Implementing Dependent Types using Sequent Calculi

Daniel Gustafsson Nicolas Guenot

IT University of Copenhagen
{dagu,ngue}@itu.dk

Types, 2015

What and how do Coq and Agda differ?

- Tactics vs. Code.
- The types accepted and Universes, Agda don't have Prop for example.
- Agda definitions written in an equational style, whereas Coq uses `match`-statements. [FOCUS OF THIS TALK]

The last point we claim is similar to the difference between Natural Deduction vs. Sequent Calculus.

Equational style corresponds to sequent calculus

Left rules (for positive connectives) corresponds to splitting in the context[1].

Lets see a small example!

[1] Serenella Cerrito and Delia Kesner. "Pattern Matching as Cut Elimination". In: *LICS'99*. 1999, pp. 98–108.

Example in Agda and sequent calculus

```
f : (ℕ × ℕ) ⊎ ℕ → ℕ
f arg = ?
```

$$arg : (\mathbb{N} \times \mathbb{N}) \uplus \mathbb{N} \vdash \mathbb{N}$$

Example in Agda and sequent calculus

```
f : (ℕ × ℕ) ⊎ ℕ → ℕ
f (inl a) = ?
f (inr z) = ?
```

$$\frac{a : \mathbb{N} \times \mathbb{N} \vdash \mathbb{N} \qquad z : \mathbb{N} \vdash \mathbb{N}}{arg : (\mathbb{N} \times \mathbb{N}) \uplus \mathbb{N} \vdash \mathbb{N}}$$

Example in Agda and sequent calculus

```
f : (ℕ × ℕ) ⊎ ℕ → ℕ
f (inl (x,y)) = ?
f (inr z) = z
```

$$\dfrac{\dfrac{x : \mathbb{N}, y : \mathbb{N} \vdash \mathbb{N}}{a : \mathbb{N} \times \mathbb{N} \vdash \mathbb{N}} \qquad \dfrac{}{z : \mathbb{N} \vdash z : \mathbb{N}}}{arg : (\mathbb{N} \times \mathbb{N}) \uplus \mathbb{N} \vdash \mathbb{N}}$$

Example in Agda and sequent calculus

```
f : (ℕ × ℕ) ⊎ ℕ → ℕ
f (inl (x,y)) = x + y
f (inr z) = z
```

$$\dfrac{\dfrac{\dfrac{}{x:\mathbb{N}, y:\mathbb{N} \vdash x+y:\mathbb{N}}}{a:\mathbb{N}\times\mathbb{N}\vdash\mathbb{N}} \qquad \dfrac{}{z:\mathbb{N}\vdash z:\mathbb{N}}}{arg:(\mathbb{N}\times\mathbb{N})\uplus\mathbb{N}\vdash\mathbb{N}}$$

Case splitting trees in Agda

- Have basically the same information as seen on previous slide.
- Used internally in Agda for computation.
- Similar in structure to coverage checking.
- Currently not expressed as terms in some calculus.

Outline

There is a connection between Agda and Focused Sequent Calculus, we will explain this connection.

1. We first present the propositional fragment, which matches closely with Logic.
2. We will then update this to dependent types, this raises some questions but we have a proposal how to solve these.
3. Finally we try to look for a way of explaining induction.

Outline

There is a connection between Agda and Focused Sequent Calculus, we will explain this connection.

1. We first present the propositional fragment, which matches closely with Logic.
2. We will then update this to dependent types, this raises some questions but we have a proposal how to solve these.
3. Finally we try to look for a way of explaining induction.

By making the connection we hope to achieve:

- Better ways of implementing Agda.
- Getting a better understanding in Proof Theory of for example equality and inductive reasoning.

