Formalizacja naiwnej teorii typów

Agnieszka Kozubek

Grudzień 2011

1 Wstęp

Językiem współczesnej matematyki jest teoria mnogości. Praktycznie wszystkie dowody matematyczne przeprowadza się na gruncie teorii zbiorów. Wszystkie książki i artykuły naukowe pisze się zakładając niejawnie aksjomaty ZF lub ZFC. Nawet nauczanie matematyki, od przedszkola po uniwersytet, prowadzi się na gruncie teorii mnogości.

Jednak teoria mnogości została wprowadzona w zupełnie innym celu. Nie miała być językiem matematyki, ale uzasadnieniem jej niesprzeczności. Dlatego wszystkie złożone konstrukcje są zbudowane z elementarnych pojęć takich jak „zbiór" i „należenie do zbioru". Przywykliśmy do tego zjawiska i nie widzimy już jego wad. Wady te ujawniają się, gdy próbujemy uczyć podstaw teorii mnogości studentów. Zamiast objaśniać własności obiektów matematycznych, takich jak para uporządkowana, suma uogólniona zbiorów, czy liczby naturalne, musimy wyjaśniać szczegóły ich kodowania.

Językiem, który jest pozbawiony wielu wad teorii mnogości, jest teoria typów. Pojęciem typu posługujemy się w sposób nieformalny znacznie częściej niż nam się wydaje. Rozważmy funkcję, która odwzorowuje elementy zbioru A w elementy zbioru B. Taką funkcję można aplikować do elementów ze zbioru („typu") A. Podobnie suma $\bigcup A$ rodziny zbiorów A jest tego samego „typu", co elementy rodziny A, a nie tego typu co A.

Obecnie coraz częściej teorii typów używa się jako języka matematyki. Ma to miejsce zwłaszcza na pograniczu matematyki i informatyki, w systemach wspomagania dowodzenia. Teorie typów są podstawą teoretyczną dla wielu takich systemów [1, 2, 3, 4] i z powodzeniem są używane do formalizacji i weryfikacji ważnych twierdzeń matematycznych [11, 12, 16].

Teoria zbiorów ma jeszcze jedną wadę. Dwie podstawowe pojęcia są sklejone w jedno pojęcie „zbioru":

- zbiór jako dziedzina czy uniwersum;
- zbiór jako predykat.

Przywykliśmy uważać to utożsamienie za naturalne i oczywiste. Być może jednak tylko dlatego, że tak zostaliśmy nauczeni. Te dwa pojęcia są w istocie zupełnie różne, a nierozróżnianie ich prowadzi do paradoksu Russella.

Celem tej pracy jest stworzenie teorii typów, w której będzie wyraźne rozróżnienie pomiędzy dziedzinami i predykatami. Ponadto chcemy utożsamić predykaty z podzbiorami.

Ta idea pochodząca jeszcze od Cantora, została porzucona po odkryciu paradoksu Russella. Chcemy także, żeby podzbiór był zwykłym obiektem, co się rzadko zdarza w teorii typów. Na przykład w systemie Coq [2] podzbiór jest konstruktorem typowym, a zatem znajduje się na wyższym poziomie hierarchii uniwersów niż obiekty tego typu. Nie jest to zbyt wygodne i jak się okazuje, nie jest konieczne. Nasz system pokazuje, że w teorii typów można z powodzeniem wrócić do idei Cantora. Chcemy ponadto, by w naszej teorii można było uprawiać matematykę. Początkowy cel to teoria obejmująca matematykę elementarną, pomocna przy nauczaniu podstaw matematyki. Nasza teoria ma zaletę, której nie mają inne teorie: nie ma konieczności używania sztucznych kodowań.

Najważniejszym pytaniem jest kwestia niesprzeczności systemu. Bardzo łatwo jest stworzyć system typów, który będzie miał zbyt wielką siłę wyrazu i będzie można w nim doprowadzić do paradoksu. Przykładowo sprzeczna była pierwsza wersja teorii typów Martina-Löfa [19], co pokazał Girard w [15]. Także pierwsza próba stworzenia naszego systemu okazała się sprzeczna.

W pracy proponujemy system typów, który może być podstawą dla poszukiwanej teorii typów. Przedstawiamy definicję systemu wraz z przykładami. Największą część pracy stanowi dowód niesprzeczności. Ponadto część pracy to dyskusja różnych paradoksów w teoriach typów.

2 Pure Type Systems

Jako podstawę naszego systemu wybraliśmy formalizm zwany Pure Type Systems (PTS). Formalizm został odkryty niezależnie przez Berardiego [6] i Terlouwa [22]. Jest on generalizacją tzw. kostki Barendregta. Podstawowe własności PTS-ów są przedstawione w [5].

