Propositions as Types

Philip Wadler
University of Edinburgh

ACCL (POPL 1990)




Explicit Substitutions

M. Abadi* L. Cardelli* P.-L. Curien† J.-J. Lévy‡

Abstract

The $\lambda\sigma$-calculus is a refinement of the λ-calculus where substitutions are manipulated explicitly. The $\lambda\sigma$-calculus provides a setting for studying the theory of substitutions, with pleasant mathematical properties. It is also a useful bridge between the classical λ-calculus and concrete implementations.

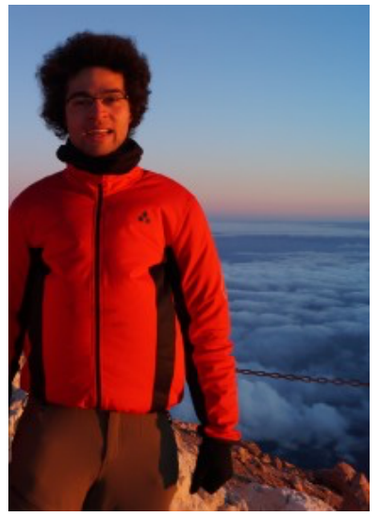
The correspondence between the theory and its implementations becomes highly nontrivial, and the correctness of the implementations can be compromised.

In this paper we study the $\lambda\sigma$-calculus, a refinement of the λ-calculus [1] where substitutions are manipulated explicitly. Substitutions have syntactic representations, and if a is a term and s is a substitution then the term $a[s]$ represents a with the substitution

ACCL (POPL 1990)

Beta	$(\lambda a)\, b = a[b \cdot id]$
VarId	$\mathbf{1}[id] = \mathbf{1}$
VarCons	$\mathbf{1}[a \cdot s] = a$
App	$(ba)[s] = (b[s])\,(a[s])$
Abs	$(\lambda a)[s] = \lambda(a[\mathbf{1} \cdot (s \circ \uparrow)])$
Clos	$a[s][t] = a[s \circ t]$
IdL	$id \circ s = s$
ShiftId	$\uparrow \circ id = \uparrow$
ShiftCons	$\uparrow \circ (a \cdot s) = s$
Map	$(a \cdot s) \circ t = a[t] \cdot (s \circ t)$
Ass	$(s_1 \circ s_2) \circ s_3 = s_1 \circ (s_2 \circ s_3)$.

Autosubst (ITP 2015)


Autosubst: Reasoning with de Bruijn Terms and Parallel Substitution

Steven Schäfer Tobias Tebbi Gert Smolka

Saarland University
June 10, 2015

To appear in Proc. of ITP 2015, Nanjing, China, Springer LNAI

Reasoning about syntax with binders plays an essential role in the formalization of the metatheory of programming languages. While the intricacies of binders can be ignored in paper proofs, formalizations involving binders tend to be heavyweight. We present a discipline for syntax with binders based on de Bruijn terms and parallel substitutions, with a decision procedure covering all assumption-free equational substitution lemmas. The approach is implemented in the Coq library AUTOSUBST, which additionally derives substitution operations and proofs of substitution lemmas for custom term types. We demonstrate the effectiveness of the approach with several case studies, including part A of the POPLmark challenge.

Autosubst (ITP 2015)

$$(st)[\sigma] \equiv (s[\sigma])(t[\sigma]) \qquad \mathsf{id} \circ \sigma \equiv \sigma$$

$$(\lambda.\, s)[\sigma] \equiv \lambda.\, (s[0 \cdot \sigma \circ \uparrow]) \qquad \sigma \circ \mathsf{id} \equiv \sigma$$

$$0[s \cdot \sigma] \equiv s \qquad (\sigma \circ \tau) \circ \theta \equiv \sigma \circ (\tau \circ \theta)$$

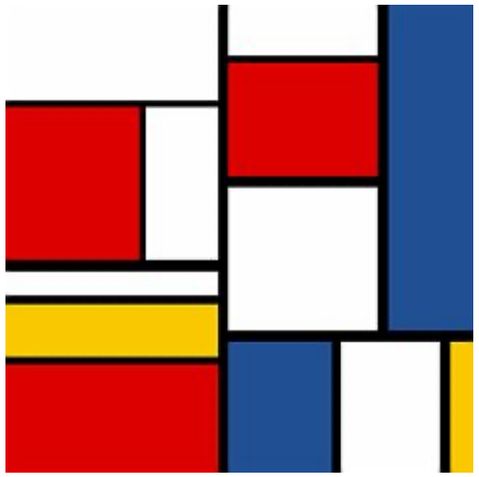
$$\uparrow \circ (s \cdot \sigma) \equiv \sigma \qquad (s \cdot \sigma) \circ \tau \equiv s[\tau] \cdot \sigma \circ \tau$$

$$s[\mathsf{id}] \equiv s \qquad s[\sigma][\tau] \equiv s[\sigma \circ \tau]$$

$$0[\sigma] \cdot (\uparrow \circ \sigma) \equiv \sigma \qquad 0 \cdot \uparrow \equiv \mathsf{id}$$

