза.

`C-c C-n`: Нормализује дати израз.

`C-c C-f`: Поставља курсор на прву следећу празнину.

`C-c C-b`: Поставља курсор на прву претходну празнину.

`C-c C-,`: Приказује очекивани тип празнине на којој се налази курсор, као и типове свих променљивих из контекста.

`C-c C-c`: Раздваја задату променљиву на све могуће случајеве, односно на све могуће конструкторе.

`C-c C-SPC`: Замењује празнину датим изразом, уколико је дати израз одговарајућег типа.

`C-c C-r`: Прерађује празнину тако што је замени датим изразом, уз потенцијално отварање нових празнина за сваки од аргумената датог израза.

`C-c C-a`: Аутоматски покушава да пронађе одговарајући израз за празнину на којој се налази курсор.

`C-x C-c`: Преводи програм.

3.2 Пример интеракције у језику Agda

Прођимо кроз један основни пример формализације типа природних бројева. Како се име сваког Agda фајла завршава екстензијом `.agda`, нека се наш фајл зове `Tutorial.agda`.

На почетку фајла описујемо неке мета-податке:

```
{-# OPTIONS -without-K -safe #-}
```

У овом случају то су опције `-without-K` и `-safe`, које обезбеђују рад на XoTT-у. Више детаља се може наћи на званичној документацији језика Agda [36].

Agda програм је сачињен од *модула*. Сваки модул има своје име (које мора да се поклопи са именом фајла) као и низ декларација. Модул за наш фајл `Tutorial.agda` започињемо на следећи начин:

```
module Tutorial where
```

Једна врста декларација је увожење других модула. Тако, на пример, можемо увести модул `Universes` на следећи начин:

```
open import Universes public
```

Друга врста декларација је декларисање променљивих. На пример, можемо рећи да $\mathcal{U}$, $\mathcal{V}$ и $\mathcal{W}$ представљају универзум:

```
variable 𝒰 𝒱 𝒲 : Universe
```

Једна од кључних декларација је декларација индуктивних типова. Декларисање индуктивних типова подразумева навођење имена тог типа, начин

на који се он формира, као и начин за конструисање каноничних елемената тог типа. У теорији типова, то смо називали правило формирања типова и правила конструисања (за детаље видети поглавље 2.4). Подсетимо се правила формирања и правила конструисања типа природних бројева $\mathbb{N}$. Тип природних бројева $\mathbb{N}$ може да се формира из празног контекста, односно $\mathbb{N} : \mathcal{U}_0$, као и да су му једини конструктори $0_{\mathbb{N}} : \mathbb{N}$ и $\mathsf{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$. У језику AGDA то записујемо као:

```
data ℕ : 𝒰₀ · where
  zero : ℕ
  succ : ℕ → ℕ

{-# BUILTIN NATURAL ℕ #-}
```

Опција `BUILTIN NATURAL` $\mathbb{N}$ обезбеђује да канонične елементе типа природних бројева $0_{\mathbb{N}}, \mathsf{succ}_{\mathbb{N}}(0_{\mathbb{N}}), \mathsf{succ}_{\mathbb{N}}(\mathsf{succ}_{\mathbb{N}}(0_{\mathbb{N}})), \ldots$ можемо да записујемо као $0, 1, 2, \ldots$.

Поред правила формирања и правила конструисања, за комплетну спецификацију индуктивних типова имали смо и правило индукције и правило израчунавања (за детаље видети поглавље 2.4). Правило индукције типа природних бројева $\mathbb{N}$, у језику AGDA записујемо као:

```
ℕ-induction : (P : ℕ → 𝒰₀ · )
            → P 0
            → ((n : ℕ) → P n → P (succ n))
            → (n : ℕ) → P n
```

Приметимо да у формулацији правила индукције фамилију типова P над типом природних бројева $\mathbb{N}$ записујемо као $P : \mathbb{N} \to \mathcal{U}_0$, као и да зависну функцију $\prod_{(n:\mathbb{N})} P(n)$ записујемо као $(n : \mathbb{N}) \to P\ n$.

Са друге стране, правила израчунавања записујемо као:

```
ℕ-induction P p₀ pₛ zero = p₀
ℕ-induction P p₀ pₛ (succ n) = pₛ n (ℕ-induction P p₀ pₛ n)
```

У многим случајевима AGDA може помоћи при декларисању правила израчунавања. Како она описују начин конструисања елемента који оправдава правило индукције можемо поћи од:

```
ℕ-induction P p₀ pₛ n = {!   !}
```

Командом `C-c C-c` раздвајамо n на случајеве:

$$\mathbb{N}\text{-induction } P\ p_o\ p_s\ \mathsf{zero} = \{!\ \ !\}$$
$$\mathbb{N}\text{-induction } P\ p_o\ p_s\ (\mathsf{succ}\ n) = \{!\ \ !\}$$

Командом `C-c C-a` аутоматски решавамо први случај, док на други примењујемо `C-c C-r` над изразом p_s:

$$\mathbb{N}\text{-induction } P\ p_o\ p_s\ \mathsf{zero} = p_o$$
$$\mathbb{N}\text{-induction } P\ p_o\ p_s\ (\mathsf{succ}\ n) = p_s\ \{!\ \ !\}\ \{!\ \ !\}$$

На првом пољу, командом `C-c C-,`, сазнајемо да је потребно конструисати тип $\mathbb{N}$, као и да је у контексту $n : \mathbb{N}$. Због тога, једноставно, попуњавамо поље са n командом `C-c C-SPC`. На другом пољу, командом `C-c C-,`, сазнајемо да је потребно конструисати тип $P(n)$ за $n : \mathbb{N}$. Приметимо да је $\mathbb{N}$-induction функција која враћа $P(n)$, те примењујемо команду `C-c C-r` над $\mathbb{N}$-induction P.

