Autosubst 2:
Reasoning with Multi-Sorted de Bruijn Terms and Vector Substitutions

Kathrin Stark, Steven Schäfer, Jonas Kaiser

SAARLAND UNIVERSITY

SIC Saarland Informatics Campus

CPP 2019
January 15

Our Motivation: The Formalisation of Metatheory

- Of programming languages and logical systems with binders,
  - e.g. Call-By-Push-Value (CBPV):

$$
\begin{array}{rcll}
V \in \mathit{vl} & ::= & x \mid () \mid (V_1, V_2) \mid \mathsf{inj}_i\, V \mid \{M\} & \text{values} \\
M \in \mathit{tm} & ::= & M\, V \mid \lambda x.M & \\
 & & \mid \mathsf{let}\, x \leftarrow M_1\, \mathsf{in}\, M_2 & \\
 & & \mid \mathsf{split}(V, x_1.x_2.M) & \\
 & & \mid \mathsf{case}\,(V, x_1.M_1, x_2.M_2) \mid V! \ldots & \text{computations}
\end{array}
$$

- Formalising proofs as
  - weak and strong normalisation
  - CBV and CBN can be simulated in CBPV

Our Motivation: The Formalisation of Metatheory

- Of programming languages and logical systems with binders,
  - e.g. Call-By-Push-Value (CBPV):

$$
\begin{array}{rcll}
V \in \mathit{vl} & ::= & x \mid () \mid (V_1, V_2) \mid \mathsf{inj}_i\, V \mid \{M\} & \text{values} \\
M \in \mathit{tm} & ::= & M\, V \mid \lambda x.M & \\
& & \mid \mathsf{let}\, x \leftarrow M_1\, \mathsf{in}\, M_2 & \\
& & \mid \mathsf{split}(V, x_1.x_2.M) & \\
& & \mid \mathsf{case}\,(V, x_1.M_1, x_2.M_2) \mid V! \ldots & \text{computations}
\end{array}
$$

- Formalising proofs as
  - weak and strong normalisation
  - CBV and CBN can be simulated in CBPV

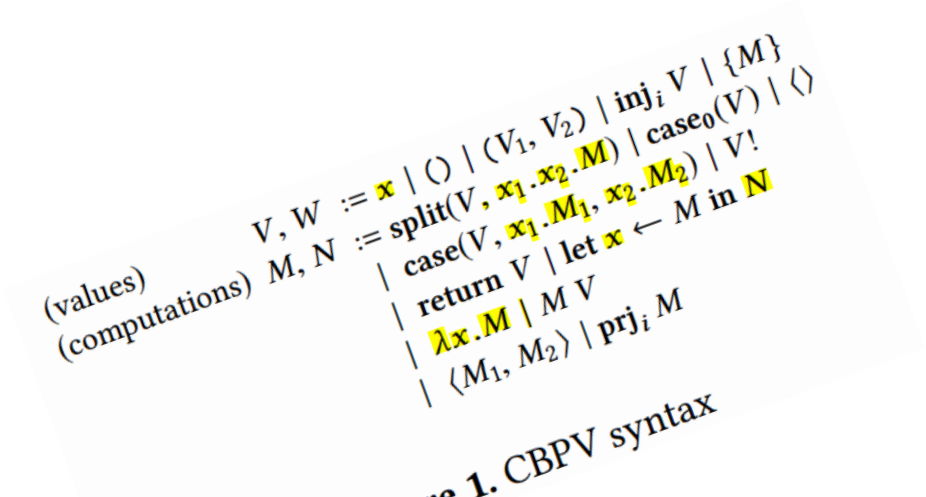
Our Motivation: The Formalisation of Metatheory

- Of programming languages and logical systems with binders,
  - e.g. Call-By-Push-Value (CBPV):

$$
\begin{array}{rcll}
V \in \mathit{vl} & ::= & x \mid () \mid (V_1, V_2) \mid \mathsf{inj}_i\, V \mid \{M\} & \text{values} \\
M \in \mathit{tm} & ::= & M\, V \mid \lambda x.M & \\
 & & \mid \mathsf{let}\, x \leftarrow M_1\, \mathsf{in}\, M_2 & \\
 & & \mid \mathsf{split}(V, x_1.x_2.M) & \\
 & & \mid \mathsf{case}\,(V, x_1.M_1, x_2.M_2) \mid V! \ldots & \text{computations}
\end{array}
$$

- Formalising proofs as
  - weak and strong normalisation
  - CBV and CBN can be simulated in CBPV

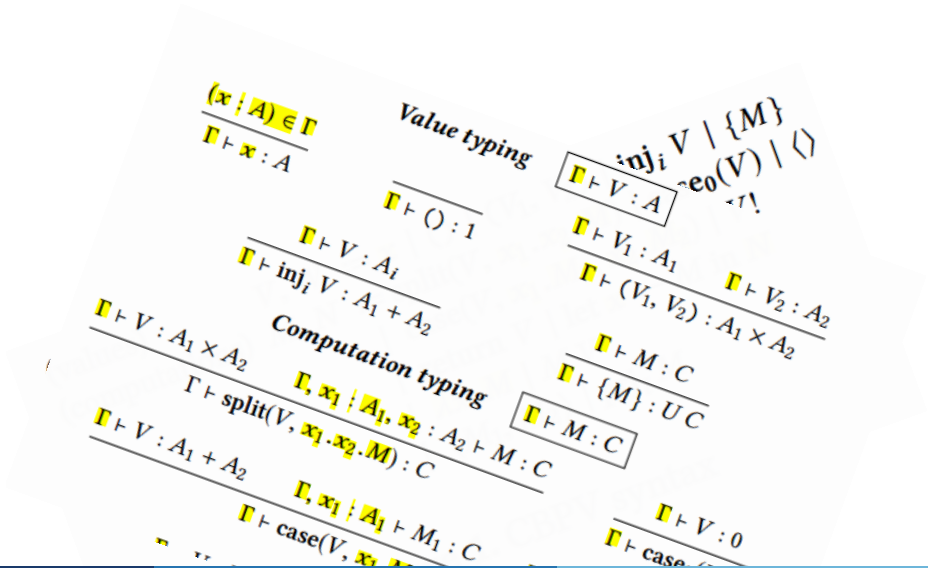
Binders Come With Boilerplate – Strong Normalization for CBPV