Prezentację formalizmu zaczniemy od składni. Załóżmy, że mamy nieskończony przeliczalny zbiór zmiennych oraz ustalony (zwykle skończony) zbiór sortów. *Termy* systemu są zdefiniowane za pomocą następującej gramatyki

$$T := s \mid x \mid (\lambda x{:}T.\, T) \mid (TT) \mid (\Pi x{:}T.\, T)$$

gdzie x jest zmienną, a s jest sortem. Przyjmujemy konwencję, że zewnętrzne nawiasy można pominąć. Ponadto aplikacja wiąże w lewo: zamiast $((PQ)R)$ możemy napisać PQR; abstrakcja i produkt wiążą w prawo: zamiast $(\lambda x{:}T_1(\lambda x{:}T_2.P))$ piszemy $\lambda x{:}T_1\lambda x{:}T_2.P$. Jeśli zmienna x nie występuje w B to zamiast $\Pi x{:}A.B$ można napisać $A \to B$.

Znaczenie termów jest następujące: term $(\lambda x{:}A.B)$ oznacza funkcję, która bierze argumentem x typu A i zwraca wartość B. Term AB to aplikacja funkcji A do argumentu B. Term $(\Pi x{:}A.B)$ oznacza zbiór wszystkich funkcji z typu A do typu B.

Dla każdego termu M definiujemy zbiór $FV(M)$ *zmiennych wolnych* termu M przez indukcję ze względu na strukturę termu M:

- $FV(s) = \emptyset$, jeśli s jest sortem,
- $FV(x) = \{x\}$, jeśli x jest zmienną,

- $FV(\lambda x{:}A.B) = FV(A) \cup (FV(B) - \{x\})$,
- $FV(AB) = FV(A) \cup FV(B)$,
- $FV(\Pi x{:}A.B) = FV(A) \cup (FV(B) - \{x\})$.

Wystąpienia zmiennych, które nie są wolne, nazywamy *związanymi*. Jak zwykle utożsamiamy termy ze względu na α-konwersję, to jest utożsamiamy wyrażenia, które różnią się tylko wystąpieniami zmiennych związanych. Szczegóły można znaleźć np. w [21].

Dla zmiennej x i termów M, N, definiujemy *podstawienie termu N w miejsce x w termie M*, co oznaczamy $M[x := N]$, przez indukcję ze względu na strukturę M:

- $x[x := N] = N$,
- $y[x := N] = y$, jeśli $x \neq y$,
- $(\lambda z{:}A.B)[x := N] = \lambda z{:}A[x := N].B[x := N]$, jeśli $z \notin FV(N)$ lub $x \notin FV(B)$,
- $(AB)[x := N] = A[x := N]B[x := N]$,
- $(\Pi z{:}A.B)[x := N] = \Pi z{:}A[x := N].B[x := N]$, jeśli $z \notin FV(N)$ lub $x \notin FV(B)$.

W zbiorze termów definiujemy *relację beta-redukcji*, którą oznaczamy $\rightarrow_\beta$. Relacja $\rightarrow_\beta$ jest zdefiniowana przez regułę

$$(\lambda x{:}T.A)B \rightarrow_\beta A[x := B]$$

oraz reguły domknięcia na kontekst: jeśli $A \rightarrow_\beta A'$ to

$$\begin{array}{cc} AB \rightarrow_\beta A'B & BA \rightarrow_\beta BA' \\ \lambda x{:}A.B \rightarrow_\beta \lambda x{:}A'.B & \lambda x{:}B.A \rightarrow_\beta \lambda x : B.A' \\ \Pi x{:}A.B \rightarrow_\beta \Pi x{:}A'.B & \Pi x{:}B.A \rightarrow_\beta \Pi x{:}B.A'. \end{array}$$

Mówimy, że term M jest w *postaci normalnej* jeśli nie istnieje takie N, że $M \rightarrow_\beta N$. Relacja $\rightarrow_\beta^*$ to domknięcie przechodnio-zwrotne relacji $\rightarrow_\beta$. Piszemy, że $A =_\beta B$, jeśli istnieje ciąg $A = A_0, A_1, \ldots, A_n = B$ taki, że dla każdego $i = 0, \ldots, n-1$ zachodzi $A_i \rightarrow_\beta A_{i+1}$ lub $A_{i+1} \rightarrow_\beta A_i$. Term M *ma postać normalną* wtedy i tylko wtedy, gdy istnieje ciąg redukcji zaczynający się od M, który kończy się termem w postaci normalnej N. Term M ma własność *silnej normalizacji*, jeśli wszystkie ciągi redukcji zaczynające się w M są skończone.

Kontekst to skończony (być może pusty) ciąg deklaracji zmiennych $x_1 : A_1, \ldots, x_n : A_n$. Będziemy używać greckich liter Γ, Δ, Σ na oznaczenie kontekstów. *Dziedziną* kontekstu $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ nazwiemy zbiór $\{x_1, \ldots, x_n\}$. Będziemy ją oznaczać przez $dom(\Gamma)$.

System typów wyprowadza asercje postaci $\Gamma \vdash A : B$, które należy czytać: „w kontekście Γ term A jest typu B".

