Mechanising Syntax with Binders in Coq

Kathrin Stark

Saarbrücken, February 14, 2020

Syntax with Binders

Church's Lambda Calculus [Church '32]

- **Binders** are a key ingredient of Church's λ-calculus:

$$s := x \mid s\,t \mid \lambda x.s$$

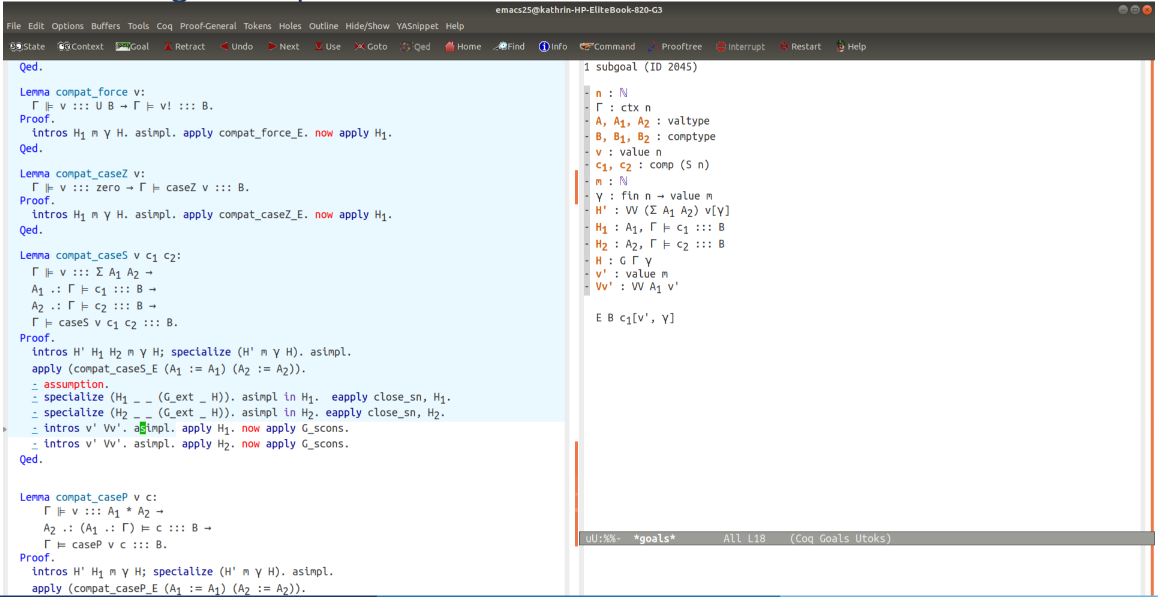
  - A function $\begin{tikzcd}[column sep=tiny] \lambda f . & f \arrow[l, bend left] \end{tikzcd}, x$ **binds** a **variable** f
  - $(\lambda f.f\ x)\ g$ **reduces** to $(f\ x)[f/g]$ where each occurrence of f is **substituted** by g
- Proofs such as
  - type safety
  - weak/strong normalisation

$\Rightarrow$ Binders are inevitable when talking about formal systems

Mechanising in Coq

- **Interactive proof assistants** allow to develop proofs restricted to a small set of reasoning principles in interplay with a computer:
  - Verification that only the agreed-on rules are used
  - Automation of easy/repetitive cases
  - Adaption of changes
- **Here:** The Coq Proof Assistant
  - Based on the Calculus of Inductive Constructions [Coquand Huet '86, Coquand and Paulin '88]
  - Proof checking is reduced to type checking via the Curry-Howard Correspondence [Howard '80]
  - ⇒ Everything in this thesis is mechanised in Coq

Mechanising in Coq



```
  Qed.

  Lemma compat_force v:
    Γ ⊫ v ::: U B → Γ ⊨ v! ::: B.
  Proof.
    intros H₁ m γ H. asimpl. apply compat_force_E. now apply H₁.
  Qed.

  Lemma compat_caseZ v:
    Γ ⊫ v ::: zero → Γ ⊨ caseZ v ::: B.
  Proof.
    intros H₁ m γ H. asimpl. apply compat_caseZ_E. now apply H₁.
  Qed.

  Lemma compat_caseS v c₁ c₂:
    Γ ⊫ v ::: Σ A₁ A₂ →
    A₁ .: Γ ⊨ c₁ ::: B →
    A₂ .: Γ ⊨ c₂ ::: B →
    Γ ⊨ caseS v c₁ c₂ ::: B.
  Proof.
    intros H' H₁ H₂ m γ H; specialize (H' m γ H). asimpl.
    apply (compat_caseS_E (A₁ := A₁) (A₂ := A₂)).
    - assumption.
    - specialize (H₁ _ _ (G_ext _ H)). asimpl in H₁.  eapply close_sn, H₁.
    - specialize (H₂ _ _ (G_ext _ H)). asimpl in H₂. eapply close_sn, H₂.
    - intros v' Vv'. asimpl. apply H₁. now apply G_scons.
    - intros v' Vv'. asimpl. apply H₂. now apply G_scons.
  Qed.


  Lemma compat_caseP v c:
      Γ ⊫ v ::: A₁ * A₂ →
      A₂ .: (A₁ .: Γ) ⊨ c ::: B →
      Γ ⊨ caseP v c ::: B.
  Proof.
    intros H' H₁ m γ H; specialize (H' m γ H). asimpl.
    apply (compat_caseP_E (A₁ := A₁) (A₂ := A₂)).
```

```
1 subgoal (ID 2045)

- n : ℕ
- Γ : ctx n
- A, A₁, A₂ : valtype
- B, B₁, B₂ : comptype
- v : value n
- c₁, c₂ : comp (S n)
- m : ℕ
- γ : fin n → value m
- H' : VV (Σ A₁ A₂) v[γ]
- H₁ : A₁, Γ ⊨ c₁ ::: B
- H₂ : A₂, Γ ⊨ c₂ ::: B
- H : G Γ γ
- v' : value m
- Vv' : VV A₁ v'

  E B c₁[v', γ]
```

Mechanising in Coq

- **Interactive proof assistants** allow to develop proofs restricted to a small set of reasoning principles in interplay with a computer:
    - Verification that only the agreed-on rules are used
    - Automation of easy/repetitive cases
    - Adaption of changes
- **Here:** The Coq Proof Assistant
    - Based on the Calculus of Inductive Constructions [Coquand Huet '86, Coquand and Paulin '88]
    - Proof checking is reduced to type checking via the Curry-Howard Correspondence [Howard '80]

  ⇒ Everything in this thesis is mechanised in Coq

Why mechanising the meta-theory of formal systems?

Mechanising in Coq

- **Interactive proof assistants** allow to develop proofs restricted to a small set of reasoning principles in interplay with a computer:
  - Verification that only the agreed-on rules are used
  - Automation of easy/repetitive cases
  - Adaption of changes
- **Here:** The Coq Proof Assistant
  - Based on the Calculus of Inductive Constructions [Coquand Huet '86, Coquand and Paulin '88]
  - Proof checking is reduced to type checking via the Curry-Howard Correspondence [Howard '80]

  ⇒ Everything in this thesis is mechanised in Coq

Call-By-Push-Value in Coq [Forster, Schäfer, Spies, Stark '19]

Syntax

$$\begin{array}{llll}
\text{(value types)} & A, B & ::= & 1 \mid U\,C \mid A_1 \times A_2 \mid 0 \mid A_1 + A_2 \\
\text{(computation types)} & C, D & ::= & \top \mid F\,A \mid A \to C \mid C_1 \,\&\, C_2 \\
\text{(environments)} & \Gamma & ::= & x_1 : A_1, \ldots, x_n : A_n
\end{array}$$

Value typing $\boxed{\Gamma \vdash V : A}$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{}{\Gamma \vdash () : 1} \qquad \frac{\Gamma \vdash M : C}{\Gamma \vdash \{M\} : U\,C} \qquad \frac{\Gamma \vdash V_1 : A_1 \qquad \Gamma \vdash V_2 : A_2}{\Gamma \vdash (V_1, V_2) : A_1 \times A_2} \qquad \frac{\Gamma \vdash V : A_i}{\Gamma \vdash \mathbf{inj}_i\ V : A_1 + A_2}$$