re 1. CBPV syntax

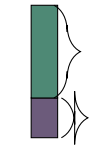
Expressions

Binders Come With Boilerplate – Strong Normalization for CBPV



Expressions

Binders Come With Boilerplate – Strong Normalization for CBPV



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

Value typing

$$M \succ M'$$

Primitive reduction

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

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

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

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

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

$$\langle M_1, M_2 \rangle \succ M_i$$

Binders Come With Boilerplate – Strong Normalization for CBPV

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

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

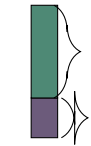
Semantic Typing

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

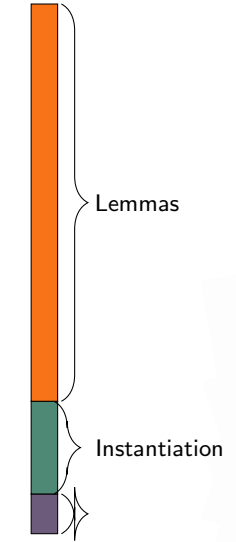
$$\mathcal{G}[\Gamma] := \{\gamma \mid \forall (x : A) \in \Gamma,\ \gamma x \in \mathcal{V}[A]\}$$

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

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



Binders Come With Boilerplate – Strong Normalization for CBPV



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

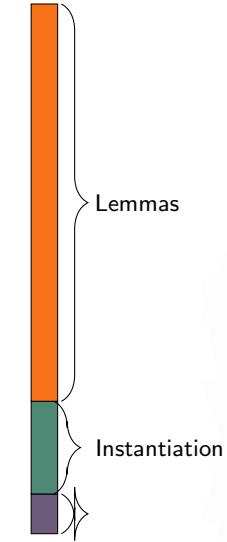
$$C[C_1 \,\&\, C_2] := \{(M_1, M_2) \mid M_1 \in \mathcal{E}$$

Lemma 4.1. *The following hold:*

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

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

Binders Come With Boilerplate – Strong Normalization for CBPV



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

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

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

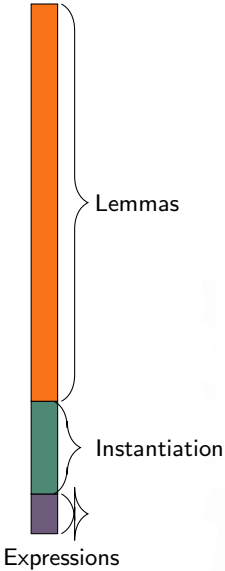
Lemma 4.1.

1. $C[C] \subseteq \mathcal{E}[$
2. $C[C]$ *only con*

If $M \rightsquigarrow^* N$ *and* N

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

Binders Come With Boilerplate – Strong Normalization for CBPV



$$c[\text{var } 0,\ \gamma \circ \langle\uparrow\rangle][v, ids] = c[v, \gamma]?$$

Binders Come With Boilerplate – Strong Normalization for CBPV

$$c[\mathsf{var}\ 0,\ \gamma \circ \langle\uparrow\rangle][v, ids] = c[v, \gamma]?$$

Related Work: Various Ways to Represent Binders

- parallel de Bruijn [de Bruijn '72]
- single-point de Bruijn
- nominal logic [Pitts '01]
- HOAS [Pfenning et al. '88]
- locally nameless [Aydemir et al. '08]
- contextual modalTT [Nanevski et al. '08]
- ...

Related Work: Various Ways to Represent Binders

- parallel de Bruijn [de Bruijn '72]
- single-point de Bruijn
- nominal logic [Pitts '01]
- HOAS [Pfenning et al. '88]
- locally nameless [Aydemir et al. '08]
- contextual modalTT [Nanevski et al. '08]
- ...

Our requirements:

- Works in a general-purpose proof assistant/Coq
- Little set-up
- Little overhead in the proofs

Related Work: Various Ways to Represent Binders

- parallel de Bruijn [de Bruijn '72]
- single-point de Bruijn
- nominal logic [Pitts '01]
- HOAS [Pfenning et al. '88]
- locally nameless [Aydemir et al. '08]
- contextual modalTT [Nanevski et al. '08]
- ...

Our requirements:

- Works in a general-purpose proof assistant/Coq
- Little set-up
- Little overhead in the proofs

Extension of Autosubst 1 [Schäfer/Tebbi/Smolka '15]

The Autosubst Workflow

Signature file, e.g.

$\texttt{app} : \texttt{term} \to \texttt{term} \to \texttt{term}$
$\texttt{lam} : (\texttt{term} \to \texttt{term}) \to \texttt{term}$

Autosubst 2 compiler

Output file
Instantiation $\_[\_]$
+ standard operations
+ automation

The Autosubst Workflow

Signature file, e.g.

$\texttt{app} : \texttt{term} \to \texttt{term} \to \texttt{term}$
$\texttt{lam} : (\texttt{term} \to \texttt{term}) \to \texttt{term}$

} Second-Order HOAS signature,
potentially multivariate and mutual inductive

⇓ Autosubst 2 compiler

Output file
Instantiation $\_[\_]$
+ standard operations
+ automation

The Autosubst Workflow

Signature file, e.g.

```
app : term → term → term
lam : (term → term) → term
```

Second-Order HOAS signature,
potentially multivariate and mutual inductive

⇓ Autosubst 2 compiler

Multi-layer compiler

Output file

Instantiation _[_]
+ standard operations
+ automation

The Autosubst Workflow

Signature file, e.g.

$\texttt{app} : \texttt{term} \to \texttt{term} \to \texttt{term}$
$\texttt{lam} : (\texttt{term} \to \texttt{term}) \to \texttt{term}$

Second-Order HOAS signature,
potentially multivariate and mutual inductive

⇓ Autosubst 2 compiler

Multi-layer compiler

Output file
Instantiation $\_[\_]$
+ standard operations
+ automation

parallel de Bruijn [de Bruijn '72]

- renamings + substitutions
- unscoped or well-scoped

+ σ-calculus [Abadi et al. '91]:
solve equations of the form $s = t$

The Autosubst Workflow

Signature file, e.g.