PTS jest zdefiniowany za pomocą trzech zbiorów $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, gdzie

- $\mathcal{S}$ to zbiór *s*ortów;

- $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ to zbiór *aksjomatów*;
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ to zbiór *reguł*.

Reguły przypisywania typów są określone przez trójkę $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ w następujący sposób:

$$\text{(Ax)} \quad \vdash s_1 : s_2 \qquad s_1 : s_2 \in \mathcal{A}$$

$$\text{(Var)}\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad x \notin dom(\Gamma)$$

$$\text{(Weak)}\ \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \quad x \notin dom(\Gamma)$$

$$\text{(App)}\ \frac{\Gamma \vdash M : (\Pi x{:}A.B) \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$$

$$\text{(Abs)}\ \frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash (\Pi x{:}A.B) : s}{\Gamma \vdash (\lambda x{:}A.M) : (\Pi x{:}A.B)}$$

$$\text{(Prod)}\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A.B) : s_3} \quad (s_1, s_2, s_3) \in \mathcal{R}$$

$$\text{(Conv)}\ \frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad \Gamma \vdash B =_\beta B'}{\Gamma \vdash A : B'}$$

System typów ma własność *normalizacji* wtedy i tylko wtedy, gdy każdy term typowalny w tym systemie ma postać normalną. System typów ma własność *silnej normalizacji* wtedy i tylko wtedy, gdy każdy term typowalny w tym systemie ma własność silnej normalizacji.

System typów jest *sprzeczny* wtedy i tylko wtedy, gdy każdy typ jest zamieszkały, to znaczy dla każdego T takiego, że $\vdash T : s$ dla pewnego sortu s, istnieje term M taki, że $\vdash M : T$. System jest *niesprzeczny*, jeśli nie jest sprzeczny.

Wiadomo, że system, który ma własność silnej normalizacji jest niesprzeczny.

3 Mniej Naiwna Teoria Typów

Pierwszą próbą stworzenia naszego systemu była Naiwna Teoria Typów (NTT). Jest to PTS o następującej specyfikacji:

$$\mathcal{S} = *, \Box$$
$$\mathcal{A} = * : \Box$$
$$\mathcal{R} = (*, *, *), (*, \Box, *), (*, \Box, \Box).$$

Najciekawsza w tym systemie jest reguła $(*, \Box, *)$. Jest ona interesująca z technicznego punktu widzenia. W literaturze najczęściej spotyka się systemy, w których wszystkie reguły są postaci (s_1, s_2, s_2). Jest tylko kilka przykładów systemów z regułami postaci (s_1, s_2, s_3), gdzie $s_2 \neq s_3$ (patrz np. [5, Rozdział 5.2]). Interesujące jest także znaczenie tej reguły. Sort $*$ reprezentuje typy (formuły). Reguła stwierdza, że produkty postaci $\tau \to *$ są w sorcie $*$, a więc są typami. Produkty tej postaci to typy potęgowe: element M typu $\tau \to *$ jest podzbiorem typu τ, po otrzymaniu elementu typu τ daje w wyniku formułę. Nasza reguła stwierdza, że podzbiory są zwykłymi obiektami, a nie konstruktorami typów. W ten sposób zgodnie z naszym postulatem utożsamiamy predykaty i podzbiory. Jednakże tak zdefiniowany system jest sprzeczny [13, 18]. W systemie można powtórzyć tzw. paradoks Girarda [7, 17].

Ponieważ pierwsza wersja systemu okazała się sprzeczna, trzeba było znaleźć inną podstawę dla naszej teorii. Druga próba to system zwany Mniej Naiwna Teorią Typów (ang. *Less Naive Type Theory*, LNTT). Jest to PTS o następującej specyfikacji

$$\begin{aligned}
\mathcal{S} &= *^t, *^p, \Box^t, \Box^p \\
\mathcal{A} &= *^t : \Box^t, *^p : \Box^p \\
\mathcal{R} &= (*^t, *^t, *^t), (*^p, *^p, *^p), (*^t, *^p, *^p), (*^t, \Box^p, *^t), (*^t, \Box^t, \Box^t), (\Box^p, *^p, *^p).
\end{aligned}$$

LNTT jest poprawioną systemu NTT. Każdy sort oryginalnego systemu został rozdzielony na dwa: jeden w wersji p, drugi w wersji t. Sorty klasy t reprezentują obiektową część systemu (typy danych), a sorty klasy p reprezentują jego część logiczną. To rozróżnienie jest podobne do sytuacji w systemie Coq [2], gdzie mamy sorty *Set* i *Prop*.