$$\mathbb{N}\text{-induction } P\ p_o\ p_s\ \mathsf{zero} = p_o$$
$$\mathbb{N}\text{-induction } P\ p_o\ p_s\ (\mathsf{succ}\ n) = p_s\ n\ (\mathbb{N}\text{-induction } P\ \{!\ \ !\}\ \{!\ \ !\}\ \{!\ \ !\})$$

Преостала поља можемо једноставно попунити аутоматски помоћу команде `C-c C-a`, или разматрањем траженог типа и контекста помоћу команде `C-c C-,`, а онда уписивањем одговарајућих израза помоћу команде `C-c C-SPC`. Тиме успешно завршавамо конструкцију:

$$\mathbb{N}\text{-induction } P\ p_o\ p_s\ \mathsf{zero} = p_o$$
$$\mathbb{N}\text{-induction } P\ p_o\ p_s\ (\mathsf{succ}\ n) = p_s\ n\ (\mathbb{N}\text{-induction } P\ p_o\ p_s\ n)$$

Глава 4

Библиотека InTT

Библиотека InTT, скраћено од Intucionistic Type Theory, чини формализацију основних појмова МЛТТ-а унутар типски зависног програмског језика Agda. Библиотеку InTT одликују минималност, концизност и проширивост. Што је чини идеалном за истраживање у оквирима МЛТТ-а. Са друге стране, лако се може проширити и на друге теорије типова у чијој је основи МЛТТ, као што је на пример и XoTT.

Комплетна библиотека је јавно доступна и може се наћи на следећој адреси: `https://github.com/andrija-urosevic/InTT`.

4.1 Садржај библиотеке InTT

Библиотека InTT укључује:

- Основне индуктивне типове: празан тип $\mathbb{0}$, јединични тип $\mathbb{1}$, тип копроизвода $A + B$, тип зависних парова $\sum_{(x:A)} B(x)$, тип парова $A \times B$, тип зависних функција $\prod_{(x:A)} B(x)$, Булов тип $\mathbb{2}$, тип природних бројева $\mathbb{N}$, тип целих бројева $\mathbb{Z}$, тип листи List_A и коначни тип Fin_n.
- Типови идентитета: индукција путањом, инверз путање, надовезивање путања, особине ∞-групоида, акција над путањом, транспорт путање.
- Операције, релације и особине дефинисаних индуктивних типова.
- Својства одлучивости дефинисаних индуктивних типова.
- Синтакса универзум типова.

4.2 Имплементација библиотеке InTT

У овом поглављу ће бити описани имплементациони детаљи одабраних делова библиотеке InTT. Њихова теоријска подлога је већ разматрана у оквиру главе 2.

Тип зависних функција

Тип зависних функција као такав постоји у језику Agda и није га потребно дефинисати. За фамилију типова P над типом X, тип зависних функција $\prod_{(x:X)} P(x)$ у језику Agda записујемо као $(x : X) \to P\ x$. Са друге стране, Agda је веома изражајна и омогућава нам да дефинишемо синтаксу на коју смо навикли:

```
∏ : {X : 𝒰 ·} (P : X → 𝒱 ·) → 𝒰 ⊔ 𝒱 ·
∏ {𝒰} {𝒱} {X} P = (x : X) → P x

-∏ : {𝒰 𝒱 : Universe} (X : 𝒰 ·) (Y : X → 𝒱 ·) → 𝒰 ⊔ 𝒱 ·
-∏ X Y = ∏ Y

syntax -∏ A (λ x → b) = ∏ x : A , b
```

Ова синтакса се веома ретко користи у библиотеци јер је подразумевани начин записивања зависних функција једноставнији.

На основу дефиниције идентитета 2.3.2 можемо дефинисати следећу функцију:

```
id : {X : 𝒰 ·} → X → X
id x = x
```

На основу дефиниције композиције функција 2.3.3 можемо дефинисати следећу функцију:

```
_∘_ : {X : 𝒰 ·} {Y : 𝒱 ·} {Z : Y → 𝒲 ·}
      → (g : (y : Y) → Z y)
      → (f : X → Y)
      → (x : X) → Z (f x)
(g ∘ f) x = g (f x)
```

Празан тип

На основу правила $\mathbb{0}$-form празан тип можемо формирати из празног контекста, односно $\mathbb{0} : \mathcal{U}_0$. Поред тога, празан тип нема ни један конструктор. У језику Agda то записујемо као:

```
data 𝟘 : 𝒰₀ · where
```

Правило индукције $\mathbb{0}$-ind заједно са недостатком правила израчунавања у језику Agda записујемо као:

```
𝟘-induction : (P : 𝟘 → 𝒰 · ) → (x : 𝟘) → P x
𝟘-induction P ()
```

Приметимо да овде () означава да празан тип нема конструкторе.

Правило рекурзије $\mathbb{0}$-rec уводимо преко правила индукције:

```
𝟘-recursion : (A : 𝒰 · ) → 𝟘 → A
𝟘-recursion A p = 𝟘-induction (λ _ → A) p

!𝟘 : {A : 𝒰 · } → 𝟘 → A
!𝟘 {𝒰} {A} = 𝟘-recursion A
```

Приметимо да израз $(\lambda \_ \to A)$ означава да тип A не зависи од $x : \mathbb{0}$, односно да тип A није фамилија типова. Функција $!\mathbb{0}$ је само облик рекурзије који имплицитно закључује тип A.