$\texttt{app} : \texttt{term} \to \texttt{term} \to \texttt{term}$
$\texttt{lam} : (\texttt{term} \to \texttt{term}) \to \texttt{term}$

Second-Order HOAS signature,
potentially multivariate and mutual inductive

⇓ Autosubst 2 compiler

Multi-layer compiler

Output file
Instantiation $\_[\_]$
+ standard operations
+ automation

parallel de Bruijn [de Bruijn '72]

- renamings + substitutions
- unscoped or well-scoped

+ σ-calculus [Abadi et al. '91]:
solve equations of the form $s = t$

Tool / case studies : www.ps.uni-saarland.de/extras/autosubst2

The Autosubst Workflow – Autosubst 1 vs. Autosubst 2

Signature file, e.g.

```
app : term → term → term
lam : (term → term) → term
```

Second-Order HOAS signature,

potentially multivariate and mutual inductive

⇓ Autosubst 2 compiler

Multi-layer compiler

Output file
Instantiation _[_]
+ standard operations
+ automation

parallel de Bruijn [de Bruijn '72]

- renamings + substitutions
- unscoped or well-scoped

+ σ-calculus [Abadi et al. '91]:
solve equations of the form $s = t$

Tool / case studies : www.ps.uni-saarland.de/extras/autosubst2

Organisation of the Talk

Signature file, e.g.

```
app : term → term → term
lam : (term → term) → term
```

Second-Order HOAS signature,

potentially multivariate and mutual inductive ❹

⇓ Autosubst 2 compiler

Multi-layer compiler ❷

Output file
Instantiation _[_]
+ standard operations
+ automation

parallel de Bruijn [de Bruijn '72]

- renamings + substitutions
- unscoped or well-scoped ❸

+ σ-calculus [Abadi et al. '91]:
solve equations of the form $s = t$

Tool ❶/ case studies ❺: www.ps.uni-saarland.de/extras/autosubst2

Organisation of the Talk

Signature file, e.g.

$\texttt{app} : \texttt{term} \to \texttt{term} \to \texttt{term}$
$\texttt{lam} : (\texttt{term} \to \texttt{term}) \to \texttt{term}$

Second-Order HOAS signature,
potentially multivariate and mutual inductive ❹

⇓ Autosubst 2 compiler

Multi-layer compiler ❷

Output file
Instantiation $\_[\_]$
+ standard operations
+ automation

parallel de Bruijn [de Bruijn '72]
- renamings + substitutions
- unscoped or well-scoped ❸

+ σ-calculus [Abadi et al. '91]:
solve equations of the form $s = t$

Tool ❶/ case studies ❺: www.ps.uni-saarland.de/extras/autosubst2

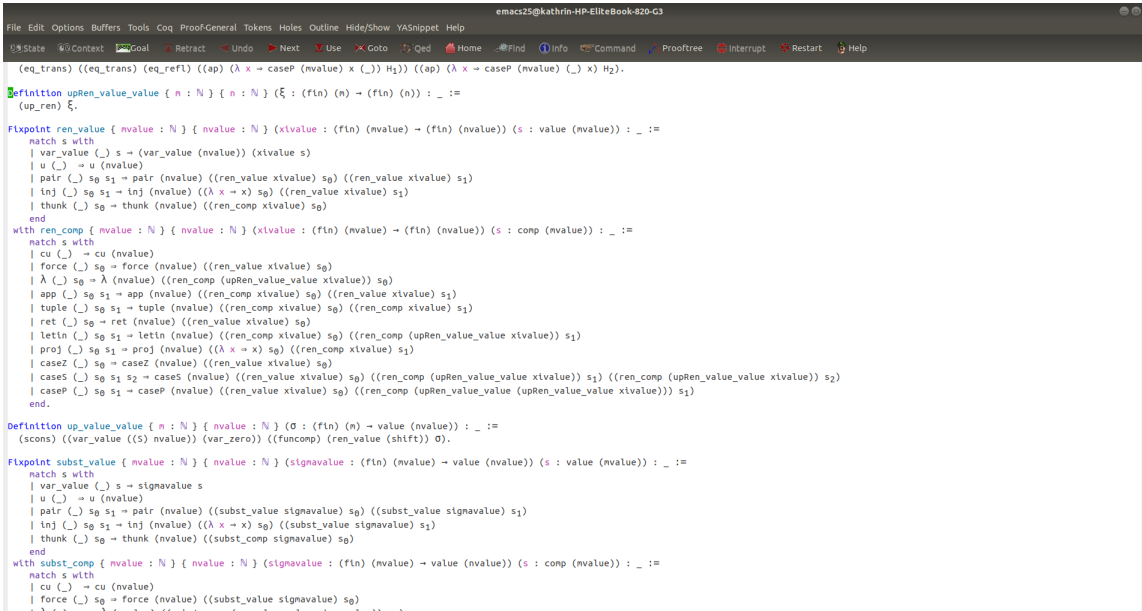
Revisiting Strong Normalization for CBPV

Revisiting Strong Normalization for CBPV

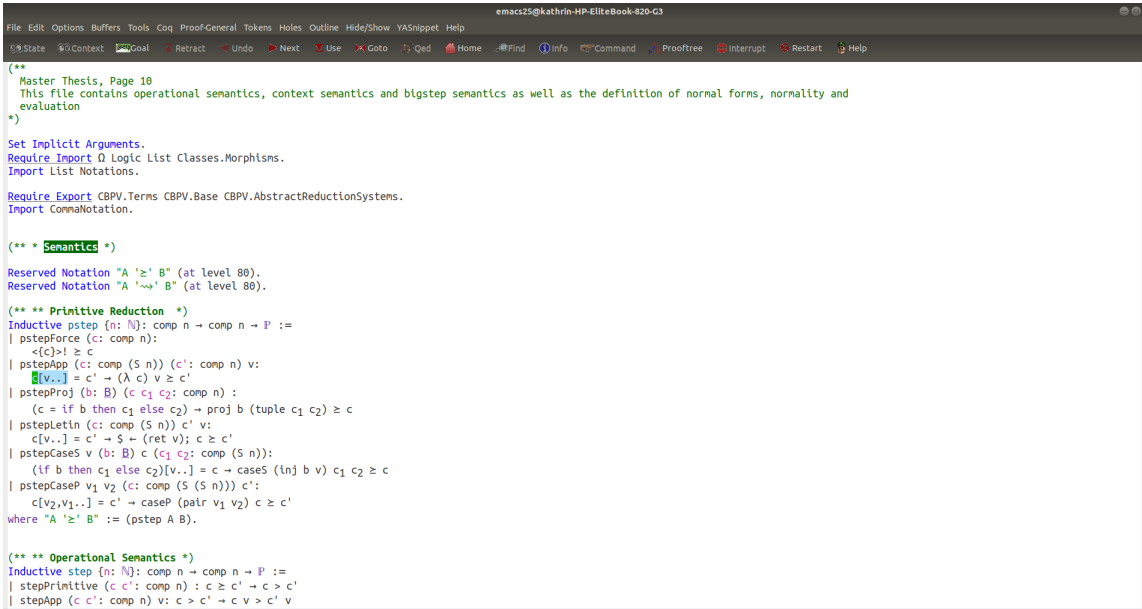
Open ▾ | cbpv.sig | ~/Documents/Git Repos Redmine/autosubst-2-ref…t-2/as2-with-functors/examples/coq/scoped-cbpv | Save

```
valtype : Type
comptype : Type
value : Type
comp : Type
bool : Type


zero: valtype
one: valtype
U: comptype -> valtype
Sigma: valtype -> valtype -> valtype
cross: valtype -> valtype -> valtype


cone: comptype
F: valtype -> comptype
Pi: comptype -> comptype -> comptype
arrow: valtype -> comptype -> comptype


u: value
pair: value -> value -> value
inj: bool -> value -> value
```