Skoro zmieniliśmy sorty, to musieliśmy także zmienić reguły. Znaczenie reguł jest następujące:

- $(*^t, *^t, *^t)$ wprowadza przestrzeń funkcyjną,
- $(*^p, *^p, *^p)$ wprowadza implikację, czyli logiczną przestrzeń funkcyjną,
- $(*^t, *^p, *^p)$ wprowadza kwantyfikator uniwersalny: jeśli τ jest typem, a φ jest formułą, to $\Pi x{:}\tau.\varphi$ jest równoważny formule pierwszego rzędu $\forall x : \tau.\varphi$,
- $(*^t, \Box^t, \Box^t)$ wprowadza typy zależne, to jest typy, które zależą od termów,
- $(\Box^p, *^p, *^p)$ wprowadza polimorfizm na formułach (logikę wyższego rzędu) – formuły mogą zależeć od formuł.

Reguła $(*, \Box, *)$ przyjmuje teraz postać $(*^t, \Box^p, *^t)$. Podzbiory są termami typu $\tau \to *^p$: otrzymawszy obiekt typu τ dają w wyniku formułę. Reguła stwierdza, że takie produkty (typy potęgowe) są typami, a zatem podzbiory są obiektami.

Składnia PTS-ów jest jednolita, ale termy typowalne w kontekście Γ możemy podzielić na kategorie. Mówimy, że

- M jest *typowalny w kontekście* Γ, jeśli istnieje T takie, że $\Gamma \vdash M : T$.

- M jest *rodzajem w kontekście* Γ, jeśli $\Gamma \vdash M : \Box^t$.
- M jest *typem w kontekście* Γ, jeśli $\Gamma \vdash M : *^t$.
- M jest *formułą w kontekście* Γ, jeśli $\Gamma \vdash M : *^p$.
- M jest *typem konstruktora w kontekście* Γ, jeśli istnieje term T taki, że $\Gamma \vdash M : T$ and $\Gamma \vdash T : \Box^t$.
- M jest *obiektem w kontekście* Γ, jeśli istnieje term T taki, że $\Gamma \vdash M : T$ i $\Gamma \vdash T : *^t$.
- M jest *dowodem w kontekście* Γ, jeśli istnieje term T taki, że $\Gamma \vdash M : T$ i $\Gamma \vdash T : *^t$.

Struktura systemu jest pokazana na rysunku 1.

	$\Box^t$		$\Box^p$	
	..	..	..	
kinds	B	$*^t$	$*^p$	
	..	..	..	
type constructors	κ	τ types	φ	formulas
		..	..	
		M objects	D	proofs

Rysunek 1: Struktura systemu

Własność silnej normalizacji, a zatem niesprzeczność tego systemu została udowodniona w [18].

4 Typy indukcyjne

Mniej Naiwna Teoria Typów ma dużą siłę wyrazu, jednak nie jest zbyt wygodna do definiowania typów takich jak liczby naturalne, listy, wartości logiczne, itp. Aby to zmienić, system został rozszerzony o typy indukcyjne. Wersja typów indukcyjnych, której używamy została wprowadzona przez Coquanda i Paulin-Mohring [10] w rachunku konstrukcji [9], a następnie rozszerzona w [20]. Ważne własności, takie jak własność silnej normalizacji, zostały udowodnione przez Wernera w [24].

Rozszerzamy składnię systemu o następujące konstrukcje:

$$\mathrm{Ind}(X : T)\{\overrightarrow{T}\} \mid \mathrm{Constr}(n, T) \mid \mathrm{Elim}(T, T, \overrightarrow{T}, T)\{\overrightarrow{T}\}$$

gdzie n jest liczbą naturalną. Wyjaśnimy znaczenie tych termów używając jako przykładu typu liczb naturalnych. Definicja typu jest następująca

$$Nat = \mathrm{Ind}(X : *^t)\{X \mid X \to X\}.$$

Używamy składni zapożyczonej z Coq-a: symbol | rozdziela elementy ciągu. Definicja mówi, że Nat to typ indukcyjny o dwóch konstruktorach: typu Nat oraz typu $Nat \to Nat$. Typy konstruktora muszą spełniać tzw. warunek ścisłej pozytywności, tzn. dla każdego typu indukcyjnego I argumenty rekurencyjne każdego konstruktora muszą być postaci $\Pi\vec{x} : \vec{T}.I$, gdzie I nie występuje w $\vec{T}$. Wiadomo, że bez tego warunku system jest sprzeczny [24].

Term $\mathrm{Constr}(n, I)$ reprezentuje n-ty konstruktor typu indukcyjnego I. Zatem term $\mathrm{Constr}(0, Nat)$ reprezentuje liczbę naturalną 0, a term $\mathrm{Constr}(1, Nat)$ reprezentuje funkcję następnika. Wtedy

- $\mathrm{Constr}(1, Nat)\mathrm{Constr}(0, Nat)$ oznacza liczbę naturalną 1,
- $\mathrm{Constr}(1, Nat)(\mathrm{Constr}(1, Nat)\mathrm{Constr}(0, Nat))$ oznacza liczbę naturalną 2.