Computation typing $\boxed{\Gamma \vdash M : C}$

$$\frac{}{\Gamma \vdash \langle\rangle : \top} \qquad \frac{\Gamma \vdash V : A}{\Gamma \vdash \mathbf{return}\ V : F\,A} \qquad \frac{\Gamma \vdash M : F\,A \qquad \Gamma, x : A \vdash N : C}{\Gamma \vdash \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N : C} \qquad \frac{\Gamma, x : A \vdash M : C}{\Gamma \vdash \lambda x.M : A \to C} \qquad \frac{\Gamma \vdash M : A \to C \qquad \Gamma \vdash V : A}{\Gamma \vdash M\ V : C}$$

$$\frac{\Gamma \vdash V : U\,C}{\Gamma \vdash V! : C} \qquad \frac{\Gamma \vdash V : A_1 \times A_2 \qquad \Gamma, x_1 : A_1, x_2 : A_2 \vdash M : C}{\Gamma \vdash \mathbf{split}(V, x_1.x_2.M) : C} \qquad \frac{\Gamma \vdash V : 0}{\Gamma \vdash \mathbf{case_0}(V) : C}$$

$$\frac{\Gamma \vdash V : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash M_1 : C \qquad \Gamma, x_2 : A_2 \vdash M_2 : C}{\Gamma \vdash \mathbf{case}(V, x_1.M_1, x_2.M_2) : C} \qquad \frac{\Gamma \vdash M_1 : C_1 \qquad \Gamma \vdash M_2 : C_2}{\Gamma \vdash \langle M_1, M_2 \rangle : C_1 \,\&\, C_2} \qquad \frac{\Gamma \vdash M : C_1 \,\&\, C_2}{\Gamma \vdash \mathbf{prj}_i\ M : C_i}$$

Call-By-Push-Value in Coq [Forster, Schäfer, Spies, S '19]

Mechanisation in 8000 lines of Coq code of

- standard operational semantics for CBPV
  - normalisation using logical relations
  - adequacy of set/algebra semantics
- unrestricted operational semantics for CBPV
  - confluence
  - strong normalisation using Kripke logical relations
  - soundness of equational theory
- translations of CBV/CBN into CBPV
  - preservation of operational semantics
  - confluence for full λ-calculus
  - strong normalisation for strong CBV/CBN
  - soundness of equational theories
  - adequate type-theoretic algebra semantics for CBV/CBN

How to represent binders and substitution?

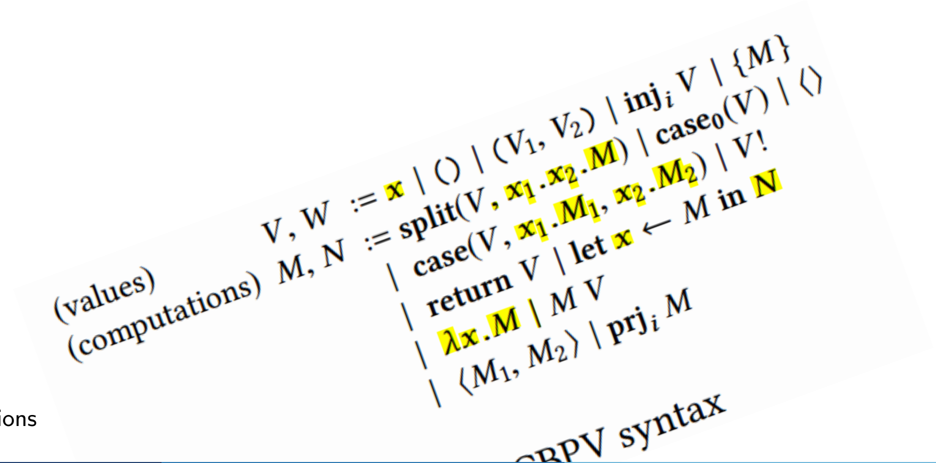
Contribution

A practical appraoch to mechanising syntax with binders in Coq for a wide range of syntactic systems.

“We assume an understanding
of the operation of substituting a
given symbol or formula for a particular
occurrence of a given symbol or formula”
Church, ’32

What Does an Understanding of Substitution Comprise in Coq?

Example: Strong Normalisation for Call-by-Push-Value[Forster et al., '19]



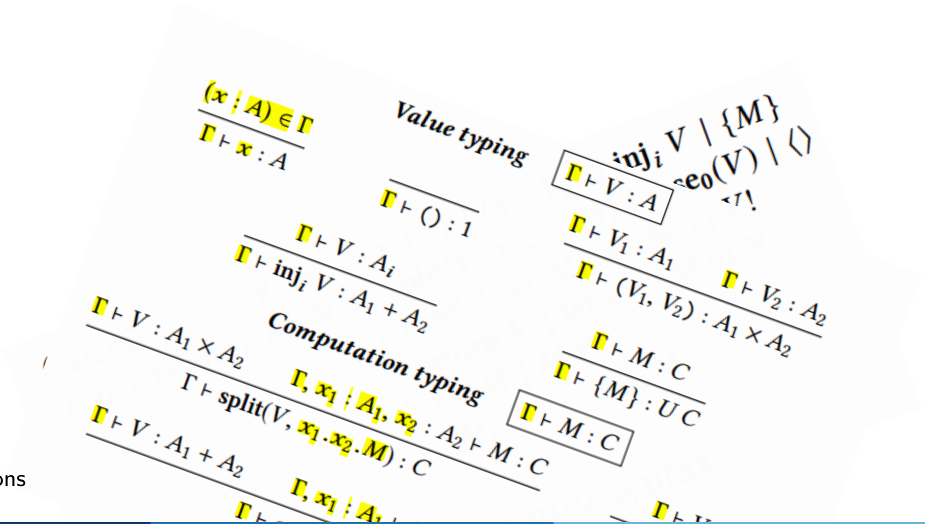
$$
\begin{array}{llll}
\text{(values)} & V, W & := & x \mid () \mid (V_1, V_2) \mid \mathbf{inj}_i\, V \mid \{M\} \\
\text{(computations)} & M, N & := & \mathbf{split}(V, x_1.x_2.M) \mid \mathbf{case}_0(V) \mid \langle\rangle \\
 & & \mid & \mathbf{case}(V, x_1.M_1, x_2.M_2) \mid V! \\
 & & \mid & \mathbf{return}\, V \mid \mathbf{let}\, x \leftarrow M\, \mathbf{in}\, N \\
 & & \mid & \lambda x.M \mid M\, V \\
 & & \mid & \langle M_1, M_2\rangle \mid \mathbf{prj}_i\, M
\end{array}
$$

CBPV syntax

▮ } Expressions

What Does an Understanding of Substitution Comprise in Coq?

Example: Strong Normalisation for Call-by-Push-Value[Forster et al., '19]



Value typing $\boxed{\Gamma \vdash V : A}$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{}{\Gamma \vdash () : 1} \qquad \frac{\Gamma \vdash V : A_i}{\Gamma \vdash \mathrm{inj}_i\ V : A_1 + A_2}$$

$$\frac{\Gamma \vdash V_1 : A_1 \qquad \Gamma \vdash V_2 : A_2}{\Gamma \vdash (V_1, V_2) : A_1 \times A_2} \qquad \frac{\Gamma \vdash M : C}{\Gamma \vdash \{M\} : U\,C}$$

Computation typing $\boxed{\Gamma \vdash M : C}$

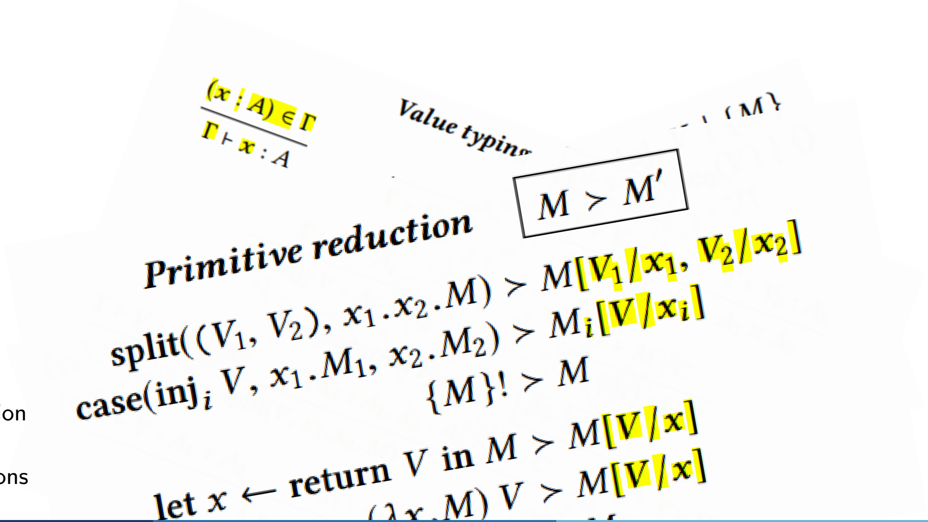
$$\frac{\Gamma \vdash V : A_1 \times A_2 \qquad \Gamma, x_1 : A_1, x_2 : A_2 \vdash M : C}{\Gamma \vdash \mathrm{split}(V, x_1.x_2.M) : C}$$

$$\Gamma \vdash V : A_1 + A_2 \qquad \Gamma, x_1 : A_1$$

▮ Expressions

What Does an Understanding of Substitution Comprise in Coq?