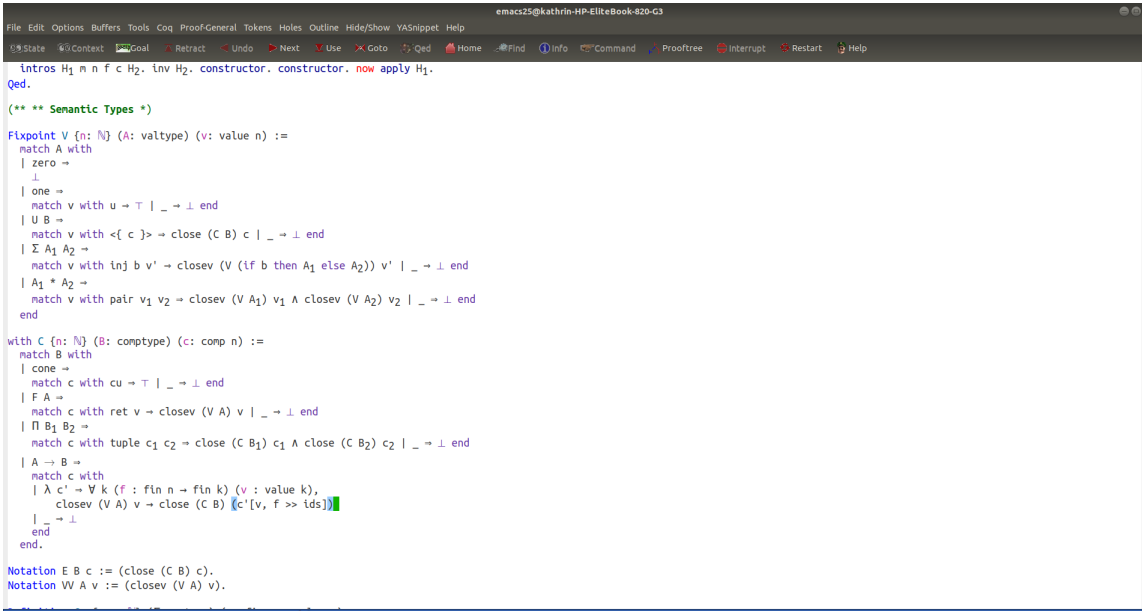
Revisiting Strong Normalization for CBPV



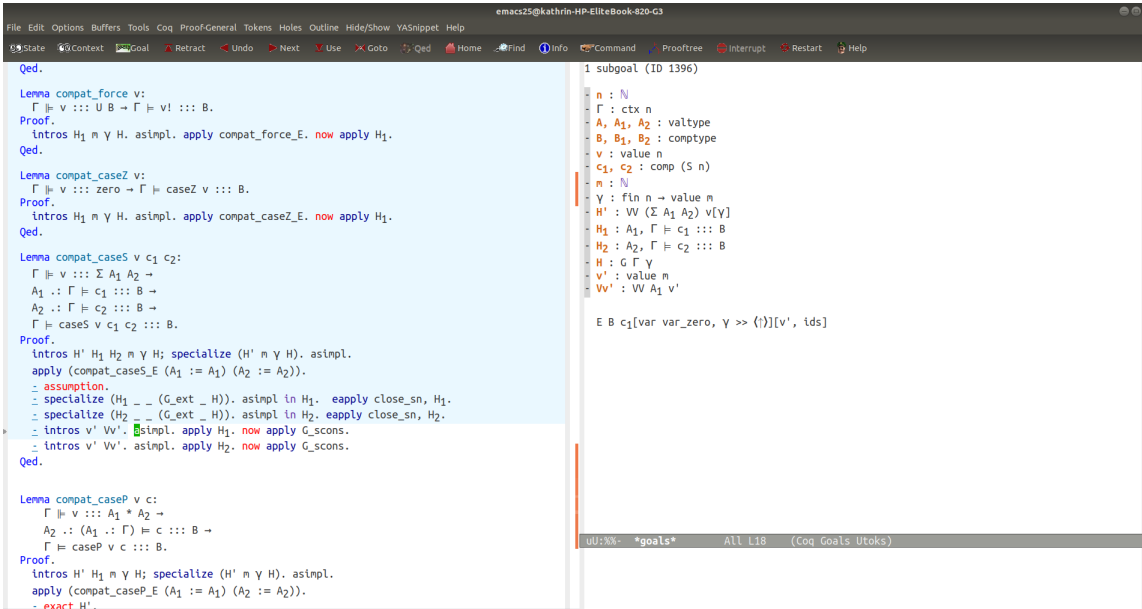
Revisiting Strong Normalization for CBPV