Termy odpowiadające schematom eliminacji są bardziej skomplikowane. Są różne rodzaje eliminacji: niezależna, zależna, słaba, silna. Najprostsza jest eliminacja niezależna. Przypuśćmy, że w kontekście Γ mamy następujące przypisania typów

$$P : *^t, \quad f_0 : P, \quad f_1 : Nat \to P \to P, \quad m : Nat.$$

Wówczas reguły typowania orzekają, że

$$\mathrm{Elim}(Nat, P, \epsilon, m)\{f_0 \mid f_1\} : P.$$

Jeśli dla ustalonego P będziemy abstrahować od f_0, f_1 i m, to dla liczb naturalnych dostaniemy term $NatElim_{nodep}$ postaci

$$NatElim_{nodep} : P \to (Nat \to P \to P) \to Nat \to P.$$

Jak widać, eliminacja reprezentuje rekursor na liczbach naturalnych.

Eliminacja zależna pozwala tworzyć obiekty o typie zależącym od eliminowanego termu. Przypuśćmy, że w środowisku Γ mamy następujące przypisania typu gdzie s to $*^t$ albo $*^p$

$$P : Nat \to s, \quad f_0 : P0, \quad f_1 : (\Pi k : Nat(Pk \to P(Sk))), \quad m : Nat.$$

Reguły typowania mówią wówczas, że

$$\mathrm{Elim}(Nat, P, \epsilon, m)\{f_0 \mid f_1\} : (Pm).$$

Abstrahując od P, f_0, f_1 i m otrzymamy, term typu

$$NatElim_{dep} : \Pi P : Nat \to s(P0 \to (\Pi k : Nat(Pk \to P(Sk))) \to \Pi n : Nat.Pn).$$

Biorąc $s = *^p$, otrzymamy term reprezentujący schemat indukcji na liczbach naturalnych.

Reguły redukcji dla liczb naturalnych są następujące:

$$\begin{aligned} \mathrm{Elim}(Nat, Q, \epsilon, 0)\{f_0 \mid f_1\} &\to f_0\ 0 \\ \mathrm{Elim}(Nat, Q, \epsilon, Sn)\{f_0 \mid f_1\} &\to f_1\ n\ \mathrm{Elim}(Nat, Q, \epsilon, n)\{f_0 \mid f_1\}. \end{aligned}$$

Relację redukcji indukowana tymi regułami nazywamy ι-redukcją. Redukcja jest częścią reguły konwersji w systemie.

4.1 Silna eliminacja

W zasadzie można by mieć także następujący schemat eliminacji

$$NatElim_{Tnodep} : \Pi P : \Box^t(P \to (Nat \to P \to P) \to Nat \to P)$$

dający możliwość tworzenia typów. Ten wariant eliminacji nazywamy *silną eliminacją*. W LNTT z typami indukcyjnymi ten schemat nie jest dozwolony. W systemach, które mają silną eliminację, np. w rachunku konstrukcji, podlega ona ważnemu ograniczeniu. Jest dozwolona tylko dla tzw. *małych typów indukcyjnych*, tj. dla typów, które nie mają argumentów będących typami. Wiadomo, że silna eliminacja na dużych typach indukcyjnych prowadzi do paradoksu [7]

W LNTT z typami indukcyjnymi nie wolno definiować typów przez silną eliminację. Mimo to da się zdefiniować wiele funkcji, do których w innych systemach konieczna jest silna eliminacja. Możemy na przykład zdefiniować sumę listy zbiorów. Rozważmy typ

$$List = \mathrm{Ind}(X : *^t)\{X \mid (\tau \to *^p) \to X \to X\}.$$

Jest to typ listy podzbiorów typu τ. Używając słabej eliminacji można w LNTT zdefiniować funkcję $union : List \to (\tau \to *^p)$ obliczającą sumę listy. Tej definicji nie można przenieść do rachunku konstrukcji indukcyjnych (CIC). W CIC typ *List* jest duży. Zatem aby zdefiniować funkcję *union* trzeba użyć silnej eliminacji, która w tej sytuacji jest niedozwolona.

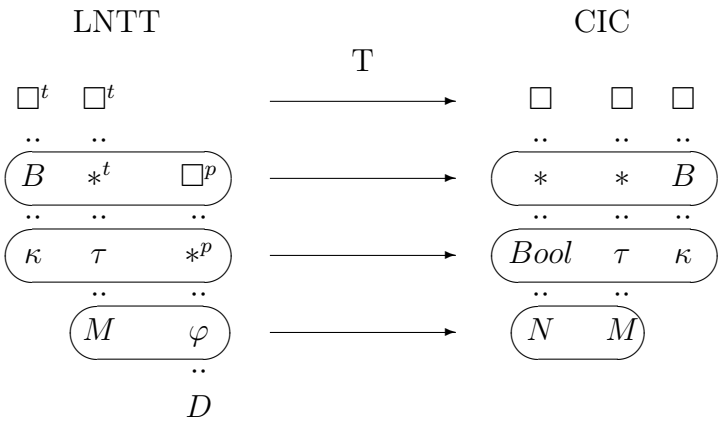
Wniosek z tego przykładu jest taki, ze LNTT jest nieporównywalna z CIC. Istnieją funkcje, które można zdefiniować w LNTT, ale nie w CIC i na odwrót. Jednak nawet bez silnej eliminacji możemy definiować funkcje, do których jest ona potrzebna w innych systemach.

