Autosubst 2:
Reasoning with de Bruijn Terms and Substitutions

Kathrin Stark
Joint Work With Jonas Kaiser, Steven Schäfer

January 25, CoqPL Workshop 2020, New Orleans

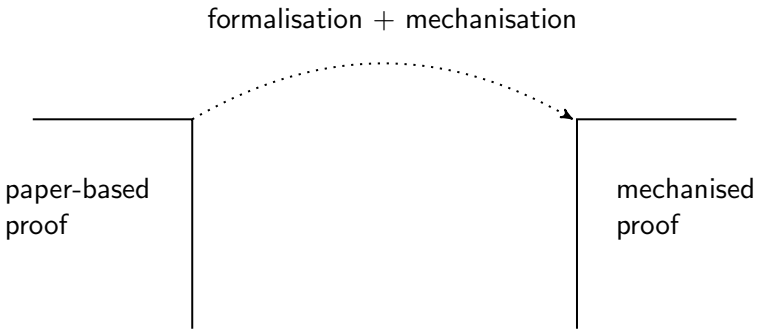
The Mechanisation of Metatheory

- Of programming languages and logical systems with binders,
  - e.g. System F with subtyping from the POPLMark Challenge [Aydemir et al. '05]:

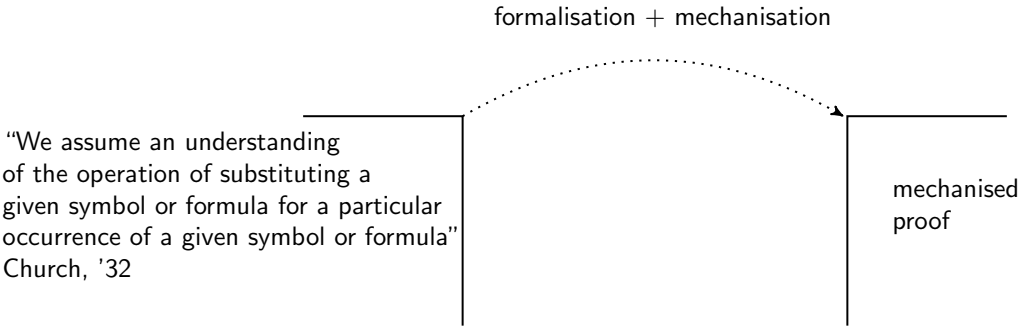
$$
\begin{array}{rcll}
A, B \in ty & ::= & X \mid A \to B \mid \forall X.A & \text{Types} \\
s, t \in tm & ::= & x \mid s\ t \mid s\ A \mid \lambda(x : A).s \mid \Lambda X.s & \text{Terms}
\end{array}
$$