Example: Strong Normalisation for Call-by-Push-Value[Forster et al., '19]



$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}$$

Value typing

$$\boxed{M \succ M'}$$

Primitive reduction

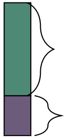
$$\mathrm{split}((V_1, V_2), x_1.x_2.M) \succ M[V_1/x_1, V_2/x_2]$$

$$\mathrm{case}(\mathrm{inj}_i\, V, x_1.M_1, x_2.M_2) \succ M_i[V/x_i]$$

$$\{M\}! \succ M$$

$$\mathrm{let}\ x \leftarrow \mathrm{return}\ V\ \mathrm{in}\ M \succ M[V/x]$$

$$(\lambda x.M)\, V \succ M[V/x]$$

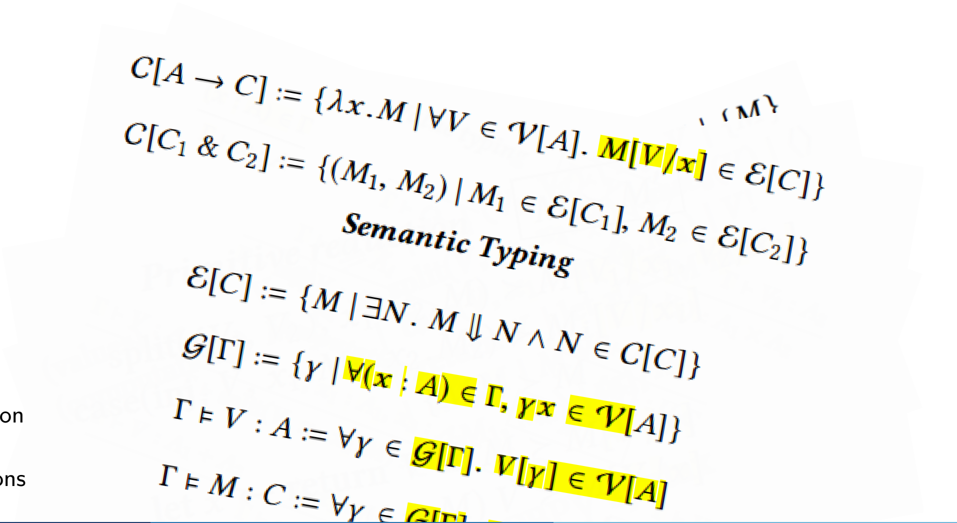


Instantiation

Expressions

What Does an Understanding of Substitution Comprise in Coq?

Example: Strong Normalisation for Call-by-Push-Value[Forster et al., '19]



$\mathcal{C}[A \to C] := \{\lambda x. M \mid \forall V \in \mathcal{V}[A].\ M[V/x] \in \mathcal{E}[C]\}$

$\mathcal{C}[C_1 \,\&\, C_2] := \{(M_1, M_2) \mid M_1 \in \mathcal{E}[C_1], M_2 \in \mathcal{E}[C_2]\}$

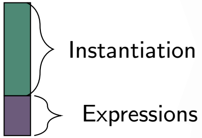
Semantic Typing

$\mathcal{E}[C] := \{M \mid \exists N.\ M \Downarrow N \wedge N \in \mathcal{C}[C]\}$

$\mathcal{G}[\Gamma] := \{\gamma \mid \forall (x : A) \in \Gamma,\ \gamma x \in \mathcal{V}[A]\}$

$\Gamma \vDash V : A := \forall \gamma \in \mathcal{G}[\Gamma].\ V[\gamma] \in \mathcal{V}[A]$

$\Gamma \vDash M : C := \forall \gamma \in \mathcal{G}[\Gamma]$

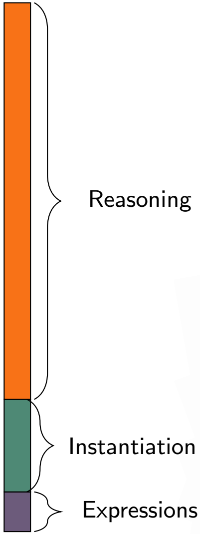


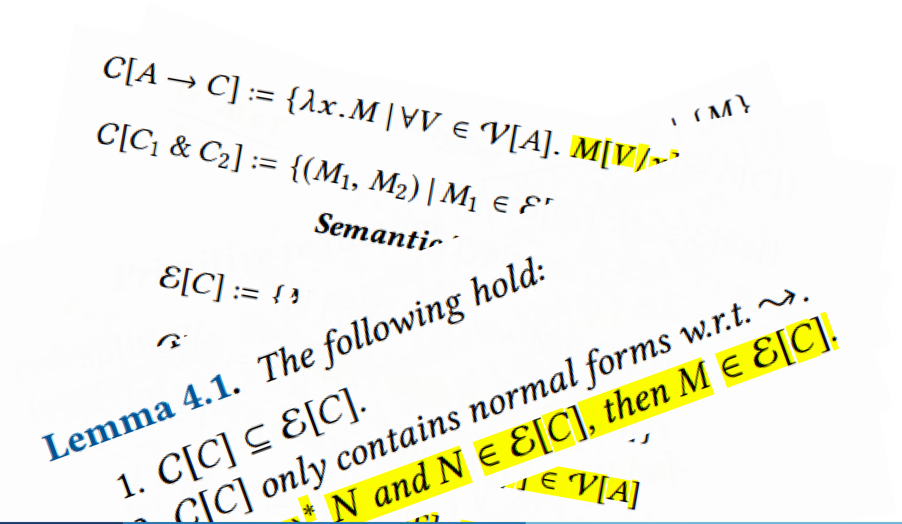
Instantiation

Expressions

What Does an Understanding of Substitution Comprise in Coq?

Example: Strong Normalisation for Call-by-Push-Value[Forster et al., '19]





$\mathcal{C}[A \to C] := \{\lambda x.M \mid \forall V \in \mathcal{V}[A].\ M[V/$ … $\{M\}$

$\mathcal{C}[C_1 \,\&\, C_2] := \{(M_1, M_2) \mid M_1 \in \mathcal{E}$

Semanti

$\mathcal{E}[C] := \{$

Lemma 4.1. The following hold:

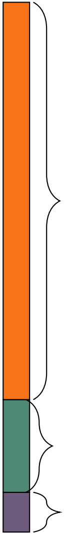
1. $\mathcal{C}[C] \subseteq \mathcal{E}[C]$.
2. $\mathcal{C}[C]$ only contains normal forms w.r.t. $\rightsquigarrow$.

* N and N $\in \mathcal{E}[C]$, then M $\in \mathcal{E}[C]$.

$\in \mathcal{V}[A]$

What Does an Understanding of Substitution Comprise in Coq?

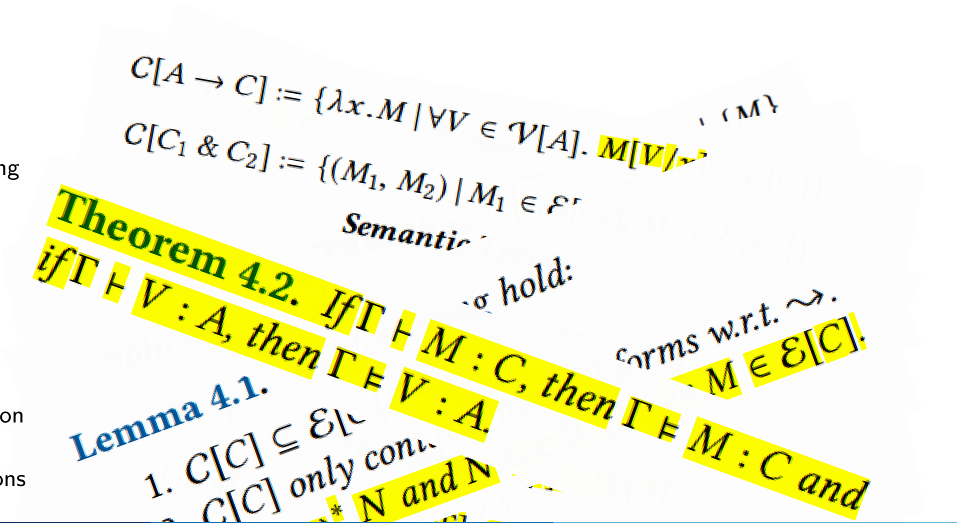
Example: Strong Normalisation for Call-by-Push-Value[Forster et al., '19]



Reasoning

Instantiation

Expressions



$\mathcal{C}[A \to C] := \{\lambda x.M \mid \forall V \in \mathcal{V}[A].\ M[V/$

$\mathcal{C}[C_1 \,\&\, C_2] := \{(M_1, M_2) \mid M_1 \in \mathcal{E}$

Semanti

hold:

forms w.r.t. $\rightsquigarrow$.