5 Główny wynik: twierdzenie o silnej normalizacji

Główny wynik pracy to twierdzenie o silnej normalizacji LNTT z typami indukcyjnymi. Konsekwencją tego twierdzenia jest niesprzeczność całego systemu.

Dowód silnej normalizacji składa się z dwóch części. Część pierwsza polega na pokazaniu translacji T z LNTT do rachunku konstrukcji indukcyjnych. Podstawowy pomysł to przetłumaczenie sortu $*^p$ na specjalny typ *Bool*. W konsekwencji formuły zostają przetłumaczone na termy typu *Bool*. Translacja nie obejmuje dowodów, ich tłumaczenie nie zmieściłoby się w hierarchii typów rachunku konstrukcji. Sort $*^t$ zostaje przetłumaczony na $*$. Na typach i obiektach translacja jest zdefiniowana syntaktycznie, zachowuje kształt termów. Dla formuł używamy specjalnych stałych do przetłumaczenia implikacji, kwantyfikatora ogólnego i kwantyfikatora wyższego rzędu. Schematycznie działanie translacji T przedstawia Rysunek 2.
Translacja zachowuje asercje typowe, to znaczy

$$\text{jeśli } \Gamma \vdash M : A, \text{ to } T(\Gamma) \vdash T(M) : T(A)$$



Rysunek 2: Translacja z LNTT do CIC

oraz redukcje, to znaczy

$$\text{jeśli } M \to_{\beta\iota} M', \text{ to } T(M) \to^+_{\beta\iota} T(M').$$

Wiadomo, że CIC ma własność silnej normalizacji [24]. Łatwo zatem wnioskujemy stąd, że termy, które nie są dowodami mają własność silnej normalizacji.

Dla LNTT bez typów indukcyjnych druga część dowodu silnej normalizacji to inna translacja t do rachunku konstrukcji indukcyjnych, tym razem obejmująca dowody. Szczegóły można znaleźć w [18]. Tej metody nie da się jednak rozszerzyć do LNTT z typami indukcyjnymi. Rozszerzenie translacji t w naturalny sposób powoduje, że słaba eliminacja na małych typach indukcyjnych jest tłumaczona na silną eliminację na dużych typach indukcyjnych, która jest w CIC niedozwolona. Istnienie lepszej translacji do CIC jest otwartym problemem, dlatego druga część dowodu została przeprowadzona metodą kandydatów Girarda [15], a właściwie jej typowalną odmianą wprowadzoną przez Galliera i Coquanda [8]. Pomysły wykorzystane w dowodzie pochodzą z [14], [23], [24].

Dowód metodą kandydatów przebiega następująco. Zaczynamy od zdefiniowania rodziny C zbiorów termów spełniających pewne warunki domkniętości. Używa się kilku rodzajów tych warunków. Jeśli używamy oryginalnych własności Girarda, to zbiory w C nazywamy *kandydatami* (z franc. *candidats de reducibilité*). Stąd pochodzi nazwa metody. W naszym dowodzie używamy nieco innego zestawu warunków, a zbiory, które rozważamy noszą nazwę *zbiorów nasyconych*. W pewnym uproszczeniu warunki wyglądają tak: zbiór X jest nasycony, jeśli

- każdy term w X jest silnie normalizowalny;
- wszystkie zmienne należą do X;
- jeśli $M \in X$ oraz $M' \to_k M$ i M' jest silnie normalizowalny, to $M' \in X$. Relacja $\to_k$ to pewien podzbiór relacji redukcji $\to_{\beta\iota}$.

Następnie każdej zmiennej typowej przypisujemy pewien zbiór nasycony, a każdemu typowi τ (także formule, rodzajowi, itp.) przypisujemy jego interpretację $[\tau]$, będącą pewnym zbiorem termów. Ściśle rzecz biorąc, zbiory $[\tau]$ są parametryzowane wartościowaniem ξ, przypisującym zbiory z C zmiennym typowym. Mamy zatem zbiory termów $[\tau]_\xi$. W kolejnym kroku pokazujemy, że każdy zbiór $[\tau]_\xi \in C$, oraz że zbiór wszystkich termów silnie normalizowalnych jest zbiorem nasyconym. Na koniec dowodzimy, że każdy term M typu τ spełnia warunek $M \in [\tau]_\xi$ dla pewnego ξ.

W naszym dowodzie używamy typowalnej wersji metody kandydatów. Zamiast zbiorów termów posługujemy się zbiorami asercji typowych postaci $(\Gamma \vdash M : \tau)$, dla ustalonego τ. Zamiast jednej rodziny zbiorów nasyconych C mamy rodzinę zbiorów nasyconych C_τ dla każdego typu τ. Interpretacja $[\Gamma \vdash \tau]_\xi$ jest zdefiniowana dla każdej asercji $\Gamma \vdash \tau : \kappa$, gdzie τ jest typem. Ponadto wartościowanie ξ musi być zgodne z kontekstem Γ. Poszczególne etapy dowodu przebiegają podobnie jak w wersji beztypowej.