Propositional fragment

Implication in Sequent Calculus:

$$\frac{}{\Psi, N \vdash N} \qquad \frac{\Psi \vdash N_0 \qquad \Psi, N_1 \vdash M}{\Psi, N_0 \rightarrow N_1 \vdash M}$$

$$\frac{\Psi, N \vdash M}{\Psi \vdash N \rightarrow M} \qquad \frac{\Psi, N, N \vdash M}{\Psi, N \vdash M}$$

Propositional fragment

Implication in Focused Sequent Calculus:

$$\frac{}{\Psi, [N] \vDash N} \qquad \frac{\Psi \vdash N_0 \qquad \Psi, [N_1] \vDash M}{\Psi, [N_0 \to N_1] \vDash M}$$

$$\frac{\Psi, N \vdash M}{\Psi \vdash N \to M} \qquad \frac{\Psi, N, [N] \vDash M}{\Psi, N \vdash M}$$

Propositional fragment

Term assignment gives Hugo's $\bar{\lambda}^2$, focus gives spines.

$$\frac{}{\Psi, [N] \vDash \varepsilon : N} \qquad \frac{\Psi \vdash t : N_0 \qquad \Psi, [N_1] \vDash k : M}{\Psi, [N_0 \to N_1] \vDash d :: k : M}$$

$$\frac{\Psi, x : N \vdash t : M}{\Psi \vdash \lambda x.\, t : N \to M} \qquad \frac{\Psi, x : N, [N] \vDash k : M}{\Psi, x : N \vdash x\ k : M}$$

In fact, Agda is already uses spines internally!

---

[2] Hugo Herbelin. "A λ-Calculus Structure Isomorphic to Gentzen-Style Sequent Calculus Structure". In: *CSL'94*. Ed. by L. Pacholski and J. Tiuryn. Vol. 933. LNCS. 1994, pp. 61–75.

Propositional fragment (II)

But $\bar{\lambda}$ only have negatives, so we add positives to explain patterns.

Judgement for left hand side (patterns):

- $\Psi \mid \Gamma \vdash t : N$ inversion phase. Γ is the inversion context which contains patterns, Ψ are for global variables.

Judgement for right hand side:

- $\Psi, [N] \vDash k : M$ left focus, we are eliminating a term of type N to a term of type M.
- $\Psi \vDash d : [P]$ right focus, we are constructing data.

(Agda don't have t-terms so she is missing the first judgement.)

Propositional fragment – Full system

$$\frac{}{\Psi, [N] \vDash \varepsilon : N} \qquad \frac{\Psi \vDash d : [P]}{\Psi \mid \cdot \vdash \triangleright\, d : \uparrow P} \qquad \frac{\Psi \mid \cdot \vdash t : N}{\Psi \vDash \triangleleft\, t : [\downarrow N]}$$

$$\frac{\Psi, x : \downarrow N, [N] \vDash k : M}{\Psi, x : \downarrow N \mid \cdot \vdash x\ k : M} \qquad \frac{\Psi, x : \downarrow N \mid \Gamma \vdash t : M}{\Psi \mid \Gamma, x : \downarrow N \vdash t : M} \qquad \frac{\Psi \mid p : P \vdash t : N}{\Psi, [\uparrow P] \vDash \kappa p.t : N}$$

$$\frac{\Psi \mid \Gamma, p : P \vdash t : N}{\Psi \mid \Gamma \vdash \lambda p.\, t : P \to N} \qquad \frac{\Psi, [N] \vDash k : L}{\Psi, [N \wedge M] \vDash .\mathtt{fst}\ k : L} \qquad \frac{\Psi, [M] \vDash k : L}{\Psi, [N \wedge M] \vDash .\mathtt{snd}\ k : L}$$

$$\frac{\Psi \vDash d : [P] \qquad \Psi, [N] \vDash k : M}{\Psi, [P \to N] \vDash d :: k : M} \qquad \frac{\Psi \mid \Gamma \vdash t : N \qquad \Psi \mid \Gamma \vdash u : M}{\Psi \mid \Gamma \vdash \langle t, u \rangle : N \wedge M}$$

$$\frac{\Gamma \vDash d : [P] \qquad \Gamma \vDash e : [Q]}{\Gamma \vDash (d, e) : [P \times Q]} \qquad \frac{\Gamma \vDash d : [P]}{\Gamma \vDash \mathtt{inl}\ d : [P \vee Q]} \qquad \frac{\Gamma \vDash d : [Q]}{\Gamma \vDash \mathtt{inr}\ d : [P \vee Q]}$$

$$\frac{\Psi \mid \Gamma, p : P, q : Q \vdash t : N}{\Psi \mid \Gamma, (p, q) : P \times Q \vdash t : N} \qquad \frac{\Psi \mid \Gamma, p : P \vdash t : N \qquad \Psi \mid \Gamma, q : Q \vdash u : N}{\Psi \mid \Gamma, x[p \mid q] : P \vee Q \vdash x[t \mid u] : N}$$