- Mechanising proofs such as
  - type safety
  - weak/strong normalisation

Mechanisations of Meta-Theory with Binders

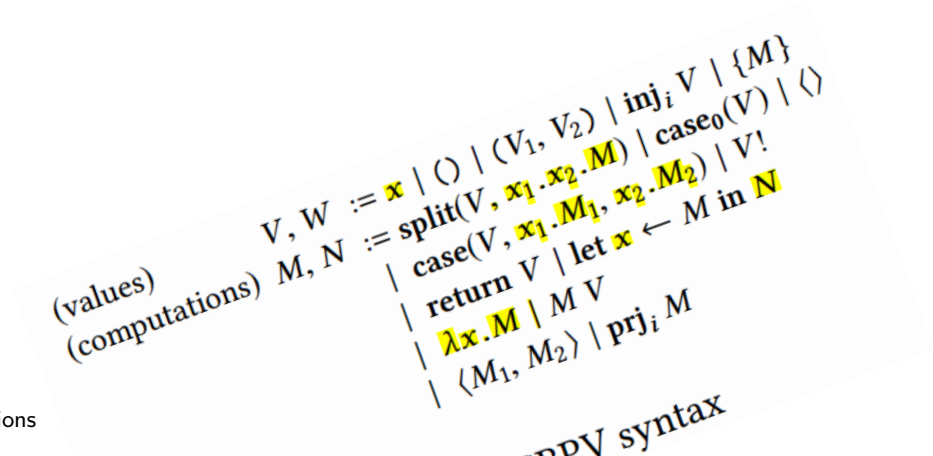


Mechanisations of Meta-Theory with Binders



Three Levels of Binders

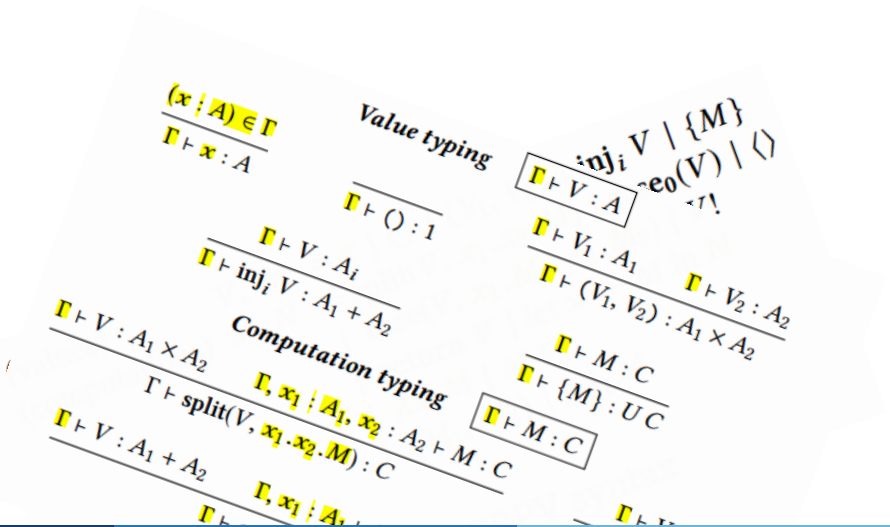
Example: Strong Normalisation for Call-by-Push-Value[Forster et al., '19]



Expressions

Three Levels of Binders

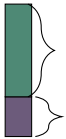
Example: Strong Normalisation for Call-by-Push-Value[Forster et al., '19]



Expressions

Three Levels of Binders

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



Instantiation

Expressions

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

Value typin

Primitive reduction $M \succ M'$

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

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

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

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

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

Three Levels of Binders

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

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

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

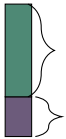
Semantic Typing

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

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

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

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

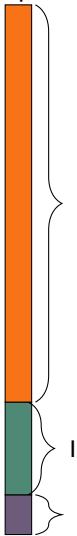


Instantiation

Expressions

Three Levels of Binders

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



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

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

Semanti

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

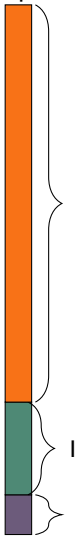
Lemma 4.1. *The following hold:*

1. $C[C] \subseteq \mathcal{E}[C]$.
2. *$C[C]$ only contains normal forms w.r.t. $\rightsquigarrow$.*
3. *If $M \rightsquigarrow^* N$ and $N \in \mathcal{E}[C]$, then $M \in \mathcal{E}[C]$.*

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

Three Levels of Binders

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



Reasoning

Instantiation

Expressions

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

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

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

Lemma 4.1.

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

Three Levels of Binders

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

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

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

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

What Makes a Practical Representation of Binders?

1. **Expressions**
   - We want a notion of α-equivalence of expressions:

$$\lambda x.\lambda z.x\ z \equiv \lambda y.\lambda x.y\ x$$

2. **Instantiation with Substitutions**
   - We want instantiation to be for only free variables and capture-avoiding:

$$(\lambda x.x)[x/t] \neq \lambda x.t$$
$$(\lambda y.x)[x/y] \neq \lambda y.y$$

3. **Reasoning**
   - We want to prove equations between expressions — depending on the representation, this can get difficult.

Guiding Principles

- We want a solution in a general-purpose proof assistant.
- Reasoning on syntax should be generally applicable.
- Boilerplate should be generated.
- All this should be available for a wide range of syntactic systems.

Outline

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]
- ...