$M \in \mathcal{E}[C]$.

Theorem 4.2. *If* $\Gamma \vdash M : C$, *then* $\Gamma \vDash M : C$ *and if* $\Gamma \vdash V : A$, *then* $\Gamma \vDash V : A$.

Lemma 4.1.

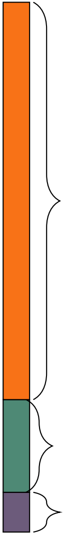
1. $\mathcal{C}[C] \subseteq \mathcal{E}[$

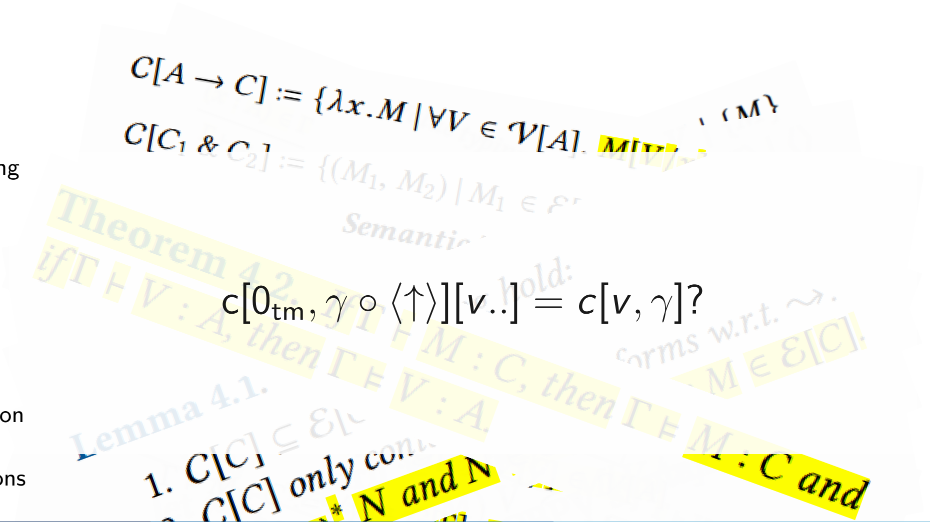
$\mathcal{C}[C]$ only con

N and N

What Does an Understanding of Substitution Comprise in Coq?

Example: Strong Normalisation for Call-by-Push-Value[Forster et al., '19]





$\mathcal{C}[A \to C] := \{\lambda x.M \mid \forall V \in \mathcal{V}[A].\ M[V/$ … $\{M\}$

$\mathcal{C}[C_1 \,\&\, C_2] := \{(M_1, M_2) \mid M_1 \in \mathcal{E}$

Semantic

Theorem 4.2. … hold:

if $\Gamma \vdash V : A$, then $\Gamma \vDash V : A$.

$\Gamma \vdash M : C$, then $\Gamma \vDash M : C$ and

norms w.r.t. $\leadsto$.

$M \in \mathcal{E}[C]$.

Lemma 4.1.

1. $\mathcal{C}[C] \subseteq \mathcal{E}[$

$\mathcal{C}[C]$ only con

N and N

$$c[0_{\mathrm{tm}}, \gamma \circ \langle\uparrow\rangle][v..] = c[v, \gamma]?$$

Related Work

Various ways to represent binders . . .

- named syntax
- unnamed syntax [de Bruijn '72]
- locally nameless [Aydemir et al. '08]
- parametric HOAS [Chlipala '08]
- nominal logic [Pitts '01]
- HOAS [Pfenning et al. '88]
- contextual modal TT [Nanevski et al. '08]
- . . .

Parts of a practical solution:

- Works in a general-purpose proof assistant, i.e., Coq
- As little overhead from the user's side as possible

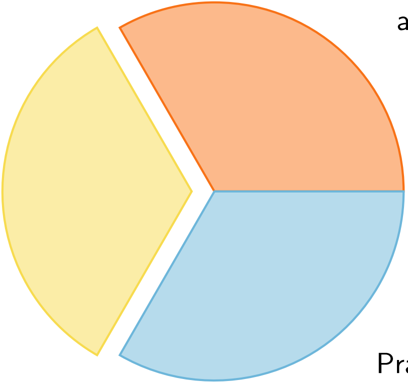
. . . and **no consensus**, see e.g. solutions to [Aydemir et al. '05].

Binders Come With Boilerplate [Rossberg et al.]

“Our experience [...] was more painful than we had anticipated. [...] **Out of a total of around 550 lemmas, approximately 400 were tedious ”infrastructure” lemmas**; only the remainder had direct relevance to the metatheory of Fω or elaboration.

Problem: We need a large number of technical lemmas to reason about substitutions.

Three Aspects of a Practical Presentation of Binders



Automatic Realisation
and Best Practices

Theoretical
Justification

Practical Evaluation

Representation of Binders in the Lambda Calculus

1. **Expressions:**
   - De Bruijn indices [de Bruijn '72]
   - Binders are presented by **references**, i.e. represented by natural numbers or a finite type:

   $$\lambda x.x(\lambda y.y\,x) \mapsto \lambda.0(\lambda.0\,1)$$

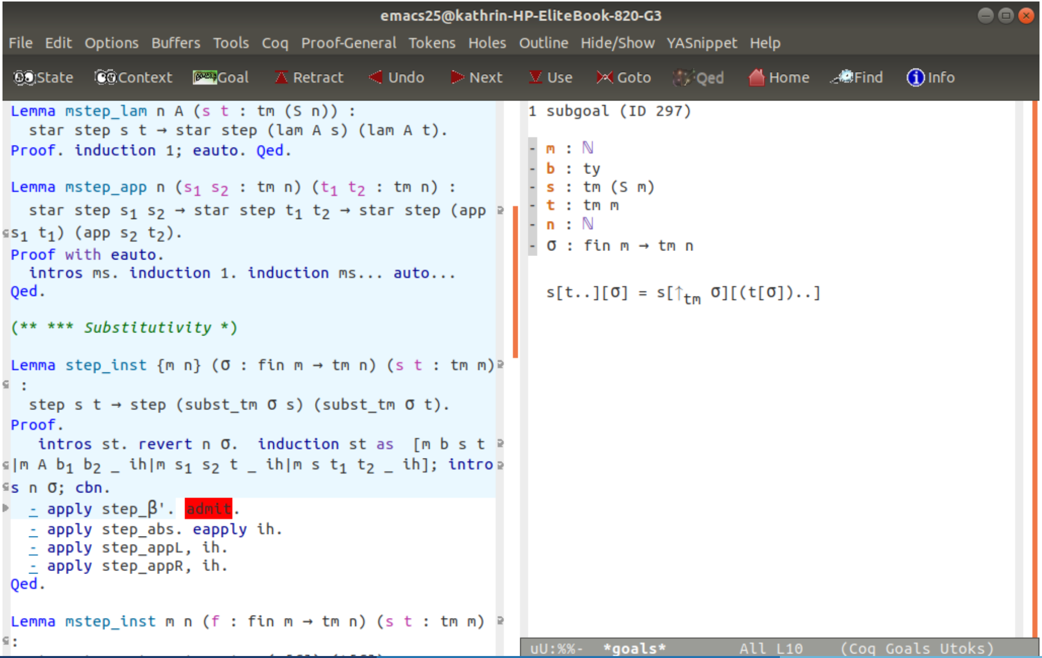
   $\Rightarrow$ α-equivalence is built-in
2. **Substitution:**
   - Parallel substitutions, first instantiation with renamings [Adams '04]
   - Primitives of the σ-calculus [Abadi et al. '91]
3. **Reasoning:**
   - By reducing to a normal form w.r.t. the reduction rules of the σ_{SP}-calculus

A Representation of Binders in the Lambda Calculus

Reasoning via Reduction to Normal Forms

Goal: Prove substitutivity of reduction, i.e. that $s \succ t$ implies $s[\sigma] \succ t[\sigma]$.