Дефиницију празног типа и негације 2.4.1, као и дефиницију дупле негације у језику Agda записујемо као:

```
empty : 𝒰 · → 𝒰 ·
empty X = X → 𝟘

¬ : 𝒰 · → 𝒰 ·
¬ X = X → 𝟘

¬¬ : 𝒰 · → 𝒰 ·
¬¬ X = ¬ (¬ X)
```

Дупла негација се понаша као функтор, односно можемо конструисати елемент $\neg\neg$-functor : $(X \to Y) \to (\neg\neg X \to \neg\neg Y)$. Да би то постигли, конструишимо прво елемент ¬-inv-functor : $(X \to Y) \to (\neg Y \to \neg X)$. Искористићемо празнине и интерактивно својство језика Agda:

```
¬-inv-functor : {X : 𝒰 ·} {Y : 𝒱 ·} → (X → Y) → (¬ Y → ¬ X)
¬-inv-functor f ny x = {!   !}
```

Тренутно треба конструисати елемент тип $\mathbb{0}$, а у контексту се налази $x : X, ny : \neg Y$ и $f : X \to Y$. Због тога примењујемо функцију $ny : \neg Y :\equiv Y \to \mathbb{0}$.

```
¬-inv-functor f ny x = ny {!   !}
```

Тренутни циљ постаје конструисање елемента типа Y, за исти контекст. Због тога, примењујемо функцију $f : X \to Y$.

```
¬-inv-functor f ny x = ny (f {!   !})
```

На крају, треба конструисати елемент типа X, за исти контекст. Како се у контексту налази одговарајућа конструкција, тј. $x : X$ је у контексту, конструкцију тривијално завршавамо:

```
¬-inv-functor f ny x = ny (f x)
```

Тражену конструкцију о функториалности дупле негације извршавамо интерактивно на сличан начин.

```
¬¬-functor : {X : 𝒰 ·} {Y : 𝒱 ·} → (X → Y) → (¬¬ X → ¬¬ Y)
¬¬-functor f nnx ny = nnx (¬-inv-functor f ny)
```

Јединични тип

На основу правила $\mathbb{1}$-form јединични тип можемо формирати из празног контекста, односно $\mathbb{1} : \mathcal{U}_0$. Поред тога, на основу правила $\mathbb{1}$-intro$_\star$ јединични тип има један конструктор $\star : \mathbb{1}$. У језику Agda то записујемо као:

```
data 𝟙 : 𝒰₀ · where
  ⋆ : 𝟙
```

Правило индукције $\mathbb{1}$-ind заједно са правилом израчунавања $\mathbb{1}$-comp, као и правило рекурзије $\mathbb{1}$-rec у језику Agda записујемо као:

```
𝟙-induction : (P : 𝟙 → 𝒰 ·) → P ⋆ → (x : 𝟙) → P x
𝟙-induction P p ⋆ = p
```

```
𝟙-recursion : (A : 𝒰 ̇ ) → A → 𝟙 → A
𝟙-recursion A = 𝟙-induction (λ _ → A)
```

Дефиницију јединствене функције 2.4.2 у језику Agda записујемо као:

```
!𝟙 : {A : 𝒰 ̇ } → A → 𝟙
!𝟙 a = ⋆
```

Тип копроизвода

На основу правила +-form тип копроизвода можемо формирати из контекста у коме су X и Y типови. Односно, ако $X : \mathcal{U}$ и $Y : \mathcal{V}$, онда $X + Y : \mathcal{U} \sqcup \mathcal{V}$. Поред тога, на основу правила $+\text{-intro}_{\mathsf{inl}}$ и $+\text{-intro}_{\mathsf{inr}}$ јединични тип има два конструктора $\mathsf{inl} : X \to X + Y$ и $\mathsf{inr} : Y \to X + Y$. У језику Agda то записујемо као:

```
data _+_ (X : 𝒰 ̇ ) (Y : 𝒱 ̇ ) : 𝒰 ⊔ 𝒱 ̇ where
  inl : X → X + Y
  inr : Y → X + Y
```

Правило индукције +-ind заједно са правилима израчунавања $+\text{-comp}_{\mathsf{inl}}$ и $+\text{-comp}_{\mathsf{inr}}$, као и правило рекурзије +-rec у језику Agda записујемо као:

```
+-induction : {X : 𝒰 ̇ } {Y : 𝒰 ̇ } (P : X + Y → 𝒰 ̇ )
            → ((x : X) → P (inl x))
            → ((y : Y) → P (inr y))
            → (z : X + Y) → P z
+-induction P pl pr (inl x) = pl x
+-induction P pl pr (inr y) = pr y

+-recursion : {X : 𝒰 ̇ } {Y : 𝒰 ̇ } (A : 𝒰 ̇ )
            → (X → A)
            → (Y → A)
            → X + Y → A
+-recursion A f g (inl x) = f x
+-recursion A f g (inr x) = g x
```

За функције $f : A \to X$ и $g : B \to Y$ можемо конструисати елемент функције копроизвод $A + B \to X + Y$. У језику Agda то записујемо као:

```
_+→_ : {A X : 𝒰 ·} {B Y : 𝒰 ·} (f : A → X) (g : B → Y)
       → (A + B) → (X + Y)
(f +→ g) (inl x) = inl (f x)
(f +→ g) (inr x) = inr (g x)
```

Даље су приказана својства о празном типу и типу копроизвода:

```
+-empty : {A : 𝒰 ·} {B : 𝒱 ·}
          → ¬ A → ¬ B → ¬ (A + B)
+-empty f g (inl a) = f a
+-empty f g (inr b) = g b

+-left-empty : {X : 𝒰 ·} {Y : 𝒰 ·}
               → ¬ X → X + Y → Y
+-left-empty {𝒰} {X} {Y} ex = +-recursion Y (!𝟘 ∘ ex) id

+-right-empty : {X : 𝒰 ·} {Y : 𝒰 ·}
                → ¬ Y → X + Y → X
+-right-empty {𝒰} {X} {Y} ey = +-recursion X id (!𝟘 ∘ ey)
```