Najważniejszą częścią dowodu metodą kandydatów jest podanie interpretacji dla każdego typu. W LNTT z typami indukcyjnymi musimy dodatkowo podać interpretację dla „dużych" obiektów – podzbiorów i obiektów indukcyjnych – ponieważ mogą one w systemie pełnić rolę konstruktorów typowych lub konstruktorów formuł.

Interpretacja jest zdefiniowana przez indukcję ze względu na strukturę termu. Osobno podajemy definicję dla produktu, abstrakcji, aplikacji, typów indukcyjnych, itd. Interpretacje dla produktu, abstrakcji i aplikacji są standardowe. Interpretacja produktu $\Gamma \vdash \Pi x : A.B$ to zbiór asercji $(\Delta \vdash M)$ takich, że dla każdego asercji $(\Delta' \vdash N) \in [\Gamma \vdash A]$ mamy $(\Delta' \vdash MN) \in [\Gamma, x : A \vdash B]$. Interpretacją abstrakcji $\Gamma \vdash (\lambda x{:}A.B)$ jest funkcja, która bierze dowolną interpretację a dla $\Gamma \vdash A$ i daje w wyniku interpretację dla B z a jako interpretacją zmiennej x. Interpretacja aplikacji AB to aplikacja interpretacji A do interpretacji B.

Niestandardowe są interpretacje dla typów indukcyjnych i predykatów indukcyjnych, obiektów indukcyjnych oraz eliminacji.

Interpretacja dla typów indukcyjnych to najmniejszy punkt stały pewnego operatora monotonicznego F. Nieformalnie działanie operatora można opisać następująco: jako argument bierze przybliżenie S interpretacji i daje w wyniku zbiór asercji $(\Delta \vdash M)$ spełniających warunek: jeśli $M \to_k^* \mathrm{Constr}(n, I)\vec{N}$, a n-ty konstruktor I jest typu $\Pi\vec{x} : \vec{\tau}.I$, to każdy term N_i należy do interpretacji typu τ_i. Oczywiście interpretacja τ_i stosowana w operatorze F używa jako znaczenia dla typu indukcyjnego I jego przybliżenia S.

Interpretacja dla predykatów indukcyjnych to najmniejszy punkt stały innego operatora monotonicznego H. Ten operator również bierze jako argument przybliżenie S interpretacji. Jako wynik daje zbiór asercji $(\Delta \vdash M)$, w których term M eliminuje się w sposób poprawny, tj. jeśli weźmiemy dowolną formułę Q w kontekście Δ i jej interpretację q oraz dowolne gałęzie eliminacji $\vec{f}$ poprawne dla predykatu indykcyjnego I, wyniku Q oraz ich interpretacji S i q, to $(\Delta \vdash \mathrm{Elim}(I, Q, M)\{\vec{f}\}) \in q$.

Definicje typów indukcyjnych i predykatów indukcyjnych są bardzo podobne, mimo to ich interpretacje różnią się. Zauważmy, że nie można użyć operatora F', podobnego do operatora F do zdefiniowania interpretacji dla predykatów indukcyjnych. System ty-

pów pozwala na polimorfizm na formułach. Zatem chcąc podać definicję F' musielibyśmy dostarczyć interpretację dla dowolnej formuły Q, w tym także dla formuły strukturalnie większej niż predykat indukcyjny, którym się zajmujemy. Wtedy jednak definicja operatora F' nie byłaby dobrze ufundowana. Podobnie nie można użyć operatora H' podobnego do operatora H dla zdefiniowania interpretacji typów indukcyjnych. Obiekty indukcyjne mogą podlegać eliminacji zależnej. Interpretacja q dla wyniku byłaby funkcją, argument byłby interpretacją dla eliminowanego obiektu M. Tej interpretacji nie da się określić, bo M może być zupełnie dowolnym termem typu I.

Interpretacją dla obiektu indukcyjnego M postaci $\mathrm{Constr}(n, I)\vec{N}$ jest ciąg postaci $\langle n, C_{j_1}, \ldots, C_{j_k}\rangle$, gdzie n jest liczbą naturalną, a C_{j_i} jest interpretacją dużego argumentu N_j. Jeśli M nie jest postaci $\mathrm{Constr}(n, I)\vec{N}$, ale jest $\beta\iota$-równy takiemu termowi, to interpretacja M jest równa interpretacji $\mathrm{Constr}(n, I)\vec{N}$. Część pierwsza dowodu – własność silnej normalizacji dla termów nie będących dowodami – jest potrzebna do uzasadnienia poprawności tej definicji. Wartość interpretacji M w pozostałych przypadkach jest równa $\langle 0\rangle$, ale tak naprawdę jej wartość nie ma znaczenia.