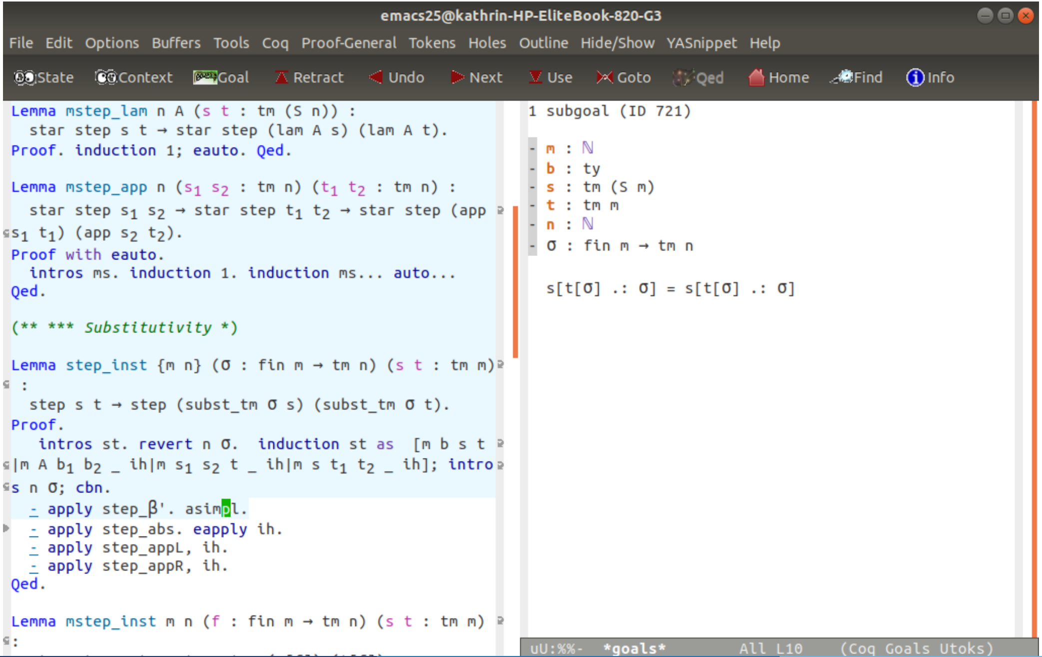
$$
\begin{aligned}
s[\sigma][t[\sigma]..] &= s[\mathsf{var}\,0 \cdot \sigma \circ \langle\uparrow\rangle][t[\sigma] \cdot \mathsf{var}] && \\
&= s[(\mathsf{var}\,0 \cdot \sigma \circ \langle\uparrow\rangle) \circ [(t[\sigma] \cdot \mathsf{var})]] && \text{compositionality} \\
&= s[(\mathsf{var}\,0)[t[\sigma] \cdot \mathsf{var}] \cdot (\sigma \circ \langle\uparrow\rangle) \circ [(t[\sigma] \cdot \mathsf{var})]] && \text{distributivity} \\
&= s[(\mathsf{var}\,0)[t[\sigma] \cdot \mathsf{var}] \cdot \sigma(\langle\uparrow\rangle \circ [t[\sigma] \cdot \mathsf{var}])] && \text{associativity} \\
&= s[(t[\sigma] \cdot \mathsf{var})\,0 \cdot (\sigma \circ [\uparrow (t[\sigma] \cdot \mathsf{var})])] && \text{compositionality} \\
&= s[t[\sigma] \cdot (\sigma \circ [\mathsf{var}])] && \cdot\text{, interaction} \\
&= s[t[\sigma] \cdot \sigma] && \text{right identity} \\
&= s[t[\sigma] \cdot (\mathsf{var} \circ [\sigma])] && \text{left identity} \\
&= s[(t \cdot \mathsf{var}) \circ [\sigma]] && \text{distributivity} \\
&= s[t \cdot \mathsf{var}][\sigma]. && \text{compositionality}
\end{aligned}
$$



```
Lemma mstep_lam n A (s t : tm (S n)) :
  star step s t → star step (lam A s) (lam A t).
Proof. induction 1; eauto. Qed.

Lemma mstep_app n (s₁ s₂ : tm n) (t₁ t₂ : tm n) :
  star step s₁ s₂ → star step t₁ t₂ → star step (app s₁ t₁) (app s₂ t₂).
Proof with eauto.
  intros ms. induction 1. induction ms... auto...
Qed.

(** *** Substitutivity *)

Lemma step_inst {m n} (σ : fin m → tm n) (s t : tm m) :
  step s t → step (subst_tm σ s) (subst_tm σ t).
Proof.
   intros st. revert n σ.  induction st as  [m b s t |m A b₁ b₂ _ ih|m s₁ s₂ t _ ih|m s t₁ t₂ _ ih]; intros n σ; cbn.
  - apply step_β'. admit.
  - apply step_abs. eapply ih.
  - apply step_appL, ih.
  - apply step_appR, ih.
Qed.

Lemma mstep_inst m n (f : fin m → tm n) (s t : tm m) :
```

```
1 subgoal (ID 297)

- m : ℕ
- b : ty
- s : tm (S m)
- t : tm m
- n : ℕ
- σ : fin m → tm n

  s[t..][σ] = s[↑tm σ][(t[σ])..]
```

```
uU:%%-  *goals*     All L10     (Coq Goals Utoks)
```



```
Lemma mstep_lam n A (s t : tm (S n)) :
  star step s t → star step (lam A s) (lam A t).
Proof. induction 1; eauto. Qed.

Lemma mstep_app n (s₁ s₂ : tm n) (t₁ t₂ : tm n) :
  star step s₁ s₂ → star step t₁ t₂ → star step (app s₁ t₁) (app s₂ t₂).
Proof with eauto.
  intros ms. induction 1. induction ms... auto...
Qed.

(** *** Substitutivity *)

Lemma step_inst {m n} (σ : fin m → tm n) (s t : tm m) :
  step s t → step (subst_tm σ s) (subst_tm σ t).
Proof.
   intros st. revert n σ.  induction st as  [m b s t |m A b₁ b₂ _ ih|m s₁ s₂ t _ ih|m s t₁ t₂ _ ih]; intros n σ; cbn.
  - apply step_β'. asimpl.
  - apply step_abs. eapply ih.
  - apply step_appL, ih.
  - apply step_appR, ih.
Qed.

Lemma mstep_inst m n (f : fin m → tm n) (s t : tm m) :
```

```
1 subgoal (ID 721)

- m : ℕ
- b : ty
- s : tm (S m)
- t : tm m
- n : ℕ
- σ : fin m → tm n

  s[t[σ] .: σ] = s[t[σ] .: σ]
```

```
uU:%%-  *goals*       All L10    (Coq Goals Utoks)
```

Representation of Binders in the Lambda Calculus

1. **Expressions:**
   - De Bruijn indices [de Bruijn '72]
   - Binders are presented by **references**, i.e. represented by natural numbers or a finite type:

   $$\lambda x.x(\lambda y.y\,x) \mapsto \lambda.0(\lambda.0\,1)$$

   $\Rightarrow$ α-equivalence is built-in
2. **Substitution:**
   - Parallel substitutions, first instantiation with renamings [Adams '04]
   - Primitives of the σ-calculus [Abadi et al. '91]
3. **Reasoning:**
   - By reducing to a normal form w.r.t. the reduction rules of the σ_{SP}-calculus
     - ★ Reduction in the σ_{SP}-calculus is sound and complete w.r.t. equality on the de Bruijn algebra [Schäfer, Smolka, Tebbi '15]
     - ★ Reduction in the σ_{SP}-calculus is convergent [Curien et al. '92]

Mechansing Convergence of the Sigma SP Calculus

Goal: To find a **normal form**, we require **confluence** and **termination** of the σ_{SP}-calculus.

Deferred by [Schäfer et al. '15]:
While the verification of our decision method is not difficult (even in Coq), a verification of the rewriting method is surprisingly complex since the existing termination proof [...] is far from straightforward. We did not succeed in simplifying this proof and think that a formalization with a proof assistant is a substantial enterprise.

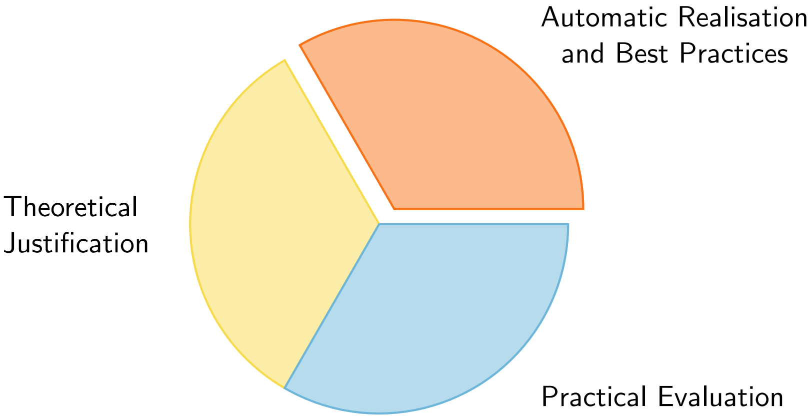
- Difficulty: Not trivially terminating
- Proved for related calculi has been proven in different manners [Hardin and Laville '86, Curien et al. '92, Zantema '92]
- Proof for σ-calculus mechanised in ALF [Kamareddine, Qiao '03]

Mechanising Convergence of the Sigma SP Calculus

Here: Mechanised for the exact system

- Simplified the proof significantly by building in intermediate reduction systems
- Merely 700 lines of code for convergence (Comparison: 1000 lines of code for verification of the rewriting method)
- (Local) confluence very easy due to automation of Coq, no critical pair analysis [Huet '77, Baader and Nipkow '99] required