Тип зависних парова

На основу правила $\sum$-form тип зависних парова можемо формирати из контекста у коме је X тип и Y фамилија типови над типом X. Односно, ако $X : \mathcal{U}$ и $Y : X \to \mathcal{V}$, онда $\sum_{(x:X)} Y(x) : \mathcal{U} \sqcup \mathcal{V}$. Поред тога, на основу правила $\sum$-intro тип зависних парова има један конструктора $(x, y(x)) : \sum_{(x:X)} Y(x)$. У језику Agda то записујемо као:

```
record ∑ {𝒰 𝒱} {X : 𝒰 ·} (Y : X → 𝒱 ·) : 𝒰 ⊔ 𝒱 · where
  constructor

    _,_
  field
    x : X
    y : Y x
```

Дефиницију пројекција на први и други елемент 2.4.3 у језику Agda записујемо као:

```
fst : {X : 𝒰 ·} {Y : X → 𝒱 ·}
    → ∑ Y → X
fst (x , y) = x

snd : {X : 𝒰 ·} {Y : X → 𝒱 ·}
    → (z : ∑ Y) → Y (fst z)
snd (x , y) = y
```

Правило индукције ∑-ind заједно са правилима израчунавања ∑-comp, као и поступак каријевања и инверзног каријевања у језику Agda записујемо као:

```
∑-induction : {X : 𝒰 ·} {Y : X → 𝒱 ·} {P : ∑ Y → 𝒲 ·}
            → ((x : X) (y : Y x) → P (x , y))
            → ((x , y) : ∑ Y) → P (x , y)
∑-induction f (x , y) = f x y

curry : {X : 𝒰 ·} {Y : X → 𝒱 ·} {P : ∑ Y → 𝒲 ·}
      → (((x , y) : ∑ Y) → P (x , y))
      → ((x : X) (y : Y x) → P (x , y))
curry f x y = f (x , y)

uncurry : {X : 𝒰 ·} {Y : X → 𝒱 ·} {P : ∑ Y → 𝒲 ·}
        → ((x : X) (y : Y x) → P (x , y))
        → ((x , y) : ∑ Y) → P (x , y)
uncurry f (x , y) = ∑-induction f (x , y)
```

Слично као и за тип зависних функција, за тип зависних парова можемо увести синтаксу на коју смо навикли:

```
-∑ : {𝒰 𝒱 : Universe} (X : 𝒰 ·) (Y : X → 𝒱 ·) → 𝒰 ⊔ 𝒱 ·
-∑ X Y = ∑ Y

syntax -∑ X (λ x → y) = ∑ x : X , y
```

На основу типа зависних парова, можемо дефинисати тип (независних) парова или Декартов производ $X \times Y$. Његово правило индукције $\times$-ind

и правило израчунавања ×-comp, директно следи из правила индукције и правила израчунавања типа зависних парова. У језику Agda то записујемо као:

```
_×_ : 𝒰 ̇ → 𝒱 ̇ → 𝒰 ⊔ 𝒱 ̇
X × Y = Σ x ꞉ X , Y

×-induction : {X : 𝒰 ̇ } {Y : 𝒱 ̇ } {P : X × Y → 𝒲 ̇ }
            → ((x : X) (y : Y) → P (x , y))
            → ((x , y) : X × Y) → P (x , y)
×-induction f (x , y) = f x y
```

Корисно је увести и следећи пар функција:

```
_↔_ : 𝒰 ̇ → 𝒱 ̇ → 𝒰 ⊔ 𝒱 ̇
X ↔ Y = (X → Y) × (Y → X)
```

Типови идентитета

На основу правила =-form тип идентитета можемо формирати из контекста у коме је X тип и у коме су $x : X$ и $y : X$ неки елементи. Односно, ако $X : \mathcal{U}$, онда $\mathsf{Id}_X : X \to X \to \mathcal{U}$. Поред тога, на основу правила =-intro тип зависних парова има један конструктора $\mathsf{refl}_X : X \to \mathsf{Id}_X$. У језику Agda то записујемо као:

```
data Id {𝒰} (X : 𝒰 ̇ ) : X → X → 𝒰 ̇ where
  refl : (x : X) → Id X x x
```

Такође, уводимо и синтаксу $x =_X y$ за тип идентитета $\mathsf{Id}_X(x, y)$ у којој се тип X закључује имплицитно.

```
infixl 10 _==_

_==_ : {X : 𝒰 ̇ } → X → X → 𝒰 ̇
x == y = Id _ x y
```

Индукцију путање =-ind заједно са правилом израчунавања =-comp у језику Agda записујемо као:

```
𝕁 : (X : 𝒰 ·) (P : (x y : X) → x == y → 𝒱 ·)
  → ((x : X) → P x x (refl x))
  → ((x y : X) (p : x == y) → P x y p)
𝕁 X P f x y (refl x) = f x
```

Индукцију путање традиционално називамо и $\mathbb{J}$ правило. Поред индукције путање можемо дефинисати и базну индукцију путање, коју традиционално називамо $\mathbb{H}$ правило, и у језику Agda записујемо као:

```
ℍ : {X : 𝒰 ·} (x : X) (P : (y : X) → x == y → 𝒱 ·)
  → P x (refl x) → (y : X) (p : x == y) → P y p
ℍ x P p-refl y (refl x) = p-refl
```

У наставку текста биће описане формализације лема о особинама типова индентитета које су наведене и доказане у поглављу 2.7.

