From the simply-typed lambda-calculus to cartesian closed categories

Andreas Abel

December 2017

Outline of the transformation.

1. Simply-typed lambda-calculus (named variables).
2. Explicit substitutions and judgemental equality.
3. Nameless (de Bruijn) presentation.
4. Removing the judgement for well-typed variables: 0 is the only variable, the others are represented using the weakening substitution.
5. Conflate contexts and types, substitutions and terms: we arrive at an internal language for Cartesian closed categories.

1 From STLC to CCCs

1.1 STLC with named variables

Syntax.

Var	$\ni$	x			variable, e.g. string
Tm	$\ni$	t, u	$::=$	$x \mid \lambda x.\, t \mid t\, u$	term
Ty	$\ni$	A, B, C	$::=$	$o \mid A \Rightarrow B$	simple type
Cxt	$\ni$	Γ, Δ	$::=$	$\varepsilon \mid \Gamma.x{:}A$	typing context

Judgements.

$\Gamma \vdash t : A$	in context Γ, term t has type A
$\Gamma \vdash t = t' : A$	in context Γ, term t is equal to t' of type A

Equality rules.

$$\beta\ \frac{\Gamma.x{:}A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x.\, t)\, u = t[x{\leftarrow}u] : B} \qquad \eta\ \frac{\Gamma \vdash t : A \Rightarrow B}{\Gamma \vdash t = \lambda x.\, t\, x}\ x\#t$$

Substitution $t[x{\leftarrow}u]$ is defined by recursion on t. We write $z\#M$ when z does not clash with the free variables of terms contained in M.

$$\begin{array}{rcll} x[x{\leftarrow}u] & = & u & \\ y[x{\leftarrow}u] & = & y & \text{if } x \neq y \\ (t_1\,t_2)[x{\leftarrow}u] & = & (t_1[x{\leftarrow}u])\,(t_2[x{\leftarrow}u]) & \\ (\lambda x.\,t)[x{\leftarrow}u] & = & \lambda x.\,t & \\ (\lambda y.\,t)[x{\leftarrow}u] & = & \lambda y.\,t[x{\leftarrow}u] & \text{if } y\#u \\ (\lambda y.\,t)[x{\leftarrow}u] & = & \lambda z.\,t[y{\leftarrow}z][x{\leftarrow}u] & \text{for some } z\#(u,y) \end{array}$$

The complication in the last case is necessary to avoid capture of free variables of u, e.g., consider $(\lambda y.\,x)[x{\leftarrow}y]$. The result should be $\lambda z.\,y$ for some $z \neq y$, not $\lambda y.\,y$. Note that because of the last case, substitution is not formally structurally recursive. However, it could be defined by recursion on the height or size of term t, since this measure does not change by a renaming $t[y{\leftarrow}z]$.

1.2 Parallel substitution

Structural recursiveness of substitution can be restored if we substitute several variables at once, or even *all* free variables.
Syntax.

$$\mathsf{Subst} \quad \ni \quad \sigma \quad ::= \quad \varepsilon \mid \sigma, x{\mapsto}u \qquad \text{substitution}$$

Application of a substitution $t\,\sigma$ is defined by recursion on t.

$$\begin{array}{rcll} x\,\sigma & = & \sigma(x) & \\ (t_1\,t_2)\,\sigma & = & (t_1\,\sigma)\,(t_2\,\sigma) & \\ (\lambda y.\,t)\,\sigma & = & \lambda z.\,t\,(\sigma, y{\mapsto}z) & \text{for some } z\#\sigma \end{array}$$

Substitution typing $\Gamma \vdash \sigma : \Delta$.

$$\frac{}{\Gamma \vdash \varepsilon : \varepsilon} \qquad \frac{\Gamma \vdash \sigma : \Delta \qquad \Gamma \vdash u : A}{\Gamma \vdash (\sigma, x{\mapsto}u) : (\Delta.x{:}A)}$$

Exercise: Prove the substitution lemma: If $\Delta \vdash t : A$ and $\Gamma \vdash \sigma : \Delta$ then $\Gamma \vdash t\,\sigma : A$.

1.3 The category of substitutions

Identity substitution id_Γ is defined by recursion on Γ and substitution composition $\sigma \circ \sigma'$ by recursion on σ, applying σ' to every term in σ.

$$\frac{}{\Gamma \vdash \mathsf{id}_\Gamma : \Gamma} \qquad \frac{\Gamma_2 \vdash \sigma : \Gamma_3 \qquad \Gamma_1 \vdash \sigma' : \Gamma_2}{\Gamma_1 \vdash \sigma \circ \sigma' : \Gamma_3}$$

Exercise: Prove the category laws (identity and associativity)!

