

Crafting Provers in Racket

absolvent: Adrian Manea
coordonator: Traian Șerbănuță

510, SLA

Introducere și motivație

Scurtă introducere în **teoria tipurilor** (MLTT) și verificări folosind **Racket**

MLTT pentru că folosește logica intuiționistă și stă la baza HoTT

Racket pentru că este pornit din Lisp (preferință personală) și creat expres pentru cercetări în limbaje de programare

- ➊ Elemente de bază pentru MLTT: λ -calcul și tipuri dependente
- ➋ PROUST, “*a nano proof assistant*”
- ➌ PIE pentru tipuri dependente

Teoria tipurilor în formularea P. Martin-Löf (~ 1980)

Dezvoltarea istorică ([Collins, 2012]):

- paradoxul lui Russell;
- probleme de limbaj: Frege, Wittgenstein, Carnap;
- ... corespondența Curry-Howard(-Lambek).

Distincții clare față de teoria mulțimilor \Rightarrow intenții fundamentale, accentuate de *teoria toposurilor* (categorii + logică)

Abordare intuïționistă:

- existențialii trebuie instanțiați finitist;
- terțul exclus dispare;
- demonstrații bazate pe judecăți și reguli de inferență.

Teoria tipurilor în formularea P. Martin-Löf (~ 1980)

Formulare (sintaxă) abstractă, interpretări (semantici) diverse:

- propoziții ca tipuri (Curry-Howard, Wadler et al.)
- categorii cartezian închise (Lambek)
- probleme și soluții (Brouwer-Heyting-Kolmogorov)

$A ?$	$a \in A$	interpretare
A mulțime	a element al A	A nevidă
A propoziție	a demonstrație constructivă a A	A este adevărată
A intenție	a metodă de a îndeplini A	A realizabilă
A problemă	a metodă de rezolvare a A	A are soluție

Ilustrație: Interpretări diverse ale $a \in A$, [Martin-Löf, 1980, p. 4]

Teoria tipurilor în formularea P. Martin-Löf (~ 1980)

Tipuri dependente: Apărute pentru interpretarea cuantificatorilor.

Expresii parametrizate, i.e. expresii „la fel” sintactic, cu interpretări diferite.

Judecăți primitive (stil Martin-Löf):

- A este un *tip corect format* în contextul Γ : $\Gamma \vdash A \text{ type}$;
- A și B sînt *tipuri identice* în contextul Γ : $\Gamma \vdash A \equiv B \text{ type}$;
- a este un *termen corect format*, de tip A , în contextul Γ : $\Gamma \vdash a : A$;
- a și b sînt *termeni egali* de tip A în contextul Γ : $\Gamma \vdash a \equiv b : A$.

Context = asumții de tipizare, de forma:

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

Lambda calcul fără tipuri (~ 1930)

Intuitiv: un formalism pentru „funcții anonime”.

Gramatica BNF:

$$t ::= x \mid \lambda x.t \mid tt$$

- variabile libere x ;
- lambda-abstracții (leagă variabila x în termenul t);
- aplicări ale unui termen pe alt termen.

Exemple: x , $\lambda x.5$, $(\lambda x.5)z$, $(\lambda x.5)(\lambda z.(z + 1))$ etc.

Astăzi, majoritatea limbajelor de programare au o formă de a declara funcții anonime cu expresii lambda.

Tipuri funcționale

A, B type; asociem elementele \Rightarrow *tipul funcțional* $A \rightarrow B$.

Inferențele pentru tipuri funcționale se fac cu *lambda calcul*.

$$\frac{\Gamma \vdash B \text{ type} \quad \Gamma, x : A \vdash b(x) : B}{\Gamma \vdash \lambda x. b(x) : A \rightarrow B} \quad (\lambda)$$

$$\frac{\Gamma \vdash f : A \rightarrow B}{\Gamma, x : A \vdash f(x) : B} \quad (\text{eval})$$

$$\frac{\Gamma \vdash B \text{ type} \quad \Gamma, x : A \vdash b(x) : B}{\Gamma, x : A \vdash (\lambda y. b(y))(x) \equiv b(x) : B} \quad (\beta)$$

Tipuri dependente: Π (produs)

Ideea: *familii de tipuri*, parametrizate de un tip anume.