Лему 1 (о инверзу) и лему 2 (о надовезивању) у језику Agda формализујемо као:

```
_⁻¹ : {X : 𝒰 ·} {x y : X}
    → x == y → y == x
(refl x) ⁻¹ = refl x

infixr 11 _·_

_·_ : {X : 𝒰 ·} {x y z : X}
    → x == y → y == z → x == z
refl _ · q = q
```

Често надовезујемо и више од две путање, те тако добијамо ланац надовезаних путањи. Због тога, згодно је на сваком кораку пратити тренутни елемент ланца, као и последњи елемент ланца. То нам омогућава следећа синтакса:

```
_==⟨_⟩_ : {X : 𝒰 ·} (x : X) {y z : X}
        → x == y → y == z → x == z
x ==⟨ p ⟩ q = p · q

_□ : {X : 𝒰 ·} (x : X)
   → x == x
x □ = refl x
```

Резултат леме 3, односно да тип идентитета на датом нивоу има својства слабог ∞-групоида, у језику Agda формализујемо као:

```
left-unit : {X : 𝒰 ·} {x y : X} (p : x == y)
          → (refl x) · p == p
left-unit (refl x) = refl (refl x)

right-unit : {X : 𝒰 ·} {x y : X} (p : x == y)
           → p · (refl y) == p
right-unit (refl x) = refl (refl x)

left-inv : {X : 𝒰 ·} {x y : X} (p : x == y)
         → p ⁻¹ · p == refl y
left-inv (refl x) = refl (refl x)

right-inv : {X : 𝒰 ·} {x y : X} (p : x == y)
          → p · p ⁻¹ == refl x
right-inv (refl x) = refl (refl x)

double-inv : {X : 𝒰 ·} {x y : X} (p : x == y)
           → (p ⁻¹) ⁻¹ == p
double-inv (refl x) = refl (refl x)

assoc : {X : 𝒰 ·} {x y z w : X}
        (p : x == y) (q : y == z) (r : z == w)
      → (p · q) · r == p · (q · r)
assoc (refl _) q r = refl (q · r)
```

Лему 4, односно акцију путање формализујемо као:

```
ap : {X : 𝒰 ·} {Y : 𝒱 ·} (f : X → Y) {x y : X}
   → x == y → f x == f y
ap f (refl x) = refl (f x)
```

Резултат леме 5, односно особине акције путање формализујемо као:

```
ap-id : {X : 𝒰 ·} {x y : X} (p : x == y)
      → p == ap id p
ap-id (refl x) = refl (refl x)
```

```
ap-comp : {X : 𝒰 ·} (f g : X → X)
          {x y z : X} (p : x == y)
        → ap g (ap f p) == ap (g ∘ f) p
ap-comp f g (refl x) = refl (refl (g (f x)))

ap-refl : {X : 𝒰 ·} {Y : 𝒱 ·} (f : X → Y) (x : X)
        → ap f (refl x) == refl (f x)
ap-refl f x = refl (refl (f x))

ap-inv : {X : 𝒰 ·} {Y : 𝒱 ·} (f : X → Y)
         {x y : X} (p : x == y)
       → ap f (p ⁻¹) == (ap f p) ⁻¹
ap-inv f (refl x) = refl (ap f (refl x))

ap-concat : {X : 𝒰 ·} {Y : 𝒱 ·} (f : X → Y)
            {x y z : X} (p : x == y) (q : y == z)
          → ap f (p · q) == ap f p · ap f q
ap-concat f (refl x) q = refl (ap f q)
```

Резултат леме 6, односно транспорт, у језику Agda формализујемо као:

```
tr : {A : 𝒰 ·} (B : A → 𝒱 ·) {x y : A}
  → x == y → B x → B y
tr B (refl x) = id
```

Природни бројеви

Природни бројеви $0, 1, 2, \ldots$ представљају основне математичке конструкције. Тип природних бројева $\mathbb{N}$ у језику Agda је већ детаљно уведен у поглављу 3.2 као:

```
data ℕ : 𝒰₀ · where
  zero : ℕ
  succ : ℕ → ℕ

{-# BUILTIN NATURAL ℕ #-}
```

```
ℕ-induction : (P : ℕ → 𝒰 ˙ )
            → P 0
            → ((n : ℕ) → P n → P (succ n))
            → (n : ℕ) → P n
ℕ-induction P p₀ pₛ zero = p₀
ℕ-induction P p₀ pₛ (succ n) = pₛ n (ℕ-induction P p₀ pₛ n)
```

Правило рекурзије ℕ-rec дефинишемо као специјалан случај правила индукције ℕ-ind када тип P не зависи од ℕ. У језику Agda то записујемо као:

```
ℕ-recursion : (A : 𝒰 ˙ )
            → A
            → (ℕ → A → A)
            → ℕ → A
ℕ-recursion A = ℕ-induction (λ _ → A)
```

Над типом природних бројева ℕ можемо дефинисати разне операције, као што су сабирање, множење, степеновање и израчунавање факторијела.

```
infixl 20 _+ℕ_
infixl 21 _*ℕ_
infixr 22 _^ℕ_
infixl 23 _!

_+ℕ_ : ℕ → ℕ → ℕ
0 +ℕ n = n
(succ m) +ℕ n = succ (m +ℕ n)

_*ℕ_ : ℕ → ℕ → ℕ
0 *ℕ n  = 0
(succ m) *ℕ n = m *ℕ n +ℕ n

_^ℕ_ : ℕ → ℕ → ℕ
m ^ℕ 0 = 1
m ^ℕ (succ n) = m *ℕ m ^ℕ n

_! : ℕ → ℕ
0 ! = 1
(succ n) ! = succ n *ℕ n !
```