The β-law is now formulated as:

$$\beta\ \frac{\Gamma.x{:}A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x.\,t)\,u = t\,(\mathsf{id}_\Gamma, x{\mapsto}u) : B}$$

1.4 Explicit substitutions

We can "freeze" substitution by making it part of the term grammar, and animate it via equality rules.

$$\mathsf{Tm} \quad \ni \quad t, u \quad ::= \quad \cdots \mid t \circ \sigma \qquad \text{explicit substitution}$$

All the computation rules for substitutions become equality rules.

$$\frac{\Delta.x{:}A \vdash t : B \qquad \Gamma \vdash \sigma : \Delta}{\Gamma \vdash (\lambda x.\, t) \circ \sigma = \lambda y.\, t \circ (\sigma, x{\mapsto}y) : B}\ y\#\Gamma$$

Exercise: Write down the complete set of equality rules!

1.5 Nameless terms

Handling of names is a constant hassle. Semantically, a variable is not more than a pointer to a binding site, so let us implement this, following de Bruijn (1972)!

We drop names from the syntax everywhere, and replace variables by natural numbers denoting how many binders to skip when traversing towards the root to reach the binding site. E.g. the named term $\lambda x.\, x\,(\lambda y.\, x\,(\lambda x.\, x\, y))$ becomes $\lambda.\, 0\,(\lambda.\, 1\,(\lambda.\, 0\, 1))$.

Syntax.

Var	$\ni$	x	$::=$	$0 \mid 1 + x$	de Bruijn index
Tm	$\ni$	t, u	$::=$	$x \mid \lambda t \mid t\, u \mid t \circ \sigma$	term
Cxt	$\ni$	Γ, Δ	$::=$	$\varepsilon \mid \Gamma.A$	typing context
Subst	$\ni$	σ	$::=$	$\varepsilon \mid \sigma, u$	substitution

The new judgement $\Gamma \ni x : A$ asserts that index x points to an occurrence of A in the list Γ:

$$\frac{}{\Gamma.A \ni 0 : A} \qquad \frac{\Gamma \ni x : A}{\Gamma.B \ni 1 + x : A}$$

Typing $\Gamma \vdash t : A$ of terms becomes:

$$\frac{\Gamma \ni x : A}{\Gamma \vdash x : A} \qquad \frac{\Gamma.A \vdash t : B}{\Gamma \vdash \lambda t : A \Rightarrow B}$$

Exercise: What becomes of the rules for application and explicit substitution?

Substitution typing $\Gamma \vdash \sigma : \Delta$ becomes:

$$\frac{}{\Gamma \vdash \varepsilon : \varepsilon} \qquad \frac{\Gamma \vdash \sigma : \Delta \qquad \Gamma \vdash u : A}{\Gamma \vdash (\sigma, u) : (\Delta.A)}$$

Exercise: Write out the equality rules!

1.6 Explicit weakening

Typings stay valid under context extensions: In the named case, $\Gamma \vdash t : A$ implies $\Gamma.x{:}B \vdash t : A$ in case $x\#\Gamma$. In the nameless case, this can be realized by a substitution (exercise!), but may also become a constructor for terms. As a consequence, we only need variable 0, and can drop the syntactic class Var entirely!

Syntax.

$$\mathsf{Tm} \quad \ni \quad t, u \quad ::= \quad 0 \mid 1{+}\, t \mid \ldots \qquad \text{term}$$

Typing.

$$\frac{}{\Gamma.A \vdash 0 : A} \qquad \frac{\Gamma \vdash t : A}{\Gamma.B \vdash 1{+}\, t : A}$$

Exercise: Equality rules!

1.7 CCCs: Contexts as products, substitutions as tuples

If we view contexts $\varepsilon.A_1.\cdots.A_n$ simply as products $((1 \times A_1) \times \cdots) \times A_n$ with the unit type 1 replacing the empty context, substitutions σ become simply nested tuples! The typing judgements become morphism typing:

$$\begin{array}{lcl} \Gamma \vdash t : A & \rightsquigarrow & t : \Gamma \longrightarrow A \\ \Gamma \vdash \sigma : \Delta & \rightsquigarrow & \sigma : \Gamma \longrightarrow \Delta \end{array}$$

Substitution typing:

$$\frac{}{! : A \to 1} \qquad \frac{t : A \to B \qquad u : A \to C}{(t, u) : A \to B \times C}$$

$$\frac{}{\mathsf{id}_A : A \longrightarrow A} \qquad \frac{t : B \longrightarrow C \qquad u : A \longrightarrow B}{t \circ u : A \longrightarrow C}$$

Variable 0 becomes the second projection π_2 and weakening $1{+}\, t$ a composition $t \circ \pi_1$ with the first projection.

$$\frac{}{\pi_2 : A \times B \longrightarrow B} \qquad \frac{}{\pi_1 : A \times B \longrightarrow A}$$

Abstraction λt is called *currying* and application $t\, u$ is decomposed into $\mathsf{eval} \circ (t, u)$.

$$\frac{t : A \times B \longrightarrow C}{\lambda t : A \longrightarrow (B \Rightarrow C)} \qquad \frac{}{\mathsf{eval} : (A \Rightarrow B) \times A \longrightarrow B}$$

Exercise: Write out the equational theory of CCCs.