KONINKL. NEDERL. AKADEMIE VAN WETENSCHAPPEN — AMSTERDAM
Reprinted from Proceedings, Series A, 75, No. 5 and Indag. Math., 34, No. 5, 1972

MATHEMATICS

LAMBDA CALCULUS NOTATION WITH NAMELESS DUMMIES, A TOOL FOR AUTOMATIC FORMULA MANIPULATION, WITH APPLICATION TO THE CHURCH-ROSSER THEOREM

BY

N. G. DE BRUIJN

(Communicated at the meeting of June 24, 1972)

ABSTRACT

In ordinary lambda calculus the occurrences of a bound variable are made recognizable by the use of one and the same (otherwise irrelevant) name at all occurrences. This convention is known to cause considerable trouble in cases of substitution. In the present paper a different notational system is developed, where occurrences of variables are indicated by integers giving the "distance" to the binding λ instead of a name attached to that λ. The system is claimed to be efficient for automatic formula manipulation as well as for metalingual discussion. As an

Why de Bruijn Syntax?

- Directly implements α-equivalence
- Implementable in a general-purpose type theory
- Is very well understood
- Allows complete reasoning principles
- Can be generalised to a variety of different systems

A Representation of Binders in the Lambda Calculus

1. **Expressions:**
   - De Bruijn indices [de Bruijn '72]
2. **Substitution:**
   - Parallel substitutions, first instantiation with renamings [Adams '04]
   - Primitives of the σ-calculus [Abadi et al. '91]
3. **Reasoning:**
   - Rewriting with the reduction rules of the σ-calculus is complete [Schäfer, Smolka, Tebbi '15]

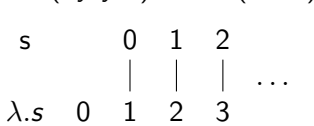
A Representation of Binders in the Lambda Calculus

Part I: Expressions as de Bruijn indices [de Bruijn '72]

Binders are presented by **references**, i.e. represented by natural numbers or a finite type:

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

Binders induce a **scope change**:



A Representation of Binders in the Lambda Calculus

Part II: Instantiation with Substitutions [de Bruijn '72, Abadi et al. '91]

1. **Goal:** Instantiation with substitutions:

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

2. Fix the **set of primitives** of the σ-calculus [Abadi et al. '91]:
   - **Identity**, $\mathsf{id} : \mathbb{N} \to \mathbb{N}$, a renaming defined by $n := n$.
   - **Shifting**, $\uparrow : \mathbb{N} \to \mathbb{N}$, a renaming defined by $\uparrow\ n := 1 + n$.
   - **Extension**, $s \cdot \sigma$, which extends a stream $\sigma : \mathbb{N} \to \mathsf{tm}$ with a new element $s : tm$ at the first position:

$$(s \cdot \sigma)\, 0 := s$$
$$(s \cdot \sigma)\, (1 + n) := \sigma\, n$$

3. Use these primitives to define β-reduction:

$$(\lambda.s)\ t \succ s[t \cdot \mathsf{var}] = s[t..]$$

A Representation of Binders in the Lambda Calculus

Part II: Instantiation with Substitutions [de Bruijn '72, Abadi et al. '91]

Define instantiation with substitutions:

$$x[\sigma] = \sigma\, x$$
$$(s\, t)[\sigma] = (s[\sigma])\,(t[\sigma])$$
$$(\lambda A.\ s)[\sigma] = \lambda A.\ s[\Uparrow \sigma]$$

- Traverses terms
  - homomorphically
- Take care of:
  - Projections
  - Castings
  - Traversals of binders

A Representation of Binders in the Lambda Calculus

Part II: Instantiation with Substitutions [de Bruijn '72, Abadi et al. '91]

Define instantiation with substitutions:

$$x[\sigma] = \sigma\, x$$
$$(s\, t)[\sigma] = (s[\sigma])\,(t[\sigma])$$
$$(\lambda A.\ s)[\sigma] = \lambda A.\ s[\Uparrow \sigma]$$

- Traverses terms
  - homomorphically
- Take care of:
  - Projections
  - Castings
  - Traversals of binders

A Representation of Binders in the Lambda Calculus

Part II: Instantiation with Substitutions [de Bruijn '72, Abadi et al. '91]

Define instantiation with substitutions:

$$x[\sigma] = \sigma\, x$$
$$(s\ t)[\sigma] = (s[\sigma])\,(t[\sigma])$$
$$(\lambda A.\ s)[\sigma] = \lambda A.\ s[\Uparrow \sigma]$$

- Traverses terms
  - homomorphically
- Take care of:
  - Projections
  - Castings
  - Traversals of binders

A Representation of Binders in the Lambda Calculus

Part II: Instantiation with Substitutions [de Bruijn '72, Abadi et al. '91]