Ове операције имају одређена својства која можемо доказати. На пример, за природне бројеве и операцију сабирања природно је да важи $0_{\mathbb{N}} +_{\mathbb{N}} n =_{\mathbb{N}} n$, као и $n +_{\mathbb{N}} 0_{\mathbb{N}} =_{\mathbb{N}} n$. Већ је дискутовано зашто је овде неопходно користити исказну једнакост (за детаље видети поглавље 2.7). У наставку су приказана нека својства природних бројева и операције сабирања, као и њихове конструкције:

```
left-zero-law-+ℕ : (n : ℕ) → 0 +ℕ n == n
left-zero-law-+ℕ n = refl n

right-zero-law-+ℕ : (n : ℕ) → n +ℕ 0 == n
right-zero-law-+ℕ 0 = refl 0
right-zero-law-+ℕ (succ n) = ap succ (right-zero-law-+ℕ n)

left-unit-law-+ℕ : (n : ℕ) → 1 +ℕ n == succ n
left-unit-law-+ℕ n = refl (succ n)

right-unit-law-+ℕ : (n : ℕ) → n +ℕ 1 == succ n
right-unit-law-+ℕ 0 = refl 1
right-unit-law-+ℕ (succ n) = ap succ (right-unit-law-+ℕ n)

left-succ-law-+ℕ : (m n : ℕ) → succ m +ℕ n == succ (m +ℕ n)
left-succ-law-+ℕ m n = refl (succ (m +ℕ n))

right-succ-law-+ℕ : (m n : ℕ) → m +ℕ succ n == succ (m +ℕ n)
right-succ-law-+ℕ 0 n = refl (succ n)
right-succ-law-+ℕ (succ m) n = ap succ (right-succ-law-+ℕ m n)

associative-+ℕ : (m n k : ℕ) → (m +ℕ n) +ℕ k == m +ℕ (n +ℕ k)
associative-+ℕ 0 n k = refl (n +ℕ k)
associative-+ℕ (succ m) n k = ap succ (associative-+ℕ m n k)

commutative-+ℕ : (m n : ℕ) → m +ℕ n == n +ℕ m
commutative-+ℕ 0 n = right-zero-law-+ℕ n ⁻¹
commutative-+ℕ (succ m) n = (succ (m +ℕ n)) ==⟨
                                ap succ (commutative-+ℕ m n)
                            ⟩ ((succ (n +ℕ m)) ==⟨
                                right-succ-law-+ℕ n m ⁻¹
                            ⟩ ((n +ℕ succ m) □))
```

Приметимо да се за конструисање доказа ових својстава користе особине путање. Прецизније, често користимо акције над путањама, инверз путање, као и надовезивање путања. Поред тога, приметимо и да у доказу о комутативности, користимо синтаксу за ланац путања.

Простор кодова над типом природних бројева, односно дефиницију 2.7.1, у језику Agda записујемо као:

```
code-ℕ : ℕ → ℕ → 𝒰₀ ·
code-ℕ 0 0 = 𝟙
code-ℕ 0 (succ n) = 𝟘
code-ℕ (succ m) 0 = 𝟘
code-ℕ (succ m) (succ n) = code-ℕ m n
```

Простор кодова је рефлексивна релација. Односно, резултат леме 7 у језику Agda формализујемо као:

```
relf-code-ℕ : (n : ℕ) → code-ℕ n n
relf-code-ℕ 0 = ⋆
relf-code-ℕ (succ n) = relf-code-ℕ n
```

Резултат леме 8, постојање функција које трансформишу једнакост у код, као и код у једнакост, у језику Agda формализујемо као:

```
encode-ℕ : {m n : ℕ} → m == n → code-ℕ m n
encode-ℕ {m} {n} p = tr (code-ℕ m) p (relf-code-ℕ m)

decode-ℕ : (m n : ℕ) → code-ℕ m n → m == n
decode-ℕ 0 0 code = refl 0
decode-ℕ (succ m) (succ n) code = ap succ (decode-ℕ m n code)
```

На основу претходне две функције можемо показати Пеанову седму и осму аксиому. Пеанова седма аксиома тврди да уколико су одређене инстанце целих бројева једнаке, тада ће бити једнаки и њихови следбеници, као и да важи обрнуто. Док Пеанова осма аксиома тврди да нула никада није следбеник неког природног броја. У језику Agda то записујемо као:

```
injective-succ-ℕ : (m n : ℕ) → succ m == succ n → m == n
injective-succ-ℕ m n e = decode-ℕ m n (encode-ℕ e)
```

```
peano-7-axiom : (n m : ℕ) → (m == n) ↔ (succ m == succ n)
peano-7-axiom n m = ap succ , injective-succ-ℕ m n

peano-8-axiom : (n : ℕ) → ¬ (0 == succ n)
peano-8-axiom n = encode-ℕ
```

По узору на дефиницију 2.7.1 (простор кодова над природним бројевима), можемо дефинисати релацију поретка $\leq_{\mathbb{N}}$ над природним бројевима као:

```
_≤ℕ_ : ℕ → ℕ → 𝒰₀ ·
0 ≤ℕ n = 𝟙
succ m ≤ℕ 0 = 𝟘
succ m ≤ℕ succ n = m ≤ℕ n
```

Булов тип

Булов тип 2 је специјалан случај типа копроизвода $A + B$ када су и A и B јединични типови, тј. $2 :\equiv \mathbb{1} + \mathbb{1}$. Као такав, њега настањују једино $\mathsf{inl}(\star)$ и $\mathsf{inr}(\star)$, које другачије називамо true и false. У језику Agda, Булов тип 2 уводимо на следећи начин:

```
2 : 𝒰₀ ·
2 = 𝟙 + 𝟙

pattern true = inl ⋆
pattern false = inr ⋆
```