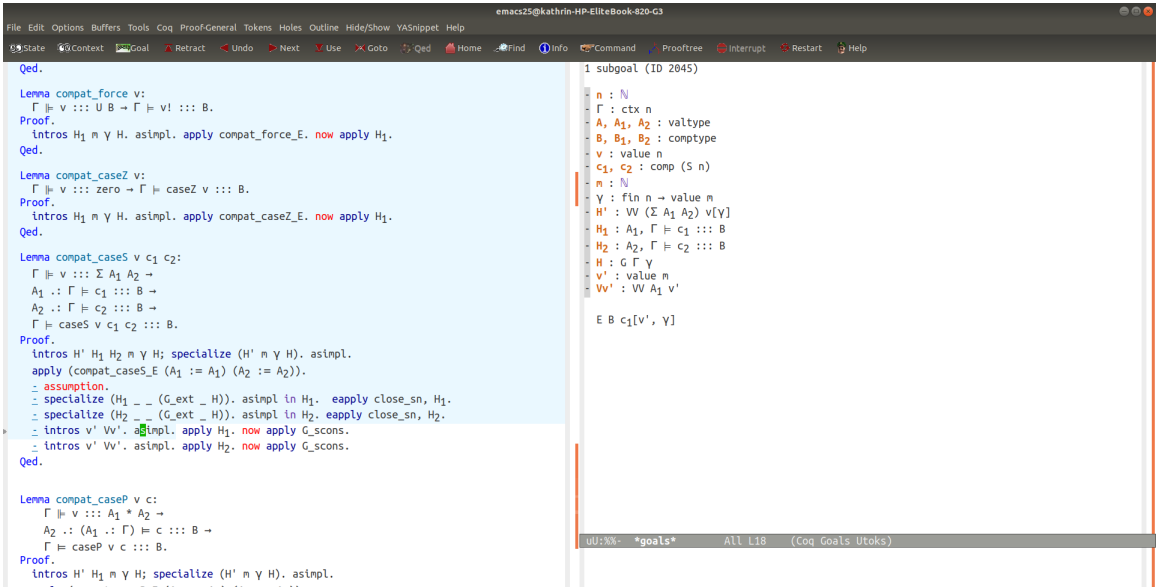
Revisiting Strong Normalization for CBPV



Revisiting Strong Normalization for CBPV

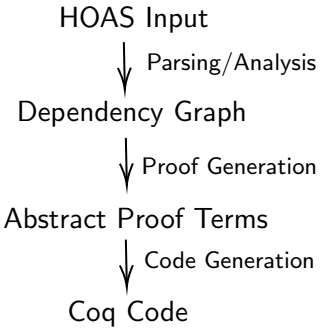


Revisiting Strong Normalization for CBPV

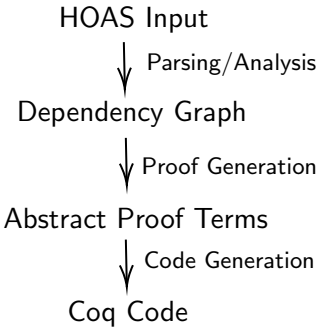


The Autosubst 2 Compiler

Generation of Code

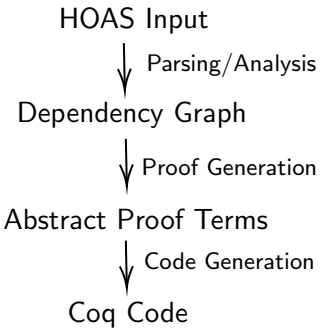


Generation of Code

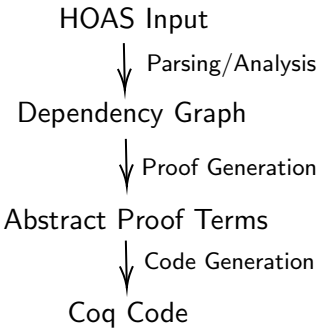


Generation of Code

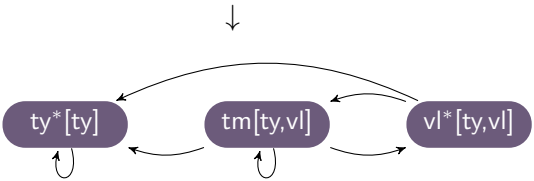
$$\texttt{arr} : \texttt{ty} \to \texttt{ty} \to \texttt{ty}$$
$$\texttt{all} : (\texttt{ty} \to \texttt{ty}) \to \texttt{ty}$$
$$\ldots$$



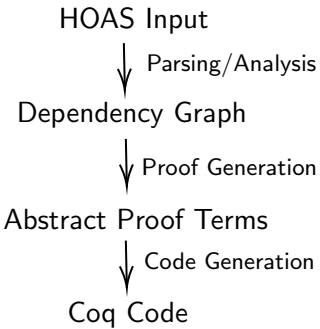
Generation of Code



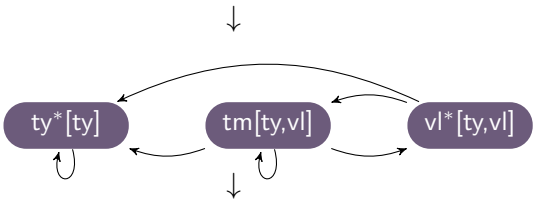

```
arr : ty → ty → ty
all : (ty → ty) → ty
...
```



Generation of Code




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




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

Generation of Code

HOAS Input

↓ Parsing/Analysis

Dependency Graph

↓ Proof Generation

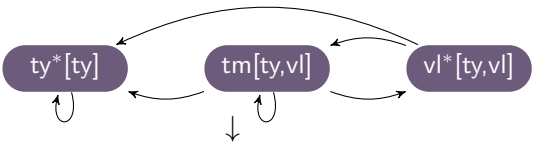
Abstract Proof Terms

↓ 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
 | all : ty (1 + n) → ty n.
```

Well-Scoped Syntax

Well-Scoped Syntax [Bird/Paterson '99]

Example

Terms are indexed by the number of free variables:

```
Inductive tm (n : nat) : Type :=
  | var_tm : fin n → tm n
  | app : tm n → tm n → tm n
  | lam : tm (1 + n) → tm n.
```

Well-Scoped Syntax [Bird/Paterson '99]

- Allows functional reasoning:

Well-Scoped Syntax [Bird/Paterson '99]

- Allows functional reasoning:

$$(\_ \vdash \_ : \_) : \mathcal{L}(\mathsf{tm}) \to \mathsf{tm} \to \mathsf{ty} \to \mathcal{P}$$

Well-Scoped Syntax [Bird/Paterson '99]

- Allows functional reasoning:

$$(\_ \vdash \_ : \_) : \mathcal{L}(\mathsf{tm}) \to \mathsf{tm} \to \mathsf{ty} \to \mathcal{P}$$

$$(\_ \vdash \_ : \_) : (\mathsf{fin}\ n \to \mathsf{ty}\ m) \to \mathsf{tm}\ m\ n \to \mathsf{ty}\ n \to \mathcal{P}$$

Well-Scoped Syntax [Bird/Paterson '99]

- Allows functional reasoning:

$$(\_ \vdash \_ : \_) : \mathcal{L}(\mathsf{tm}) \to \mathsf{tm} \to \mathsf{ty} \to \mathcal{P}$$

$$(\_ \vdash \_ : \_) : (\mathsf{fin}\ n \to \mathsf{ty}\ m) \to \mathsf{tm}\ m\ n \to \mathsf{ty}\ n \to \mathcal{P}$$

$$\frac{x < |\Gamma|}{\Gamma \vdash x : \mathsf{nth}\ x\ \Gamma} \qquad \text{vs.} \qquad \frac{}{\Gamma \vdash x : \Gamma\ x}$$

Well-Scoped Syntax [Bird/Paterson '99]

- Allows functional reasoning:

$$(\_ \vdash \_ : \_) : \mathcal{L}(\mathsf{tm}) \to \mathsf{tm} \to \mathsf{ty} \to \mathcal{P}$$

$$(\_ \vdash \_ : \_) : (\mathsf{fin}\ n \to \mathsf{ty}\ m) \to \mathsf{tm}\ m\ n \to \mathsf{ty}\ n \to \mathcal{P}$$

$$\frac{x < |\Gamma|}{\Gamma \vdash x : \mathsf{nth}\ x\ \Gamma} \quad \text{vs.} \quad \frac{}{\Gamma \vdash x : \Gamma\ x}$$

$$\frac{\mathsf{map}\ \langle\uparrow\rangle\ \Gamma \vdash s : A}{\Gamma \vdash \Lambda.s : \forall.A} \quad \text{vs.} \quad \frac{\Gamma \circ \langle\uparrow\rangle \vdash s : A}{\Gamma \vdash \Lambda.s : \forall.A}$$

Well-Scoped Syntax [Bird/Paterson '99]

- Allows functional reasoning:

$$(\_ \vdash \_ : \_) : \mathcal{L}(\mathsf{tm}) \to \mathsf{tm} \to \mathsf{ty} \to \mathcal{P}$$

$$(\_ \vdash \_ : \_) : (\mathsf{fin}\ n \to \mathsf{ty}\ m) \to \mathsf{tm}\ m\ n \to \mathsf{ty}\ n \to \mathcal{P}$$

$$\frac{x < |\Gamma|}{\Gamma \vdash x : \mathsf{nth}\ x\ \Gamma} \quad \text{vs.} \quad \frac{}{\Gamma \vdash x : \Gamma\ x}$$

$$\frac{\mathsf{map}\ \langle\uparrow\rangle\ \Gamma \vdash s : A}{\Gamma \vdash \Lambda.s : \forall.A} \quad \text{vs.} \quad \frac{\Gamma \circ \langle\uparrow\rangle \vdash s : A}{\Gamma \vdash \Lambda.s : \forall.A}$$

- Statements about closed terms:

Well-Scoped Syntax [Bird/Paterson '99]

- Allows functional reasoning:

$$(\_ \vdash \_ : \_) : \mathcal{L}(\mathsf{tm}) \to \mathsf{tm} \to \mathsf{ty} \to \mathcal{P}$$

$$(\_ \vdash \_ : \_) : (\mathsf{fin}\ n \to \mathsf{ty}\ m) \to \mathsf{tm}\ m\ n \to \mathsf{ty}\ n \to \mathcal{P}$$

$$\frac{x < |\Gamma|}{\Gamma \vdash x : \mathsf{nth}\ x\ \Gamma} \quad \text{vs.} \quad \frac{}{\Gamma \vdash x : \Gamma\ x}$$

$$\frac{\mathsf{map}\ \langle\uparrow\rangle\ \Gamma \vdash s : A}{\Gamma \vdash \Lambda.s : \forall.A} \quad \text{vs.} \quad \frac{\Gamma \circ \langle\uparrow\rangle \vdash s : A}{\Gamma \vdash \Lambda.s : \forall.A}$$

- Statements about closed terms:

$$\forall (s : \mathsf{tm}\ m\ 0)\ A.\ \vdash s : A \to \mathsf{SN}\ s$$

Well-Scoped Syntax [Bird/Paterson '99]

- More type safety:

Well-Scoped Syntax [Bird/Paterson '99]

- More type safety:

$$\frac{\Gamma \vdash s : A}{\Gamma \vdash \Lambda X.s : \forall X.A} \qquad \text{vs.} \qquad \frac{\Gamma \circ \langle \uparrow \rangle \vdash s : A}{\Gamma \vdash \Lambda.s : \forall.A}$$

Well-Scoped Syntax [Bird/Paterson '99]

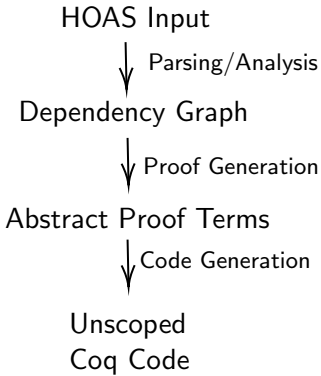
- More type safety:

$$\frac{\Gamma \vdash s : A}{\Gamma \vdash \Lambda X.s : \forall X.A} \quad \text{vs.} \quad \frac{\Gamma \circ \langle\uparrow\rangle \vdash s : A}{\Gamma \vdash \Lambda.s : \forall.A}$$

$$\begin{aligned} \overline{M\,N} = \text{let } x \leftarrow \overline{M} \text{ in} \\ \text{let } y \leftarrow \overline{N} \text{ in} \\ (!x)\,y \end{aligned} \quad \text{vs.} \quad \begin{aligned} \overline{M\,N} = \text{let } \leftarrow \overline{M} \text{ in} \\ \text{let } \leftarrow \overline{N}\langle\uparrow\rangle \text{ in} \\ (!1)\,0 \end{aligned}$$