Three Aspects of a Practical Presentation of Binders

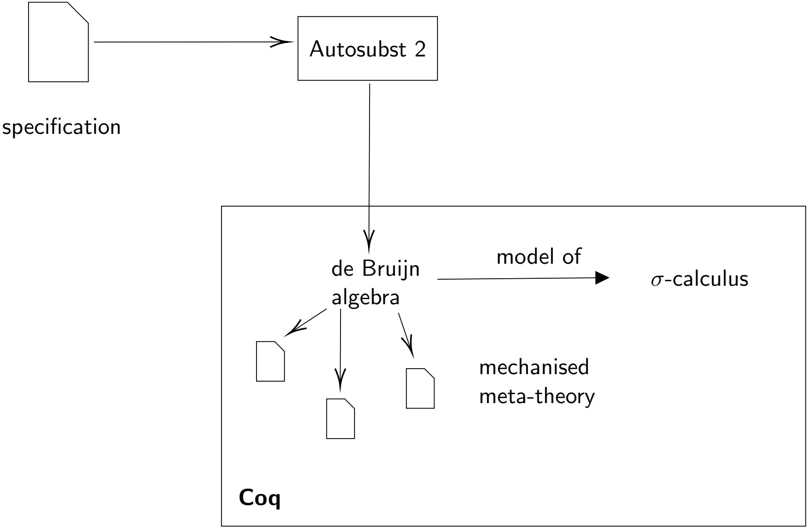


Best Practices and Automatic Realisation

1. What are the **best practices** for syntax with binders for more **extended systems**?
2. How to make this **reusable** and how to avoid boilerplate?
    - Development of **Autosubst 2** [Stark, Kaiser, Schäfer '19], a **compiler** from **HOAS-like syntax** [Pfenning, Elliot '88] to **de Bruijn algebras** + reasoning on de Bruijn algebras
    - Approach via **code generation** [Sewell et al. '07, Keuchel et al. '16]
    - The extended expressivity requires a fundamentally different design from Autosubst 1 [Schäfer, Tebbi, Smolka '15]

The Autosubst 2 Compiler [Stark, Kaiser, Schäfer '19]

A High-Level View



specification

Autosubst 2

de Bruijn algebra

model of

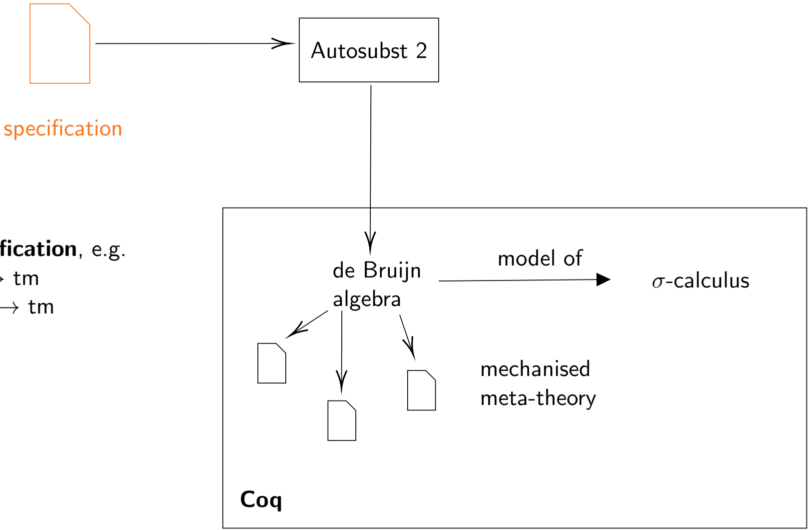
σ-calculus

mechanised meta-theory

Coq

The Autosubst 2 Compiler [Stark, Kaiser, Schäfer '19]

A High-Level View



specification

Autosubst 2

de Bruijn algebra

model of

σ-calculus

mechanised meta-theory

Coq

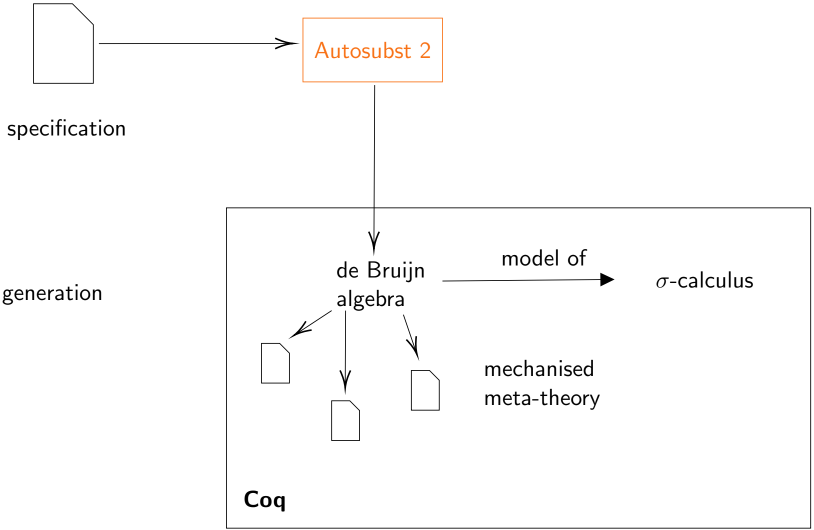
HOAS-like Specification, e.g.

app : tm → tm → tm

abs : (tm → tm) → tm

The Autosubst 2 Compiler [Stark, Kaiser, Schäfer '19]

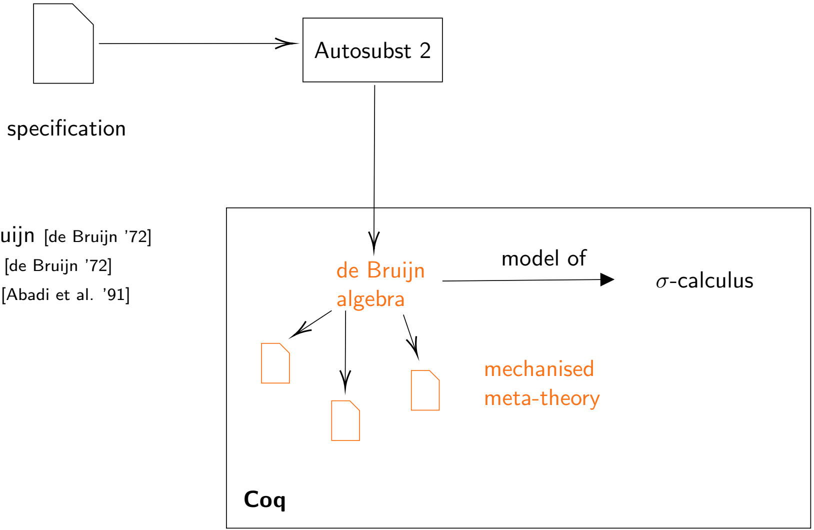
A High-Level View



External
Based on code generation

The Autosubst 2 Compiler [Stark, Kaiser, Schäfer '19]

A High-Level View



specification

Autosubst 2

de Bruijn algebra

model of

σ-calculus

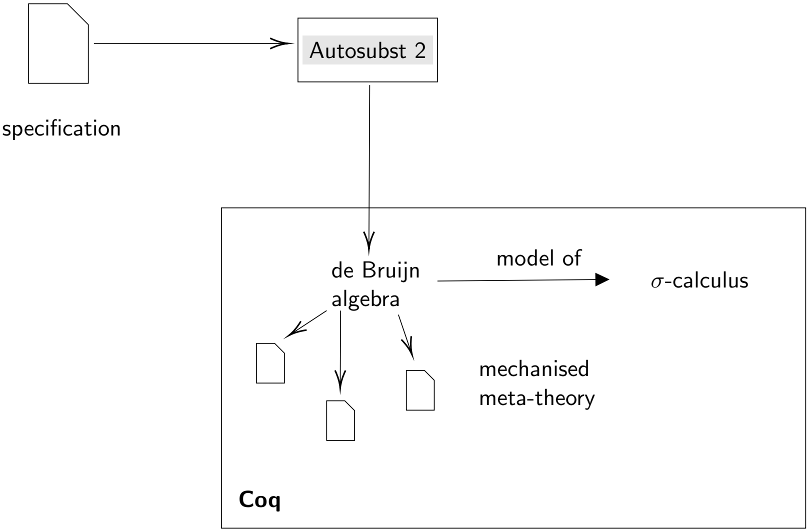
mechanised meta-theory

Coq

unscoped/scoped de Bruijn [de Bruijn '72]
+ parallel substitutions [de Bruijn '72]
+ laws of σ_{SP}-calculus [Abadi et al. '91]
+ notations and tactics

