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**

Introducere și motivație

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

MLTT pentru că folosește logica intuiționistă și stă la baza HoTT

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

Plan

1. Elemente de bază pentru MLTT: λ-calcul și tipuri dependente

Plan

1. Elemente de bază pentru MLTT: λ-calcul și tipuri dependente
2. PROUST, *“a nano proof assistant”*

Plan

1. Elemente de bază pentru MLTT: λ-calcul și tipuri dependente
2. PROUST, *"a nano proof assistant"*
3. PIE pentru tipuri dependente

Teoria tipurilor în formularea P. Martin-Löf ($\sim$1980)

Dezvoltarea istorică ([Collins, 2012]):

Teoria tipurilor în formularea P. Martin-Löf ($\sim$1980)

Dezvoltarea istorică ([Collins, 2012]):

- paradoxul lui Russell;

Teoria tipurilor în formularea P. Martin-Löf ($\sim$1980)

Dezvoltarea istorică ([Collins, 2012]):

- paradoxul lui Russell;
- probleme de limbaj: Frege, Wittgenstein, Carnap;

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).

Teoria tipurilor în formularea P. Martin-Löf ($\sim$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)

Teoria tipurilor în formularea P. Martin-Löf ($\sim$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 intuiționistă:

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 intuiționistă:

- existențialii trebuie instanțiați finitist;

Teoria tipurilor în formularea P. Martin-Löf ($\sim$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 intuiționistă:

- existențialii trebuie instanțiați finitist;
- terțul exclus dispare;

Teoria tipurilor în formularea P. Martin-Löf ($\sim$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 intuiționistă:

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

Teoria tipurilor în formularea P. Martin-Löf ($\sim$1980)

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

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

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

- propoziții ca tipuri (Curry-Howard, Wadler et al.)

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)

Teoria tipurilor în formularea P. Martin-Löf ($\sim$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)

Teoria tipurilor în formularea P. Martin-Löf ($\sim$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]

Tipuri dependente:

Teoria tipurilor în formularea P. Martin-Löf ($\sim$1980)

Tipuri dependente: Apărute pentru interpretarea cuantificatorilor.

Teoria tipurilor în formularea P. Martin-Löf ($\sim$1980)

Tipuri dependente: Apărute pentru interpretarea cuantificatorilor.
Expresii parametrizate, i.e. expresii „la fel” sintactic, cu interpretări diferite.

Tipuri dependente: Apărute pentru interpretarea cuantificatorilor.
Expresii parametrizate, i.e. expresii „la fel” sintactic, cu interpretări diferite.

Judecăți primitive (stil Martin-Löf):

Teoria tipurilor în formularea P. Martin-Löf ($\sim$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: $\Gamma \vdash A$ type;

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: $\Gamma \vdash A$ type;
- A și B sînt *tipuri identice* în contextul Γ: $\Gamma \vdash A \equiv B$ type;

Teoria tipurilor în formularea P. Martin-Löf ($\sim$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: $\Gamma \vdash A$ type;
- A și B sînt *tipuri identice* în contextul Γ: $\Gamma \vdash A \equiv B$ type;
- a este un *termen corect format*, de tip A, în contextul Γ: $\Gamma \vdash a : A$;

Teoria tipurilor în formularea P. Martin-Löf ($\sim$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: $\Gamma \vdash A$ type;
- A și B sînt *tipuri identice* în contextul Γ: $\Gamma \vdash A \equiv B$ 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: $\Gamma \vdash a \equiv b : A$.

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: $\Gamma \vdash A$ type;
- A și B sînt *tipuri identice* în contextul Γ: $\Gamma \vdash A \equiv B$ 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 = asumpții de tipizare, de forma:

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

Lambda calcul fără tipuri ($\sim$1930)

Intuitiv: un formalism pentru „funcții anonime”.

Lambda calcul fără tipuri ($\sim$1930)

Intuitiv: un formalism pentru „funcții anonime”.

Gramatica BNF:

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

Intuitiv: un formalism pentru „funcții anonime”.

Gramatica BNF:

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

- variabile libere x;

Lambda calcul fără tipuri ($\sim$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);

Lambda calcul fără tipuri ($\sim$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.

Lambda calcul fără tipuri ($\sim$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.

Lambda calcul fără tipuri ($\sim$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* $A \to B$.

Tipuri funcționale

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

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

Tipuri funcționale

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

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

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

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

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

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

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

Tipuri funcționale

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

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

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

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

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

Tipuri dependente: Π (produs)

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

Tipuri dependente: Π (produs)

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

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

Tipuri dependente: Π (produs)

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

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

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

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

Tipuri dependente: Π (produs)

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

Pentru fiecare $x : A$, asociem *câte 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 $S : \mathbb{N} \to \mathbb{N}$.

Tipuri dependente: Π (produs)

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

Pentru fiecare $x : A$, asociem *câte 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 $S : \mathbb{N} \to \mathbb{N}$.

Principiul de inducție, ca regulă de inferență:

Tipuri dependente: Π (produs)

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

Pentru fiecare $x : A$, asociem *câte 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 $S : \mathbb{N} \to \mathbb{N}$.

Principiul de inducție, ca regulă de inferență:

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

Tipuri dependente: Σ (sumă)

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

Tipuri dependente: Σ (sumă)

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

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

Tipuri dependente: Σ (sumă)

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

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

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

Tipuri dependente: Σ (sumă)

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

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

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:

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

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

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 $y : \mathbb{N}$;

Tipuri dependente: Σ (sumă)

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

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

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. cîte un d, *dacă există.*

Proust (2016)

Demonstrator DIY, cu accent didactic.

Proust (2016)

Demonstrator DIY, cu accent didactic.

Verifică tipizarea (în context) pentru:

Proust (2016)

Demonstrator DIY, cu accent didactic.

Verifică tipizarea (în context) pentru:

- tipizări explicite (*type annotations*), `Ann (expr type)`;

Proust (2016)

Demonstrator DIY, cu accent didactic.

Verifică tipizarea (în context) pentru:

- tipizări explicite (*type annotations*), `Ann (expr type)`;
- tipuri funcționale, `Arrow (domain codomain)`;

Proust (2016)

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)`;

Proust (2016)

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)`;

Proust (2016)

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)
```

Pie (2018)

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

Pie (2018)

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

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

Pie (2018)

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

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

Exemple:

Pie (2018)

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)`;

Pie (2018)

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));`

Pie (2018)

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))`;

Pie (2018)

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)))`.

Bibliografie și lecturi suplimentare I

(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.

Bibliografie și lecturi suplimentare III

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.