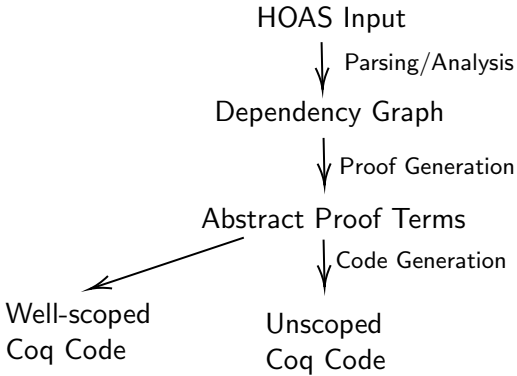
Well-Scoped Syntax

Implementation



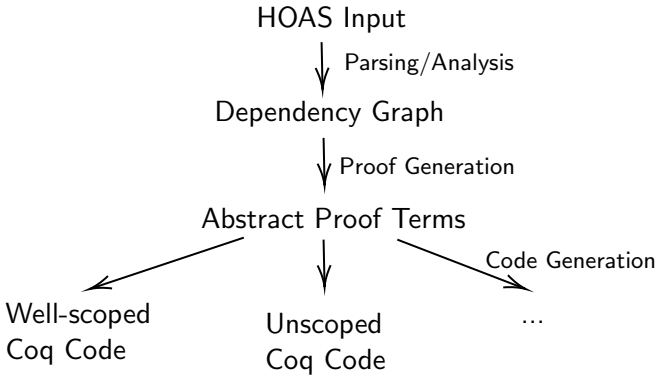
Well-Scoped Syntax

Implementation



Well-Scoped Syntax

Implementation



Vector Substitutions

Vector Substitutions

Motivation

How to formalize instantiation for multivariate, potentially mutually inductive systems – e.g. call-by-value System F ($\mathrm{F}_{\mathrm{CBV}}$)?

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

Vector Substitutions

Motivation

How to formalize instantiation for multivariate, potentially mutually inductive systems – e.g. call-by-value System F ($\mathrm{F}_{\mathrm{CBV}}$)?

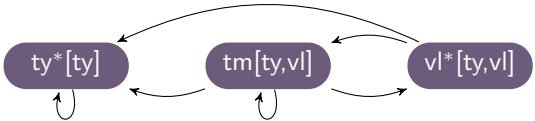
$$
\begin{array}{rcll}
A, B \in ty & ::= & X \mid A \to B \mid \forall X.A & \text{Types} \\
s, t \in tm & ::= & s\,t \mid s\,A \mid v & \text{Terms} \\
u, v \in vl & ::= & x \mid \lambda(x : A).s \mid \Lambda X.s & \text{Values}
\end{array}
$$

Solution: *Vectorise* parallel substitutions:

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

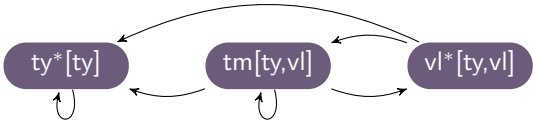
Vector Substitutions

Implementation



Vector Substitutions

Implementation

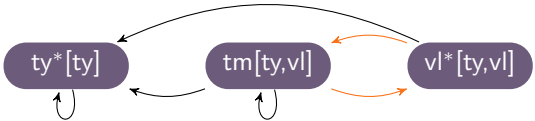


$$x[\sigma, \tau] = \tau\, x$$
$$(\lambda A.\ s)[\sigma, \tau] = \lambda A[\sigma].\ s[\Uparrow^{vl}_{tm}\ (\sigma, \tau)]$$
$$(\Lambda.\ s)[\sigma, \tau] = \Lambda.\ s[\Uparrow^{ty}_{tm}\ (\sigma, \tau)]$$

- Traverses values
  - homomorphically

Vector Substitutions

Implementation

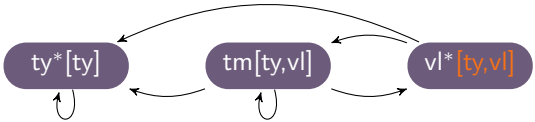


$$x[\sigma, \tau] = \tau\, x$$
$$(\lambda A.\ s)[\sigma, \tau] = \lambda A[\sigma].\ s[\Uparrow^{vl}_{tm}\ (\sigma, \tau)]$$
$$(\Lambda.\ s)[\sigma, \tau] = \Lambda.\ s[\Uparrow^{ty}_{tm}\ (\sigma, \tau)]$$

- Traverses values
  - homomorphically
  - mutually recursive

Vector Substitutions

Implementation

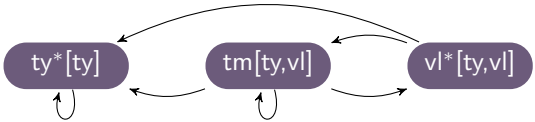


$$x[\sigma,\tau] = \tau\, x$$
$$(\lambda A.\ s)[\sigma,\tau] = \lambda A[\sigma].\ s[\Uparrow^{vl}_{tm}\ (\sigma,\tau)]$$
$$(\Lambda.\ s)[\sigma,\tau] = \Lambda.\ s[\Uparrow^{ty}_{tm}\ (\sigma,\tau)]$$

- Traverses values
  - homomorphically
  - mutually recursive
  - with the inferred vector

Vector Substitutions

Implementation

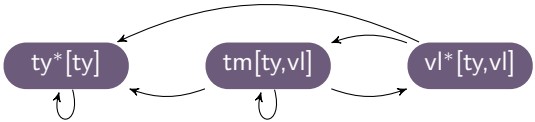


$$x[\sigma, \tau] = \tau\, x$$
$$(\lambda A.\ s)[\sigma, \tau] = \lambda A[\sigma].\ s[\Uparrow^{vl}_{tm}\ (\sigma, \tau)]$$
$$(\Lambda.\ s)[\sigma, \tau] = \Lambda.\ s[\Uparrow^{ty}_{tm}\ (\sigma, \tau)]$$

- Traverses values
  - homomorphically
  - mutually recursive
  - with the inferred vector
- Take care of:

Vector Substitutions

Implementation



$$x[\sigma, \tau] = \tau\, x$$

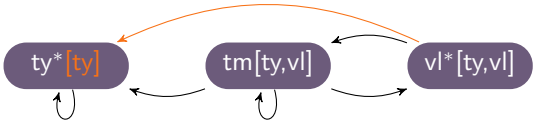
$$(\lambda A.\ s)[\sigma, \tau] = \lambda A[\sigma].\ s[\Uparrow^{vl}_{tm}\ (\sigma, \tau)]$$

$$(\Lambda.\ s)[\sigma, \tau] = \Lambda.\ s[\Uparrow^{ty}_{tm}\ (\sigma, \tau)]$$

- Traverses values
  - homomorphically
  - mutually recursive
  - with the inferred vector
- Take care of:
  - Projections

Vector Substitutions

Implementation

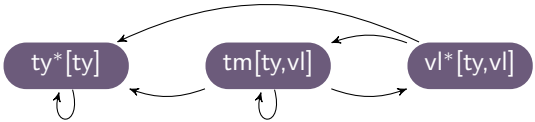


$$x[\sigma,\tau] = \tau\, x$$
$$(\lambda A.\ s)[\sigma,\tau] = \lambda A[\sigma].\ s[\Uparrow^{vl}_{tm}\ (\sigma,\tau)]$$
$$(\Lambda.\ s)[\sigma,\tau] = \Lambda.\ s[\Uparrow^{ty}_{tm}\ (\sigma,\tau)]$$

- Traverses values
  - homomorphically
  - mutually recursive
  - with the inferred vector
- Take care of:
  - Projections
  - Castings

Vector Substitutions

Implementation

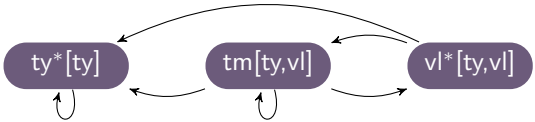


$$x[\sigma, \tau] = \tau\, x$$
$$(\lambda A.\ s)[\sigma, \tau] = \lambda A[\sigma].\ s[\Uparrow^{vl}_{tm}\ (\sigma, \tau)]$$
$$(\Lambda.\ s)[\sigma, \tau] = \Lambda.\ s[\Uparrow^{ty}_{tm}\ (\sigma, \tau)]$$

- Traverses values
  - homomorphically
  - mutually recursive
  - with the inferred vector
- Take care of:
  - Projections