Pentru fiecare $x : A$, asociem *cîte un tip* $B(x)$.

Notația $\prod_{(x:A)} B(x)$, numite și *funcții dependente*.

Interpretare: cuantificarea universală, $(\forall x : A) B(x) \rightsquigarrow \prod_{(x:A)} B(x)$.

Exemplu: **numerele naturale**. Fie $0 : \mathbb{N}$ și $S : \mathbb{N} \rightarrow \mathbb{N}$.

Principiul de inducție, ca regulă de inferență:

$$\frac{\Gamma, n : \mathbb{N} \vdash P \text{ type} \quad \Gamma \vdash p_0 : P(0) \quad \Gamma \vdash p_S : \prod_{(n:\mathbb{N})} (P(n) \rightarrow P(S(n)))}{\Gamma \vdash \text{ind}_{\mathbb{N}}(p_0, p_S) : \prod_{(n:\mathbb{N})} P(n)} (\mathbb{N} - \text{ind}).$$

Tipuri dependente: Σ (sumă)

Ideea: Pentru $a : A$, asociem câte un tip $B(a) \Rightarrow \sum_{(a:A)} B(a)$

Interpretare: cuantificarea existențială, $(\exists x : A) B(x) \rightsquigarrow \sum_{(x:A)} B(x)$.

Exemplu: Fie $x : \mathbb{N}$ fixat. Definim predicatul $P(y) : d \cdot y = x$.

Formăm tipul sumă $\sum_{(y:\mathbb{N})} P(y)$, populat de:

- elemente de tip $y : \mathbb{N}$;
- pentru fiecare y , demonstrații ale $P(y)$, i.e. câte un d , *dacă există*.

Demonstrator DIY, cu accent didactic.

Verifică tipizarea (în context) pentru:

- tipizări explicite (*type annotations*), `Ann (expr type);`
- tipuri funcționale, `Arrow (domain codomain);`
- aplicări de funcții, `App (func arg);`
- expresii lambda, `Lam (var body);`

Exemplu:


```
(check-expect (check-proof '((lambda x => x) : (A -> A))) true)
```


Limbaj specific (DSL), cu accent pe *tipizare statică* și *tipuri dependente*.

Implementat inițial în Racket, mutat apoi în Haskell (pie-hs).

Exemple:

- Rezervarea tipului: `(claim one Nat);`
- Definiția: `(define one (add1 zero));`
- Tipizarea explicită: `(the Nat (add1 one));`
- Tipuri Π : `(Pi ((A U) (D U)) (-> (Pair A D) (Pair D A)))`.

 (2020).
The PLT Group.
<https://racket-lang.org/people.html>.
Accessed: March 2020.

 (2020).
Racket.
<https://racket-lang.org/>.
Accessed: March 2020.

 Collins, J. (2012).
A History of the Theory of Types.
Lambert Academic Publishing.

 Felleisen, M. and Findler, B. (2014).
How to Design Programs.
MIT Press.



Friedman, D. and Christiansen, D. (2018).
The Little Typer.
MIT Press.



Group, T. H. (2013).
Homotopy Type Theory.
<https://homotopytypetheory.org/>.
Accessed March 2020.



Martin-Löf, P. (1980).
Intuitionistic Type Theory.
Lectures in Padua.
Notes by Giovanni Sambin.



Nordström, B. and Petersson, K. (1990).
Programming in Martin-Löf Type Theory.
Oxford University Press.

Freely available at

<http://www.cse.chalmers.se/research/group/logic/book/>.



Ragde, P. (2016).

Proust: A Nano Proof Assistant.

In Jeuring, J. and McCarthy, J., editors, *Trends in Functional Programming in Education*, pages 63–75. arXiv/cs.PL.