Formal systems project: Implementing a dependently-typed calculus

Jesper Cockx

KU Leuven

4 november 2014

Formal systems project

Intro: waarvoor zijn types nuttig?

Bepaalde klasse van fouten statisch uitsluiten.

Wat voor fouten?

- Lijst waar een getal wordt verwacht?
- Lijst van Ints waar een lijst van Bools wordt verwacht?
- Lijst van lengte 0 waar een niet-lege lijst wordt verwacht?

Wat zijn dependent types?

Types kunnen afhankelijk zijn van *termen*.

Vb: Vec A n is het type van lijsten van lengte n.

$head : (n : \mathrm{Nat}) \to Vec\ A\ (1 + n) \to A$
kan niet worden toegepast op een lege lijst!

Waarom dependent types?

Dependent types laten toe om willekeurige eigenschappen van een programma uit te drukken in het type.

Vb: $\mathit{sort} : \text{List} \to \Sigma[x : \text{List}]\ (\text{Sorted}\ \ x)$
geeft een lijst terug, *en* een bewijs dat deze lijst gesorteerd is.

Talen met dependent types

Programmeertalen: Agda, Idris, F*, (binnenkort) Haskell, ...

Bewijsassistenten: Coq, NuPRL, Dedukti, ...

De Curry-Howard-overeenkomst

- Types komen overeen met *proposities*.
- Een term van een bepaald type komt overeen met een *bewijs* van die propositie.

Propositielogica en de STLC

- Een bewijs van $A \Rightarrow B$ is een functie die een bewijs van A afbeeldt op een bewijs van B.
- Een bewijs van $A \wedge B$ is een koppel (p, q) waar p een bewijs is van A en q van B.

Predikatenlogica en dependent types

- Een bewijs van $\forall x : A.\ P(x)$
  is een (dependent!) functie
  die $x : A$ afbeeldt op een bewijs van $P(x)$.
- Een bewijs van $\exists x : A.\ P(x)$
  is een (dependent!) koppel (a, p)
  waar $a : A$ en p een bewijs is van $P(a)$.

Voorbeeld: Even natural numbers

$$
\begin{aligned}
&\mathit{data}\quad \mathit{Even} : (n : \mathbb{N}) \to \mathit{Set}\ \mathit{where}\\
&\qquad \text{even-zero} : \mathit{Even}\ \mathit{zero}\\
&\qquad \text{even-ss} : (n : \text{Nat}) \to \mathit{Even}\ n \to \mathit{Even}\ (\mathit{suc}\ (\mathit{suc}\ n))
\end{aligned}
$$

$$\text{double-even} : (n : \mathbb{N}) \to \mathit{Even}\ (\mathit{double}\ n)$$

Formal systems project

Syntax

$s, t ::=$		**terms**
	x	variables
	$\lambda x : t.t$	abstraction
	$t\ t$	application
	$(x : t) \rightarrow t$	dep. function type
	Set	type of types

Contexten met dependent types

Types in een context kunnen afhankelijk zijn van voorgaande variabelen:

$$(n : Nat) \ldots (v : Vec\ Bool\ n) \ldots$$

Type-regel voor variabelen:

$$\frac{x : t \in \Gamma}{\Gamma \vdash x : t} \qquad (\textsc{T-Var})$$

Set: het type der types

$$\frac{}{\Gamma \vdash \mathsf{Set} : \mathsf{Set}} \qquad (\textsc{T-SetInSet})$$

Abstracties en applicaties

$$\frac{\Gamma \vdash A : \mathsf{Set} \qquad \Gamma, x : A \vdash B : \mathsf{Set}}{\Gamma \vdash (x : A) \to B : \mathsf{Set}} \quad (\textsc{T-Pi})$$

$$\frac{\Gamma \vdash A : \mathsf{Set} \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : (x : A) \to B} \quad (\textsc{T-Abs})$$

$$\frac{\Gamma \vdash t_1 : (x : A) \to B \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1\ t_2 : B[x \mapsto t_2]} \quad (\textsc{T-App})$$

Sigma types

$$\frac{\Gamma \vdash A : Set \qquad \Gamma(x : A) \vdash B : Set}{\Gamma \vdash \Sigma[x : A]\ B : Set} \quad (\textsc{T-Sigma})$$

$$\frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : [x \mapsto s]B}{\Gamma \vdash (s, t) : \Sigma[x : A]\ B} \quad (\textsc{T-Pair})$$

$$\frac{\Gamma \vdash t : \Sigma[x : A]\ B}{\Gamma \vdash \mathsf{fst}\ \ t : A} \quad (\textsc{T-Fst})$$

$$\frac{\Gamma \vdash t : \Sigma[x : A]\ B}{\Gamma \vdash \mathsf{snd}\ \ t : [x \mapsto \mathsf{fst}\ \ t]B} \quad (\textsc{T-Snd})$$

Identity types

Optionele opgave, zie Toledo.

Formal systems project

Wat loopt er mis?

$$\begin{aligned} \mathit{foo} &: (b : \mathsf{Bool}) \to \mathsf{if}\ b\ \mathsf{then}\ \mathsf{Nat}\ \mathsf{else}\ \mathsf{Bool} \\ \mathit{bar} &: \mathsf{Bool} \\ \mathit{bar} &= \mathit{not}\ (\mathit{foo}\ \ \mathsf{false}) \end{aligned}$$

Type error: $\mathsf{if}\ \mathsf{false}\ \mathsf{then}\ \mathsf{Nat}\ \mathsf{else}\ \mathsf{Bool} \neq \mathsf{Bool}$.

Aangepaste typeregel voor applicaties

$$\frac{\Gamma \vdash t_1 : A \qquad A \longrightarrow^* ((x : B) \to C) \qquad \Gamma \vdash t_2 : B}{\Gamma \vdash t_1\ t_2 : C[x \mapsto t_2]}$$

Evaluatie tijdens typechecking

Mogelijke problemen:

- Zij-effecten (IO, mutable state, ...): zijn niet (rechtstreeks) toegelaten.
- Non-terminatie: net als de STLC is onze taal *totaal* (alle programma's zijn eindig)

Typechecking vs type inference

Type *checking*:

- Check term tegen een gegeven type
- Eenvoudig, maar veel schrijfwerk

Type *inference*:

- Leid type af uit structuur van de term
- Handig, maar niet altijd mogelijk

Bidirectionele typechecking

Idee: typechecker kan wisselen tussen *checking mode* en *inference mode*.

$\Gamma \vdash t \Uparrow A$: type A is gereconstrueerd uit t.

$\Gamma \vdash t \Downarrow A$: t typecheckt als term van type A.

Bidirectionele regel voor applicatie

$$\frac{\Gamma \vdash t_1 \Uparrow A \qquad A \longrightarrow^* ((x : B) \to C) \qquad \Gamma \vdash t_2 \Downarrow B}{\Gamma \vdash t_1 \; t_2 \Uparrow C[x \mapsto t_2]}$$

Formal systems project

Benodigdheden voor het project

- Scala (versie 2.11.2)
- ScalaIDE (versie 3.0.4)
- ScalaTest for ScalaIDE plugin (versie 2.9.3)

ScalaIDE update site:

`http://download.scala-ide.org/sdk/helium/e38/scala211/stable/site`

Structuur van de code

- DepParser.scala
- Syntax.scala
- Evaluator.scala
- Typer.scala
- DepCalculus.scala
- DepTest.scala

Indienen

Code: via Toledo

Geschreven opgave:

PDF via Toledo of op secretariaat