Propositional fragment – Constructors

Data constructors can be seen as $\vee$ with arguments of $\times$.

$$\frac{\Psi \mid \Gamma, p : P, q : Q \vdash t : N}{\Psi \mid \Gamma, (p, q) : P \times Q \vdash t : N} \qquad \frac{\Psi, x : \downarrow N \mid \Gamma \vdash t : M}{\Psi \mid \Gamma, x : \downarrow N \vdash t : M}$$

$$\frac{\Psi \mid \Gamma, p : P \vdash t : N \qquad \Psi \mid \Gamma, q : Q \vdash u : N}{\Psi \mid \Gamma, x[p \mid q] : P \vee Q \vdash x[t \mid u] : N}$$

Propositional fragment - Focus

Focusing rules are the borders(the $=$-sign in the definition) between pattern matching and right hand side.

$$\frac{\Psi, x : \downarrow N, [N] \vDash k : M}{\Psi, x : \downarrow N \mid \cdot \vdash x\ k : M} \qquad \frac{\Psi \vDash d : [P]}{\Psi \mid \cdot \vdash \triangleright\ d : \uparrow P}$$

Propositional fragment – Copatterns[3]

Records can be constructed using copatterns:

$$
\begin{aligned}
&\mathtt{swap} : (\mathtt{A} \wedge \mathtt{B}) \rightarrow \mathtt{B} \wedge \mathtt{A} \\
&\mathtt{fst}\ (\mathtt{swap}\ \mathtt{x}) = \mathtt{snd}\ \mathtt{x} \\
&\mathtt{snd}\ (\mathtt{swap}\ \mathtt{x}) = \mathtt{fst}\ \mathtt{x}
\end{aligned}
$$

$$
\frac{\Psi \mid \Gamma \vdash t : N \qquad \Psi \mid \Gamma \vdash u : M}{\Psi \mid \Gamma \vdash \langle t, u \rangle : N \wedge M}
$$

$$
\frac{\Psi, [N] \vDash k : L}{\Psi, [N \wedge M] \vDash .\mathtt{fst}\ k : L}
\qquad
\frac{\Psi, [M] \vDash k : L}{\Psi, [N \wedge M] \vDash .\mathtt{snd}\ k : L}
$$

[3] Andreas Abel et al. "Copatterns: programming infinite structures by observations". In: *POPL'13*. 2013, pp. 27–38.

The Cuts of the System

There are two focused cuts in this system.

```
filter : (A → Bool) → List A → List A
filter p nil = nil
filter p (cons x xs) with p x
filter p (cons x xs) | false = filter p xs
filter p (cons x xs) | true  = cons x (filter p xs)
```

The first one corresponds to `with` in Agda, which adds a new term to pattern match on. Similar to pattern guard in Haskell.

$$\frac{\Psi \vDash d : [P] \qquad \Psi \mid \Gamma, p : P \vdash t : N}{\Psi \mid \Gamma \vdash p = d \;\mathtt{in}\; t : N}$$

The Cuts of the System (II)

The second of the cuts corresponds calling a previously defined function.

$$\frac{\Psi \mid \Gamma \vdash \color{red}{t} : N \qquad \Psi, [N] \vDash \color{red}{k} : M}{\Psi \mid \Gamma \vdash \color{red}{t\ k} : M}$$

Think of t as a constant for a previously defined function.

Outline (II)

- That concludes the propositional fragment, which have been very principled way of building the system.
- We will now add dependent types, to make it closer to Agda.
- Prior work exists[4], which is also based on $\bar{\lambda}$, though it only handles Π.
- Question is how to generalise the patterns.

[4] Stéphane Lengrand, Roy Dyckhoff, and James McKinna. "A Focused Sequent Calculus Framework for Proof Search in Pure Type Systems". In: *Logical Methods in Computer Science* 7.1 (2011).