The Autosubst 2 Compiler [Stark, Kaiser, Schäfer '19]

A High-Level View



specification

Autosubst 2

de Bruijn algebra

model of

σ-calculus

mechanised meta-theory

Coq

The Autosubst 2 Tool

Generation of Code

HOAS Input

↓ Parsing/Analysis

Dependency Graph

↓ Proof Generation

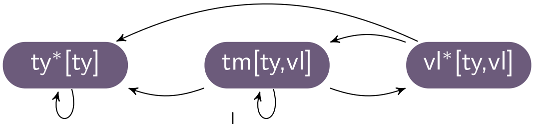
Abstract Proof Terms

↓ Code Generation

Coq Code

```
arr : ty → ty → ty
all : (ty → ty) → ty
...
```

↓



↓

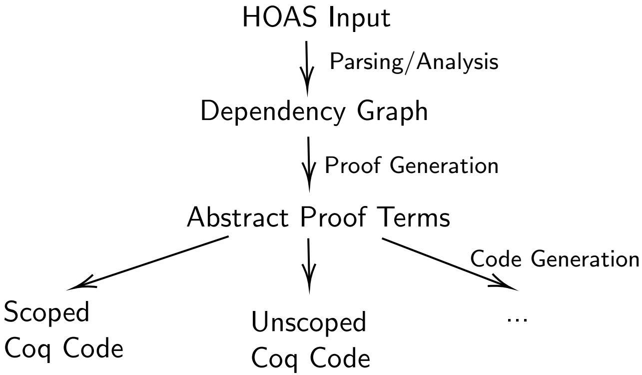
```
[SentenceInductive (Inductive
[InductiveBody "ty" [("n", TermConst Nat)]
    TermType [...]]),    ...]
```

↓

```
Inductive ty (n : nat) : Type :=
 | var_ty : fin n → ty n
 | arr : ty n → ty n → ty n
```

The Autosubst 2 Tool

Generation of Code



HOAS Input

Parsing/Analysis

Dependency Graph

Proof Generation

Abstract Proof Terms

Code Generation

Scoped
Coq Code

Unscoped
Coq Code

...

Best Practices and Automatic Realisation

1. What are the **best practices** for syntax with binders for more **extended systems**?
   - Applicable to: unscoped and **scoped** syntax [Bird, Paterson '99], **polyadic** binders, **first-class renamings**, **external sorts** and sort constructors, **many-sorted** syntax, **mutual inductive** syntax, **variadic** syntax, simplified definitions for **first-order** sorts, and **modular** syntax
2. How to make this **reusable** and how to avoid boilerplate?

What Is Needed for an Extension?

- **Three parts**:
  - Expressions
  - Instantiation with substitutions
  - Reasoning
- **Important:**
  - Restricted set of substitution primitives
  - These primitives are strong enough to express the whole scope change

Vector Substitutions [Stark, Kaiser, Schäfer '19]

Example: Call-by-Value System F ($\mathrm{F}_{\mathrm{CBV}}$)

Expressions

$$\begin{array}{rcll} A, B \in ty & ::= & X \mid A \to B \mid \forall X.A & \text{Types} \\ s, t \in tm & ::= & s\,t \mid s\,A \mid v & \text{Terms} \\ u, v \in vl & ::= & x \mid \lambda(x : A).s \mid \Lambda X.s & \text{Values} \end{array}$$

Substitutions *Vectorise* parallel substitutions:

$$\_[\_;\_] : \mathsf{vl} \to (\mathbb{N} \to \mathsf{ty}) \to (\mathbb{N} \to \mathsf{vl}) \to \mathsf{vl}$$

Reasoning Lift reasoning principles

Vector Substitutions [S., Kaiser, Schäfer '19]

Why Vector Substitutions?

Autosubst 1: Separate instantiation, e.g.

$$\_[\_]_{ty} : \mathsf{vl} \to (\mathbb{N} \to \mathsf{ty}) \to \mathsf{vl}$$
$$\_[\_]_{vl} : \mathsf{vl} \to (\mathbb{N} \to \mathsf{vl}) \to \mathsf{vl}$$

Problems:

- We might need instantiation on both sorts to go under binders
  $\Rightarrow$ No mutual inductive syntax
- How to get an elegant equational theory, e.g. how to commute the different kinds of instantiation?

$$s[\tau]_{vl}[\sigma]_{ty} = s[\sigma]_{ty}[\sigma \circ [\tau]_{ty}]_{vl}$$

Not all terms come with instantiation of all substitutions, e.g. for types:

$$\_[\_] : \mathsf{ty} \to (\mathbb{N} \to \mathsf{ty}) \to \mathsf{ty}$$

Variadic Syntax

- **Variadic binders** bind a variadic number of n variables at once, e.g. in a **multivariate λ-calculus** [Pottinger, '90]:

$$s, t \in tm_k ::= x \mid s^k\{t_1^k..t_n^k\} \mid \lambda_n.s^{n+k} \qquad\qquad x \in \mathbb{N}$$

- Other examples: **Pattern matching**, **recursive let-bindings**
- **Goal:** Omit arithmetic reasoning on indices

Variadic Syntax

Lifting the Monadic Primitives from the Sigma Calculus [Abadi et al. '91] to Variadic Primitives

1. **Variadic shifting** $\uparrow^m$: $\mathsf{fin}\,\mathsf{n} \to \mathsf{fin}\,(\mathsf{m}+\mathsf{n})$
2. **Variadic head**, $\mathsf{hd}_m : \mathsf{fin}\,\mathsf{m} \to \mathsf{fin}\,(\mathsf{m}+\mathsf{n})$
3. **Variadic extension** $\_\cdot_m\_ : (\mathsf{fin}\,\mathsf{m} \to X) \to (\mathsf{fin}\,\mathsf{n} \to X) \to (\mathsf{fin}\,(\mathsf{m}+\mathsf{n}) \to X)$, which precedes an arbitrary stream $\tau : \mathsf{fin}\,\mathsf{n} \to X$ with a new stream $\sigma : \mathsf{fin}\,\mathsf{m} \to X$:

+ definition of instantiation + adaption of reasoning principles

Modular Syntax [Forster, Stark '20]

The Expression Problem[Wadler, 2003]

- You start with the λ-calculus:

  $$s, t \in tm \;::=\; x \mid s\,t \mid \lambda x.s$$

  You give

  - recursive functions on terms,
  - proofs by induction on terms,
  - and predicates and proofs over the terms.

- . . . and then want to **extend** this calculus, e.g. by boolean expressions:

  $$s, t \in tm \;::=\; \cdots \mid b \mid \text{if } s \text{ then } t \text{ else } u$$

- **True modularity**: *"[..] add new cases to the datatype [..] without recompiling existing code."*

Modular Syntax [Forster, Stark '20]

A Practical Approach to Modular Syntax [Forster, Stark '20]

- Modular syntax via functors and variants with direct injections inspired by Data Types à la Carte [Swierstra '08]:

```
Inductive exp_λ (exp : Type) :=
  | var : nat → exp_λ exp
  | app : exp → exp → exp_λ exp
  | abs : exp → exp_λ exp.

Inductive exp :=
  | inj_λ : exp_λ exp → exp
  | inj_𝔹 :  exp → exp.
```

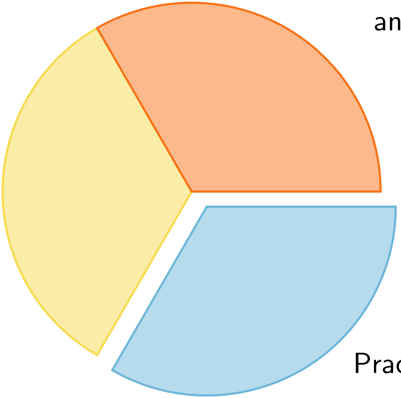
- **Tool support:**
  - Boilerplate generation with an extension of Autosubst 2
  - Assembling via MetaCoq [Sozeau et al., 2019]
- **Result:**
  - Practical modular developments
  - Improvement from 1000 loc/feature to 125 loc/feature

Modular Syntax [Forster, Stark '20]

Modular Syntax with Binders

1. **Expressions:**
    - Parameterised by full expressions
2. **Substitution:**
    - Parallel substitutions, parameterised by instantiation for full expressions
    - Same primitives of the σ-calculus
3. **Reasoning:**
    - Substitution laws for features parameterised by substitution laws for full expressions
    - Rewriting with substitution laws + equations for feature functions