Правило индукције 2-ind заједно са правилом израчунавања 2-comp, у језику Agda записујемо као:

```
2-induction : (P : 2 → 𝒰 · )
            → (P true)
            → (P false)
            → (b : 2) → (P b)
2-induction P p₀ p₁ true = p₀
2-induction P p₀ p₁ false = p₁
```

Уколико у правилу индукције 2-ind заменимо одговарајуће аргументе добијамо конструкцију коју називамо *if-then-else*:

$$\begin{aligned}
&\mathsf{if_then_else} : \{P : \mathbb{2} \to \mathcal{U}\ \cdot\ \} \\
&\qquad\qquad \to (b : \mathbb{2}) \\
&\qquad\qquad \to (P\ \mathsf{true}) \\
&\qquad\qquad \to (P\ \mathsf{false}) \\
&\qquad\qquad \to (P\ b) \\
&\mathsf{if\ true\ then}\ x\ \mathsf{else}\ y = x \\
&\mathsf{if\ false\ then}\ x\ \mathsf{else}\ y = y
\end{aligned}$$

Глава 5

Закључак

Основни циљ овог рада је теоријско и практично изучавање МЛТТ-а као и формализација основних објеката и конструкција МЛТТ-а у типски зависном програмском језику Agda. Као што је већ напоменуто, формализација у оквиру интерактивних доказивача се једина сматра формалном. Са једне стране, формализација у оквиру интерактивних доказивача нам доноси комплетност и прецизност тако што минимизује настанак грешке. Док са друге стране, представља исцрпан посао и укључује техниче вештине, које многи математичари не поседују. Штавише, рачунарци лако савладају техничке вештине, али им недостаје теоријско знање, и због тога МЛТТ, а и ХоТТ, представљају идеалну синергију између математике и рачунарства.

Као резултат овог рада настала је библиотека InTT, у којој су формализовани основни концепти МЛТТ-а. Због тога се библиотека InTT може користити као почетна тачка која се даље може надоградити у ХоТТ. Поред тога, како садржи само основне концепте и конструкције, представља идеалан извор за изучавање МЛТТ-а и ХоТТ-а. Већина других библиотека покушава да формализује што више математичког знања, што ствара огромне графове зависности који могу одбити новајлије од области.

Поред практичног резултата, овај рад покушава да помери границе фундаменталног бављена математиком. Пружа један алтернативни начин заснивања математике. Одбацује теорију скупова и класичну логику уводећи теорију типова као основни систем, у коме се даље формулишу појмови и доказују теореме на најригорознији могући начин. Методи доказивања теорема са класичног приступа прелазе на конструктивистички приступ, пружајући прецизније и ефектније доказе. Свеобухватно, пружа један концизан и систе-

матски начин рада.

Овај начин формализације математике се тренутно сматра најбољим, али и он има своја ограничења. Иако је велика база знања формализована, постоји још већа база генералног математичког знања која тек треба да се формализује. Са друге стране, сво то математичко знање се чува неформално и његова основа често није теорија типова. Често је за један неформалан доказ потребно и више од 1000 линија кôда. Све то читав поступак формализације, у оквиру МЛТТ-а и ХоТТ-а, чини исцрпним, тешким, спорим и изазовним.

Даљи рад подразумева надоградњу тренутне библиотеке концептима из ХоТТ-а. Тиме се добија добра основа за формализовање генералног математичког знања у оквиру ХоТТ-а. Након тога, даљи рад може да се настави у два правца: (1) наставак развијања библиотеке напреднијим концептима и генералним математичким знањем; (2) формализација генералног математичког знања и допринос у развоју библиотека Agda-UniMath и HoTT-Agda.

Коначно, овај рад има за циљ да допринесе развоју ХоТТ-а, тако што ће привући људе из разних области и подстаћи их да изучавају ХоТТ и формализују математику. Срећно формализовање!

Литература

[1] Mark van Atten. „The Development of Intuitionistic Logic". У: *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, 2023.

[2] Steve Awodey и Michael A. Warren. „Homotopy theoretic models of identity types". У: *Mathematical Proceedings of the Cambridge Philosophical Society* 146.1 (2009), стр. 45–55.

[3] Henk P Barendregt. „Introduction to generalized type systems". У: *Journal of Functional Programming* 1.2 (1991), стр. 124–154.

[4] Andrej Bauer и др. *The HoTT Library: A formalization of homotopy type theory in Coq*. 2016. arXiv: `1610.04591 [cs.LO]`. URL: `https://arxiv.org/abs/1610.04591`.

[5] Marc Bezem и др. *Symmetry*. Commit: `bc24ac2`. Авг. 22, 2024. URL: `https://github.com/UniMath/SymmetryBook`.

[6] Nicolas Bourbaki. *Elements of Mathematics (Éléments de mathématique)*. Springer-Verlag, 1935–2024. URL: `https://www.springer.com/series/47`.

[7] Edwin Brady. „Idris, a General Purpose Dependently Typed Programming Language: Design and Implementation". У: *Journal of Functional Programming* 23.5 (2013), стр. 552–593.

[8] Douglas Bridges, Erik Palmgren и Hajime Ishihara. „Constructive Mathematics". У: *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, 2022.

[9] Luitzen Egbertus Jan Brouwer. „On the foundations of mathematics". У: *Collected works* 1 (1907), стр. 11–101.

[10] Guillaume Brunerie и др. *Homotopy Type Theory in Agda*. URL: `https://github.com/HoTT/HoTT-Agda`.

[11] Alonzo Church. „A formulation of the simple theory of types". У: *The journal of symbolic logic* 5.2 (1940), стр. 56–68.

[12] Alonzo Church. *The calculi of lambda-conversion*. 6. Princeton University Press, 1985.