Figure 1: The convergent rewriting system of the σ-calculus

ACMM (CPP 2017)



Type-and-Scope Safe Programs and Their Proofs

Guillaume Allais
gallais@cs.ru.nl
Radboud University,
The Netherlands

James Chapman Conor McBride
{james.chapman,conor.mcbride}@strath.ac.uk
University of Strathclyde, UK

James McKinna
james.mckinna@ed.ac.uk
University of Edinburgh, UK

Abstract

We abstract the common type-and-scope safe structure from computations on λ-terms that deliver, e.g., renaming, substitution, evaluation, CPS-transformation, and printing with a name supply. By exposing this structure, we can prove generic simulation and fusion lemmas relating operations built this way. This work has been fully formalised in Agda.


```
ren : (∀σ. Var σ Γ → Var σ Δ) → (∀σ. Tm σ Γ → Tm σ Δ)
ren ρ (`var v)  = `var (ρ v)
ren ρ (f `$ t)  = ren ρ f `$ ren ρ t
ren ρ (`λ b)    = `λ (ren ((su ∘ ρ) –, ze) b)

sub : (∀σ. Var σ Γ → Tm σ Δ) → (∀σ. Tm σ Γ → Tm σ Δ)
sub ρ (`var v) = ρ v
```

ACMM (CPP 2017)

```
ren : (∀σ. Var σ Γ → Var σ Δ) → (∀σ. Tm σ Γ → Tm σ Δ)
ren ρ ('var υ)  = 'var (ρ υ)
ren ρ (f '$ t)  = ren ρ f '$ ren ρ t
ren ρ ('λ b)    = 'λ (ren ((su ∘ ρ) –, ze) b)

sub : (∀σ. Var σ Γ → Tm σ Δ) → (∀σ. Tm σ Γ → Tm σ Δ)
sub ρ ('var υ)  = ρ υ
sub ρ (f '$ t)  = sub ρ f '$ sub ρ t
sub ρ ('λ b)    = 'λ (sub ((ren su ∘ ρ) –, 'var ze) b)
```

Figure 1. Renaming and Substitution for the STλC

```
kit : (∀σ. Var σ Γ → ♦ σ Δ) → (∀σ. Tm σ Γ → Tm σ Δ)
kit ρ ('var υ)  = Kit.var κ (ρ υ)
kit ρ (f '$ t)  = kit ρ f '$ kit ρ t
kit ρ ('λ b)    = 'λ (kit ((Kit.wkn κ ∘ ρ) –, Kit.zro κ) b)
```

Figure 2. Kit traversal for the STλC, for κ of type Kit ♦

ABW (CPP 2025?)

Substitution without copy and paste

Thorsten Altenkirch
University of Nottingham
Nottingham, United Kingdom
thorsten.altenkirch@nottingham.ac.uk

Nathaniel Burke
Imperial College London
London, United Kingdom
nathaniel.burke21@imperial.ac.uk

Philip Wadler
University of Edinburgh
Edinburgh, United Kingdom
wadler@inf.ed.ac.uk

Abstract

When defining substitution recursively for a language with binders like the simply typed λ-calculus, we need to define substitution and renaming separately. When we want to verify the categorical properties of this calculus, we end up repeating the same argument many times. In this paper we present a lightweight method that avoids this repetition and is implemented in Agda.

dependent type theory which may have interesting applications for the coherence problem, i.e. interpreting dependent types in higher categories.

1.1 In a nutshell

When working with substitution for a calculus with binders, we find that you have to differentiate between renamings ($\Delta \models \mathrm{v}\ \Gamma$) where variables are substituted only for variables

ABW (CPP 2025?)

$$\_\mathsf{v}[\_]\mathsf{v} : \Gamma \ni A \to \Delta \models\mathsf{v}\, \Gamma \to \Delta \ni A$$
$$\_\mathsf{v}[\_] : \Gamma \ni A \to \Delta \models \Gamma \to \Delta \vdash A$$
$$\_[\_]\mathsf{v} : \Gamma \vdash A \to \Delta \models\mathsf{v}\, \Gamma \to \Delta \vdash A$$
$$\_[\_] : \Gamma \vdash A \to \Delta \models \Gamma \to \Delta \vdash A$$

$$\_[\_] : \Gamma \vdash [\ \mathsf{q}\ ]\, A \to \Delta \models [\ \mathsf{r}\ ]\, \Gamma \to \Delta \vdash [\ \mathsf{q} \sqcup \mathsf{r}\ ]\, A$$