Raw syntax of the initial CCC:

$$\begin{array}{lcll} \mathsf{Ob} & \ni & A, B, C & ::= \quad 1 \mid A \times B \mid A \Rightarrow B \\ \mathsf{Tm} & \ni & t, u & ::= \quad \mathsf{id} \mid t \circ u \mid ! \mid (t, u) \mid \pi_1 \mid \pi_2 \mid \lambda t \mid \mathsf{eval} \end{array}$$

2 Exercises (Pen and Paper)

2.1 Simply-typed lambda-calculus with explicit substitutions

1. Write down the typing rules for simply-typed lambda-calculus.
2. Write down the typing rules for substitutions.
3. Write down the equality rules.
   a) β- and η-equality.
   b) Rules for the propagation of explicit substitutions into terms.
   c) Rules for the composition of substitutions.

2.2 Categories

1. Category of monoids.
   a) Define the category of (small) monoids (where the carrier of the monoid is a set).
   b) Show that the length function for lists is a monoid morphism.
2. Category of categories.
   a) Generalize the notion of monoid morphism to the concept of morphism between categories.
   b) Show that the (small) categories (where the objects form a set) form a category themselves. (This category is large in the sense that objects form a class.)

2.3 Products

1. Define the product in the category of monoids.
2. Let 1 denote the terminal object of some category $\mathcal{C}$. Show that the following span is a product of A and 1.

$$A \xleftarrow{\mathsf{id}} A \xrightarrow{!} 1$$

2.4 Cartesian closed categories

Recall $\mathsf{eval} : (A \Rightarrow B) \times A \longrightarrow B$ and $\mathsf{curry}\, f : C \longrightarrow (A \Rightarrow B)$ for $f : C \times A \longrightarrow B$.

1. Show $\mathsf{curry}\,\mathsf{eval} = \mathsf{id}$.
2. Show $A \cong (1 \Rightarrow A)$.

3 Exercises (Agda)

3.1 Simply-typed lambda-calculus in Agda

1. Represent simply-typed terms in Agda, using de Bruijn indices.
   a) Code simple types a and contexts Γ as data types.
   b) Define well-typed variables as indexed data type $\mathsf{Var}\,\Gamma\,a$.
   c) Define well-typed terms as indexed data type $\mathsf{Tm}\,\Gamma\,a$.
2. Add explicit substitutions via an indexed data type $\mathsf{Sub}\,\Gamma\,\Delta$.
3. Define an equality judgement as relation between well-typed terms: $\_\cong\_ : \mathsf{Tm}\,\Gamma\,a \to \mathsf{Tm}\,\Gamma\,a \to \mathsf{Set}$. Each rule is one constructor of this indexed data type.
4. Likewise, implement an equality judgement for well-typed substitutions.

3.2 CCCs in Agda

1. An E-category is a category with an equivalence relation on homsets. Define the notion of E-category in Agda.
   a) There is a Set of objects Ob.
   b) For each two objects $a, b : \mathsf{Ob}$ there is a Set of (homo)morphisms $\mathsf{Hom}\,a\,b$ from a to b.
   c) There is an equivalence relation on $\mathsf{Hom}\,a\,b$ (for each $a, b : \mathsf{Ob}$).
   d) There is an associative morphism composition $f \circ g : \mathsf{Hom}\,a\,c$ for each $f : \mathsf{Hom}\,b\,c$ and $g : \mathsf{Hom}\,a\,b$.
   e) Composition respects equality.
   f) There are morphisms $\mathsf{id}\,a$ for each $a : \mathsf{Ob}$ which are left and right units for composition.
2. Add products and terminal objects.
3. Add exponentials.

3.3 Interpretation of STLC in CCCs

1. Fix an arbitrary CCC.
2. Write an interpretation of types and contexts as objects in the CCC.
3. Write an interpretation of well-typed terms and substitutions as morphisms in the CCC.
4. Write an interpretation of judgemental equality as equality of morphisms in the CCC.

- First, prove each rule of judgemental equality as a theorem about morphisms in a CCC.
- Then, map the rules to these theorems.

References

de Bruijn, N.G. (1972). "Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem". In: *Indagationes Mathematicae (Proceedings)* 75.5, pp. 381–392. ISSN: 1385-7258. DOI: https://doi.org/10.1016/1385-7258(72)90034-0.