[13] Cyril Cohen и др. „Cubical Type Theory: a constructive interpretation of the univalence axiom". У: (2016). arXiv: `1611.02108 [cs.LO]`.

[14] Robert L. Constable и др. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, 1986.

[15] Thierry Coquand, Simon Huber и Anders Mörtberg. „On Higher Inductive Types in Cubical Type Theory". У: (2018). arXiv: `1802.01170 [cs.LO]`.

[16] Nicolaas Govert De Bruijn. „Automath: a language for mathematics". У: (1973).

[17] Euclid. *The Thirteen Books of Euclid's Elements*. Ур. Thomas L. Heath. 2nd. Dover Publications, 1956.

[18] Robert Harper. *The Holy Trinity*. 2011.

[19] Arend Heyting. *Intuitionism: an introduction*. Св. 41. Elsevier, 1966.

[20] Martin Hofmann и Thomas Streicher. „The groupoid interpretation of type theory". У: *Twenty-five years of constructive type theory (Venice, 1995)* 36 (1998), стр. 83–111.

[21] William Alvin Howard. „The Formulae-as-Types Notion of Construction". У: *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Academic Press, 1980.

[22] Andrej Kolmogoroff. „Zur deutung der intuitionistischen logik". У: *Mathematische Zeitschrift* 35.1 (1932), стр. 58–65.

[23] Per Martin-Löf. „An intuitionistic theory of types". У: *Twenty-five years of constructive type theory* 36 (1998), стр. 127–172.

[24] Per Martin-Löf. „An Intuitionistic Theory of Types: Predicative Part". У: Studies in Logic and the Foundations of Mathematics 80 (1975), стр. 73–118.

[25] Per Martin-Löf. „Constructive Mathematics and Computer Programming". У: Studies in Logic and the Foundations of Mathematics 104 (1982), стр. 153–175.

[26] Per Martin-Löf. „Philosophical aspects of intuitionistic type theory". У: *Unpublished notes by M. Wijers from lectures given at the Faculteit der Wijsbegeerte, Rijksuniversiteit Leiden* (1993).

[27] Per Martin-Löf и Giovanni Sambin. *Intuitionistic type theory*. Св. 9. Bibliopolis Naples, 1984.

[28] Leonardo de Moura и Sebastian Ullrich. *The Lean 4 Theorem Prover and Programming Language*. Microsoft Research и Karlsruhe Institute of Technology, 2021. URL: `https://lean-lang.org`.

[29] Leonardo de Moura и др. *The Lean Theorem Prover*. Microsoft Research и Carnegie Mellon University, 2015. URL: `https://lean-lang.org`.

[30] Tobias Nipkow, Lawrence C. Paulson и Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science. Technische Universität München. Springer-Verlag, 2002. URL: `https://isabelle.in.tum.de`.

[31] Ulf Norell. „Dependently typed programming in Agda". У: *Proceedings of the 4th International Workshop on Types in Language Design and Implementation*. Association for Computing Machinery, 2009, стр. 230–266.

[32] Ulf Norell. „Towards a practical programming language based on dependent type theory". Докторска теза. Chalmers University of Technology, 2007.

[33] Egbert Rijke. *Introduction to Homotopy Type Theory*. 2022. arXiv: `2212.11082 [math.LO]`.

[34] Egbert Rijke и др. *The agda-unimath library*. URL: `https://github.com/UniMath/agda-unimath/`.

[35] Bertrand Russell. „Mathematical logic as based on the theory of types". У: *American journal of mathematics* 30.3 (1908), стр. 222–262.

[36] The Agda Development Team. *The Agda Reference Manual*. Version 2.6.3. Chalmers University of Technology. 2024. URL: `https://agda.readthedocs.io/en/v2.6.3/`.

[37] The Coq Development Team. *The Coq Proof Assistant*. Inria. URL: `https://coq.inria.fr`.

[38] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: `https://homotopytypetheory.org/book`, 2013.

[39] Mark Van Atten и Göran Sundholm. „LEJ Brouwer's 'Unreliability of the Logical Principles': A New Translation, with an Introduction". У: *History and Philosophy of Logic* 38.1 (2017), стр. 24–47.

[40] Andrea Vezzosi, Anders Mörtberg и Andreas Abel. „Cubical Agda: A dependently typed programming language with univalence and higher inductive types". У: *Journal of Functional Programming* 31 (2021), e8.

[41] Vladimir Voevodsky. *A very short note on the homotopy λ-calculus*. 2006.

[42] Vladimir Voevodsky. *Foundations – Voevodsky's original development of the univalent foundations of mathematics in Coq*. URL: `https://github.com/vladimirias/Foundations`.

[43] Vladimir Voevodsky. „The origins and motivations of Univalent Foundations". У: *The Institute Letter* (2014).

[44] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson и др. *UniMath — a computer-checked library of univalent mathematics*. URL: `https://github.com/UniMath/UniMath`.

[45] Philip Wadler. „Propositions as types". У: *Communications of the ACM* 58.12 (2015), стр. 75–84.

Биографија аутора

Андрија Урошевић је мастер студент смера Информатика, Математичког факултета, Универзитета у Београду. Основне студије је завршио 2022. године на истом факултету, где је развио интересе за математичку логику и формалне системе. Андрија је стекао вредно искуство током истраживачке праксе Математичког института, Српске академије наука и уметности (МИСАНУ), где се први пут сусрео са научним радом и објавио свој први научни рад. Поред истраживачог рада, током мастер студија, Андрија је био запослен као сарадник у настави Катедре за рачунарство и информатику, Математичог факултета у Београду. По завршетку мастер студија, Андрија настоји да унапреди своје образовање докторатом из Информатике, трудећи се да пружи напредак у формализацији математике.