  - Castings
  - Traversals of binders

Vector Substitutions

Implementation



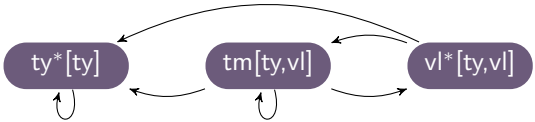
$$x[\sigma,\tau] = \tau\, x$$
$$(\lambda A.\ s)[\sigma,\tau] = \lambda A[\sigma].\ s[\Uparrow^{vl}_{tm}\ (\sigma,\tau)]$$
$$(\Lambda.\ s)[\sigma,\tau] = \Lambda.\ s[\Uparrow^{ty}_{tm}\ (\sigma,\tau)]$$

$$\Uparrow^{vl}_{tm}\ (\sigma,\tau) = (\sigma, 0_{vl} \cdot \tau \circ [\mathsf{id}_{ty}, \uparrow])$$
$$\Uparrow^{ty}_{tm}\ (\sigma,\tau) = (0_{ty} \cdot \sigma \circ \uparrow, \tau \circ [\uparrow, \mathsf{id}_{vl}])$$

- Traverses values
  - homomorphically
  - mutually recursive
  - with the inferred vector
- Take care of:
  - Projections
  - Castings
  - Traversals of binders

Vector Substitutions

Implementation



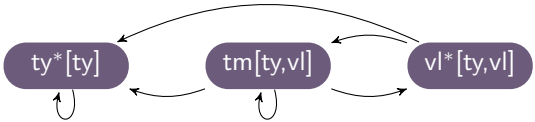
$$x[\sigma,\tau] = \tau\, x$$
$$(\lambda A.\ s)[\sigma,\tau] = \lambda A[\sigma].\ s[\Uparrow^{vl}_{tm}\ (\sigma,\tau)]$$
$$(\Lambda.\ s)[\sigma,\tau] = \Lambda.\ s[\Uparrow^{ty}_{tm}\ (\sigma,\tau)]$$

$$\Uparrow^{vl}_{tm}\ (\sigma,\tau) = (\sigma, 0_{vl} \cdot \tau \circ [\mathsf{id}_{ty}, \uparrow])$$
$$\Uparrow^{ty}_{tm}\ (\sigma,\tau) = (0_{ty} \cdot \sigma \circ \uparrow, \tau \circ [\uparrow, \mathsf{id}_{vl}])$$

- Traverses values
  - homomorphically
  - mutually recursive
  - with the inferred vector
- Take care of:
  - Projections
  - Castings
  - Traversals of binders

Vector Substitutions

Implementation



$$x[\sigma,\tau] = \tau\, x$$
$$(\lambda A.\ s)[\sigma,\tau] = \lambda A[\sigma].\ s[\Uparrow^{vl}_{tm}\ (\sigma,\tau)]$$
$$(\Lambda.\ s)[\sigma,\tau] = \Lambda.\ s[\Uparrow^{ty}_{tm}\ (\sigma,\tau)]$$

$$\Uparrow^{vl}_{tm}\ (\sigma,\tau) = (\sigma, 0_{vl} \cdot \tau \circ [\mathrm{id}_{ty}, \uparrow])$$
$$\Uparrow^{ty}_{tm}\ (\sigma,\tau) = (0_{ty} \cdot \sigma \circ \uparrow, \tau \circ [\uparrow, \mathrm{id}_{vl}])$$

- Traverses values
  - homomorphically
  - mutually recursive
  - with the inferred vector
- Take care of:
  - Projections
  - Castings
  - Traversals of binders

Why vector substitutions?

Why vector substitutions?

Autosubst 1: Separate instantiation, e.g.

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

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

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?

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

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:

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

Case Studies

Case Studies for Autosubst 2

Contents	**Spec**	**Proofs**
POPLMark challenge, part A [Aydemir et al. '05]	151	165
Well-scoped variant of the POPLMark Reloaded Challenge, strong normalization 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 et al. '19]	3950	3750

Restrictions and Future Work

Extensions of the input language:

- To recursive functions [Kaiser et al. '18],
  e.g. logical relations
- To more complex binders, e.g. for patterns:
  Part B of the POPLMark Challenge
- To dependent types [Schäfer et al. '18]:
  Context renaming/context morphism lemmas for typing for free

Wrap-up

Contributions:

- Re-implementation of Autosubst to increase flexibility
- Extension of Autosubst to
  - vector substitutions
  - first-order renamings (in the paper!)
  - well-scoped syntax.
- Several (larger) case studies

Wrap-up

Contributions:

- Re-implementation of Autosubst to increase flexibility
- Extension of Autosubst to
  - vector substitutions
  - first-order renamings (in the paper!)
  - well-scoped syntax.
- Several (larger) case studies

Take-Home Message:

If done right, formalising meta-theory with de Bruijn is easy!

Wrap-up

Contributions:

- Re-implementation of Autosubst to increase flexibility
- Extension of Autosubst to
  - vector substitutions
  - first-order renamings (in the paper!)
  - well-scoped syntax.
- Several (larger) case studies

Take-Home Message:
If done right, formalising meta-theory with de Bruijn is easy!

Available online:

www.ps.uni-saarland.de/extras/autosubst2