Making it with dependent types

Here Σ is the generalises $\times$.

$$\frac{\Psi \mid \Gamma, x : P \vdash t : N}{\Psi \mid \Gamma \vdash \lambda x.\, t : \Pi(x : P).N} \qquad \frac{\Psi \vDash d : [P] \qquad \Psi, [N\{d/x\}] \vDash k : M}{\Psi, [\Pi(x : P).N] \vDash d :: k : M}$$

$$\frac{\Psi \mid \Gamma, y : P, z : Q \vdash t : N\{(y,z)/x\}}{\Psi \mid \Gamma, x : \Sigma(y : P).Q \vdash y, z = x \;\texttt{in}\; t : N} \qquad \frac{\Gamma \vDash d : [P] \qquad \Gamma \vDash e : [Q\{d/x\}]}{\Gamma \vDash (d, e) : [\Sigma(x : P).Q]}$$

$$\frac{\Psi \mid \Gamma, y : P \vdash t : N\{\texttt{inl}\ y/x\} \qquad \Psi \mid \Gamma, z : Q \vdash u : N\{\texttt{inr}\ z/x\}}{\Psi \mid \Gamma, x : P \vee Q \vdash x[y.t \mid z.u] : N}$$

Making it with dependent types (II)

It is unclear what the proper solution for eliminating dependent records, and therefore how copatterns become dependent.

$$\frac{\Psi, [M\{?/x\}] \vDash k : L}{\Psi, [\Sigma(x : N).M] \vDash \texttt{.snd}\ k : L}$$

Prior work exists[5] but this basically defines a natural deduction system and converts between them. We need some form of zipper.

[5] Roy Dyckhoff and Luís Pinto. "Sequent Calculi for the Normal Terms of the $\lambda\Pi$- and $\lambda\Pi\Sigma$-Calculi". In: *Electronic Notes in Theoretical Computer Science* 17 (1998), pp. 1–14.

Induction

$$
\begin{aligned}
\mathtt{ind}_{\mathbb{N}} &: \mathtt{P\,zero} \rightarrow ((\mathtt{x} : \mathbb{N}).\, \mathtt{P\,x} \rightarrow \mathtt{P\,(suc\,x)}) \rightarrow (\mathtt{n} : \mathbb{N}).\, \mathtt{P\,n} \\
\mathtt{ind}_{\mathbb{N}}\ \mathtt{base\ ih\ zero} &= \mathtt{base} \\
\mathtt{ind}_{\mathbb{N}}\ \mathtt{base\ ih\ (suc\ n)} &= \mathtt{ih\ n}\ (\mathtt{ind}_{\mathbb{N}}\ \mathtt{base\ ih\ n})
\end{aligned}
$$

How do we handle induction?

- Agda uses an external termination checker, and provide simple definitions.
- It would be cool if induction[6] from proof theory would coincide with induction principles from dependent type theory.

[6] David Baelde. "Least and Greatest Fixed Points in Linear Logic". In: *ACM Transactions on Computational Logic* 13.1 (2012), p. 2.

Induction (II)

But the "natural" μ-rules seems to not be compatible with dependencies in the types.

$$\frac{\Gamma \vdash B\{\mu a.B/a\}}{\Gamma \vdash \mu a.B} \qquad \frac{B\{C\{?/x\}/a\} \vdash C\{?/x\}}{\Gamma, x : \mu a.B \vdash C}$$

But there are many ways of doing induction[7].

[7] Tarmo Uustalu and Varmo Vene. "Least and greatest fixed points in intuitionistic natural deduction". In: *Theor. Comput. Sci.* 272.1-2 (2002), pp. 315–339.

Conclusion

- We have set-up a program for discussing the connection between Agda and focused sequent calculus.
- We provides a sequent calculus with dependent types, although the design space is big.
- We leave induction and the identity type for future work.

Thanks! Questions?

Identity type – my intution

$$\frac{\Psi \vdash \sigma\,\text{unifier}(M, N) : \Gamma \longrightarrow \Delta \quad \Psi \mid \Delta \vdash t : C\{\sigma\}}{\Psi \mid \Gamma, x : Id_A(M, N) \vdash J(x; t) : C}$$

- Should be the rule that performs unification, which have non-local effects on the context.
- Do we need inaccessible patterns in Γ?
- Understanding the unification judgement could give us an calculus for talking about the without K rule[8].

```
hmm? : (x : A). P x → (y : A). Id A x y → P y
hmm? ⌊y⌋ p y refl = p
```

[8] Jesper Cockx et al. "Pattern Matching Without K". . In: *ICFP'14*. 2014, pp. 257–268.