Three Aspects of a Practical Presentation of Binders



Automatic Realisation
and Best Practices

Theoretical
Justification

Practical Evaluation

Practical Mechanisations of Meta-Theory

- **Goal:** See the performance in actual developments on the meta-theory
- **Criteria:**
  - Substitution-heavy case studies such as type safety or normalisation
  - Used in challenges, such as the POPLMark Challenge [Aydemir et al. '05] or POPLMark Reloaded Challenge [Abel, Allais, Hameer, Pientka, Momigliano, Schäfer, Stark 19]
- **Evaluation**:
  Concise, **transparent**, and **accessible**[1] proofs of type safety, equivalence of algorithmic and definitional equivalence, the meta-theory of call-by-push-value
  - Competitive on case studies tested with native support for binders except for names
  - Features of a general-purpose proof assistant were valuable
  - Built in new features such as support for first-order renamings or first-order sorts due to case studies

---

[1] [Aydemir et al. '05]

Case Studies for Autosubst 2

Contents	**Spec**	**Proofs**
POPLMark challenge, part A [Aydemir et al. '05]	151	165
Scoped variant of the POPLMark Reloaded Challenge, strong normalisation for STLC + Sums [Abel et al. '17]	248	312
Weak normalisation of call-by-value System F	114	60
Equivalence of algorithmic and definitional equivalence [Crary '05, Cave and Pientka '15]	88	135
Call-By-Push-Value [Levy '99, Forster, Schäfer, Spies, Stark '19]	3950	3750
Modular development of preservation/weak head normalisation/strong normalisation for a modular λ-calculus with boolean and arithmetic expressions[Forster and Stark '20]	540	655
First-order syntax [Kirst et al. '20]		
Undecidability of higher-order unification [Spies and Forster '20]		

Call-By-Push-Value in Coq [Forster, Schäfer, Spies, Stark '19]

Syntax

$$\begin{array}{lrl} \text{(value types)} & A, B & ::= 1 \mid U\,C \mid A_1 \times A_2 \mid 0 \mid A_1 + A_2 \\ \text{(computation types)} & C, D & ::= \top \mid F\,A \mid A \to C \mid C_1 \,\&\, C_2 \\ \text{(environments)} & \Gamma & ::= x_1 : A_1, \ldots, x_n : A_n \end{array}$$

Value typing $\boxed{\Gamma \vdash V : A}$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{}{\Gamma \vdash () : 1} \qquad \frac{\Gamma \vdash M : C}{\Gamma \vdash \{M\} : U\,C} \qquad \frac{\Gamma \vdash V_1 : A_1 \qquad \Gamma \vdash V_2 : A_2}{\Gamma \vdash (V_1, V_2) : A_1 \times A_2} \qquad \frac{\Gamma \vdash V : A_i}{\Gamma \vdash \mathsf{inj}\; i V : A_1 + A_2}$$

Computation typing $\boxed{\Gamma \vdash M : C}$

$$\frac{}{\Gamma \vdash \langle\rangle : \top} \qquad \frac{\Gamma \vdash V : A}{\Gamma \vdash \mathbf{return}\; V : F\,A} \qquad \frac{\Gamma \vdash M : F\,A \qquad \Gamma, x : A \vdash N : C}{\Gamma \vdash \mathbf{let}\; x \leftarrow M \;\mathbf{in}\; N : C} \qquad \frac{\Gamma, x : A \vdash M : C}{\Gamma \vdash \lambda x.M : A \to C} \qquad \frac{\Gamma \vdash M : A \to C \qquad \Gamma \vdash V : A}{\Gamma \vdash M\;V : C}$$

$$\frac{\Gamma \vdash V : U\,C}{\Gamma \vdash V! : C} \qquad \frac{\Gamma \vdash V : A_1 \times A_2 \qquad \Gamma, x_1 : A_1, x_2 : A_2 \vdash M : C}{\Gamma \vdash \mathbf{split}(V, x_1.x_2.M) : C} \qquad \frac{\Gamma \vdash V : 0}{\Gamma \vdash \mathbf{case_0}(V) : C}$$

$$\frac{\Gamma \vdash V : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash M_1 : C \qquad \Gamma, x_2 : A_2 \vdash M_2 : C}{\Gamma \vdash \mathbf{case}(V, x_1.M_1, x_2.M_2) : C} \qquad \frac{\Gamma \vdash M_1 : C_1 \qquad \Gamma \vdash M_2 : C_2}{\Gamma \vdash \langle M_1, M_2\rangle : C_1 \,\&\, C_2} \qquad \frac{\Gamma \vdash M : C_1 \,\&\, C_2}{\Gamma \vdash \mathbf{prj}_i\; M : C_i}$$

Call-By-Push-Value in Coq [Forster, Schäfer, Spies, S '19]

Mechanisation in 8000 lines of Coq code of

- standard operational semantics for CBPV
  - normalisation using logical relations
  - adequacy of set/algebra semantics
- unrestricted operational semantics for CBPV
  - confluence
  - strong normalisation using Kripke logical relations
  - soundness of equational theory
- translations of CBV/CBN into CBPV
  - preservation of operational semantics
  - confluence for full λ-calculus
  - strong normalisation for strong CBV/CBN
  - soundness of equational theories
  - adequate type-theoretic algebra semantics for CBV/CBN

Binders, substitutions, and reasoning is automated by Autosubst 2.

Conclusion

Conclusion

Main Contributions

- Mechanised proof that the σ_{SP}-calculus is **confluent and terminating**, following and simplifying a previous proof by [Curien et al. '92]
- Development of **Autosubst 2**
  - Introduction of EHOAS as specification language
  - Handling of polyadic binders, first-class renamings, vector substitutions, with only first-order binders, syntax with variadic binders
- Combination of Autosubst and **modular syntax** with an approached based on functors and direct injections
- Mechanised meta-theory
  - Concise, transparent, and accessible [Ayedemir et al. '05] proofs of strong normalisation
  - Solution to the substitution-relevant parts of the POPLMark Challenge and POPLMark Reloaded Challenge
  - First truly modular proof of type safety and strong normalisation

Conclusion

Future Work

- Calculi of Explicit Substitutions
  - Formal justification for **custom syntax**, see e.g. [Keuchel et al. '18]
  - How do other **variants of calculi of explicit substitutions**, e.g. the shift calculus [Hardin and Lévy, '89] work?
- Compiling syntactic specifications
  - Extend the expressiveness even further, e.g. by **dependent predicates** [Keuchel et al. '18]
  - Support for **recursive functions** [Allais et al. '17, Kaiser, Schäfer, Stark '18]
  - Support for switching to a **named representation**
- Modular syntax
  - Support for **scoped syntax** and **dependent predicates**
  - Solutions to the POPLMark challenge [Ayedemir et al. '05] and POPLMark Reloaded Challenge [Abel et al. '19]
  - Support of **more dimensions of modularity** [Delaware et al. '13]

Conclusion

Main Contributions

- Mechanised proof that the σ_{SP}-calculus is **confluent and terminating**, following and simplifying a previous proof by [Curien et al. '92]
- Development of **Autosubst 2**
  - Introduction of EHOAS as specification language
  - Handling of polyadic binders, first-class renamings, vector substitutions, with only first-order binders, syntax with variadic binders
- Combination of Autosubst and **modular syntax** with an approached based on functors and direct injections
- Mechanised meta-theory
  - Concise, transparent, and accessible [Ayedemir et al. '05] proofs of strong normalisation
  - Solution to the substitution-relevant parts of the POPLMark Challenge and POPLMark Reloaded Challenge
  - First truly modular proof of type safety and strong normalisation

Thank you for your attention!

Publications

- Jonas Kaiser, Steven Schafer, and Kathrin Stark. Autosubst 2: Towards reasoning with multi-sorted de Bruijn terms and vector substitutions. In Proceedings of the Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP '17, pages 10–14. ACM, 2017
- Jonas Kaiser, Steven Schafer, and Kathrin Stark. Binder aware recursion over well- scoped de Bruijn syntax. In Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018, pages 293–306, 2018
- Steven Schafer and Kathrin Stark.Embedding higher-order abstract syntax in type theory. In Abstract for Types Workshop, June 18 – 21 2018
- Kathrin Stark, Steven Schafer, and Jonas Kaiser. Autosubst 2 :reasoning with multi- sorted de Bruijn terms and vector substitutions. In Proceedings of the 8th ACM SIG- PLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019, pages 166–180, 2019
- Yannick Forster, Steven Schafer, Simon Spies, and Kathrin Stark. Call-by-push- value in Coq: operational, equational, and denotational theory. In Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019, pages 118–131, 2019
- Andreas Abel, Guillaume Allais, Aliya Hameer, Brigitte Pientka, Alberto Momigliano, Steven Schafer, and Kathrin Stark. POPLMark Reloaded: Mechanizing proofs by logical relations. Journal of Functional Programming, 29:e19, 2019
- Yannick Forster and Kathrin Stark. Coq a la carte- a practical approach to modular syntax with binders. In Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, USA, January 20–21, 2020, January 2020