Musimy także podać interpretację dla tych eliminacji, w wyniku których powstają podzbiory. Podstawowe wymaganie jest takie, ze ta interpretacja ma zachowywać relację ι-redukcji, to znaczy w przypadku liczb naturalnych spełniać warunki

$$\begin{aligned}[\Gamma \vdash \mathrm{Elim}(Nat, Q, 0)\{f_0 \mid f_1\}] &= [\Gamma \vdash f_0]\\ [\Gamma \vdash \mathrm{Elim}(Nat, Q, Sn)\{f_0 \mid f_1\}] &= [\Gamma \vdash f_1 n \mathrm{Elim}(Nat, Q, n)\{f_0 \mid f_1\}].\end{aligned}$$

Oczywiście nie możemy powyższej własności przyjąć za definicję interpretacji dla eliminacji. Taka definicja nie byłaby dobrze ufundowana. W definicji używamy operatora G, który stara się obliczyć, jaki byłby wynik redukcji. Operator jako jeden z argumentów bierze przybliżenie interpretacji dla typu I. Rekurencyjne wywołanie operatora, konieczne do poprawnego obsłużenia argumentów rekurencyjnych, jako argument bierze wcześniejsze (mniejsze) przybliżenie interpretacji I. W ten sposób mamy gwarancję, że definicja operatora jest poprawna.

Literatura

[1] The Agda Wiki. `http://wiki.portal.chalmers.se/agda/`.

[2] The Coq proof assistant. `http://coq.inria.fr`.

[3] Epigram 2. `http://www.e-pig.org/`.

[4] Isabelle proof assistant. `http://www.cl.cam.ac.uk/research/hvg/isabelle/`.

[5] H.P. Barendregt. Lambda calculi with types. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume II, pages 117–309. Oxford University Press, 1992.

[6] S. Berardi. *Type dependence and constructive mathematics*. PhD thesis, University of Torino, 1990.

[7] T. Coquand. An analysis of Girard's paradox. In *Symposium on Logic in Computer Science*, pages 227–236. IEEE Computer Society Press, 1986.

[8] T. Coquand and J. Gallier. A proof of strong normalization for the theory of constructions using a Kripke-like interpretation. In *In Workshop on Logical Frameworks–Preliminary Proceedings*, 1990.

[9] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, 1988.

[10] T. Coquand and C. Paulin-Mohring. Inductively defined types. In P. Martin-Löf and G. Mints, editors, *Proceedings of Colog'88*, volume 417 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[11] J.-F. Dufourd and Y. Bertot. Formal study of plane Delaunay triangulation. In M. Kaufmann and L. C. Paulson, editors, *Interactive Theorem Proving 2010*, volume 6172 of *Lecture Notes in Computer Science*, pages 211–226. Springer-Verlag, 2010.

[12] Gonthier G. The four colour theorem: Engineering of a formal proof. In Deepak Kapur, editor, *Asian Symposium on Computer Mathematics 2007*, volume 5081 of *Lecture Notes in Computer Science*, page 333. Springer, 2007.

[13] H. Geuvers. Private communication. 2006.

[14] H. Geuvers and M.-J. Nederhof. Modular proof of strong normalization for the Calculus of Constructions. *Journal of Functional Programming*, 1(2):155–189, 1991.

[15] J. Y. Girard. Interprétation fonctionelle et élimination des coupures dans l'aritmétique d'ordre supérieur. Thèse d'Etat, Université Paris 7, 1972.

[16] G. Gonthier, A. Mahboubi, L. Rideau, E. Tassi, and L. Thery. A modular formalisation of finite group theory. In K. Schneider and J. Brand, editors, *Proceedings of TPHOLs 2007*, volume 4732 of *Lecture Notes in Computer Science*, pages 86–101. Springer-Verlag, 2007.

[17] A.J.C. Hurkens. A simplification of Girard's paradox. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Typed Lambda Calculi and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 266–278. Springer-Verlag, 1995.

[18] A. Kozubek and P. Urzyczyn. In the search of a naive type theory. In M. Miculan, I. Scagnetto, and F. Honsell, editors, *Types for Proofs and Programs 2007*, volume 4941 of *Lecture Notes in Computer Science*, pages 110–124. Springer-Verlag, 2008.

[19] P. Martin-Löf. A theory of types. Technical Report 71-3, University of Stockholm, 1971.

[20] C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In Bezem M. and Groote J. F., editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer-Verlag, 1993.

[21] M.H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Elsevier, 2006.

[22] J. Terlouw. Een nadere bewijstheoretische analyse van GSTT's. Manuscript (in Dutch), 1989.

[23] D. Walukiewicz. *Termination of Rewriting in the Calculus of Constructions*. PhD thesis, University of Warsaw and Université de Paris Sud, 2002.

[24] B. Werner. *Une Théorie des Constructions Inductives*. PhD thesis, Université Paris 7, 1994.