Define instantiation with substitutions:

$$x[\sigma] = \sigma\, x$$
$$(s\ t)[\sigma] = (s[\sigma])\,(t[\sigma])$$
$$(\lambda A.\ s)[\sigma] = \lambda A.\ s[\Uparrow \sigma]$$

$$\Uparrow \sigma = 0_{tm} \cdot \sigma \circ [\uparrow]$$

- Traverses terms
  - homomorphically
- Take care of:
  - Projections
  - Castings
  - Traversals of binders

A Representation of Binders in the Lambda Calculus

Part II: Instantiation with Substitutions [de Bruijn '72, Abadi et al. '91]

Define instantiation with substitutions:

$$x[\sigma] = \sigma\, x$$
$$(s\ t)[\sigma] = (s[\sigma])\,(t[\sigma])$$
$$(\lambda A.\ s)[\sigma] = \lambda A.\ s[\Uparrow \sigma]$$

$$\Uparrow \sigma = 0_{tm} \cdot \sigma \circ [\uparrow]$$

- Traverses terms
  - homomorphically
- Take care of:
  - Projections
  - Castings
  - Traversals of binders

A Representation of Binders in the Lambda Calculus

Part II: Instantiation with Substitutions [de Bruijn '72, Abadi et al. '91]

Define instantiation with substitutions:

$$x[\sigma] = \sigma\, x$$
$$(s\ t)[\sigma] = (s[\sigma])\,(t[\sigma])$$
$$(\lambda A.\ s)[\sigma] = \lambda A.\ s[\Uparrow \sigma]$$

$$\Uparrow \sigma = 0_{tm} \cdot \sigma \circ [\uparrow]$$

- Traverses terms
  - homomorphically
- Take care of:
  - Projections
  - Castings
  - Traversals of binders

A Representation of Binders in the Lambda Calculus

Part III: Reasoning

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

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

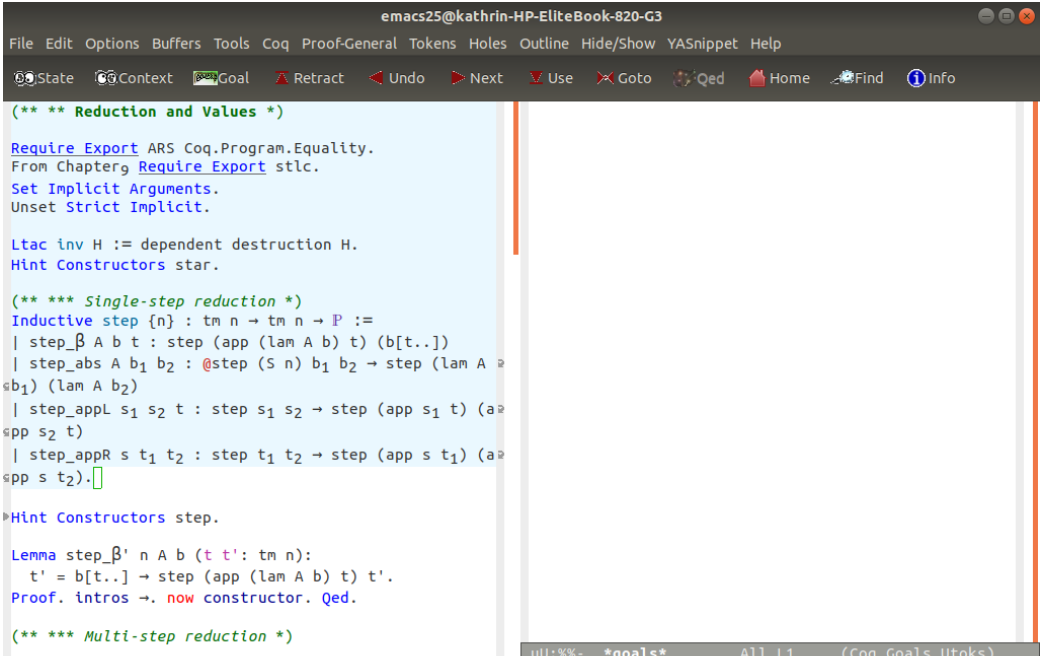
1. Which laws are the right ones?
2. How to prove these laws?
3. How to reason on these laws?

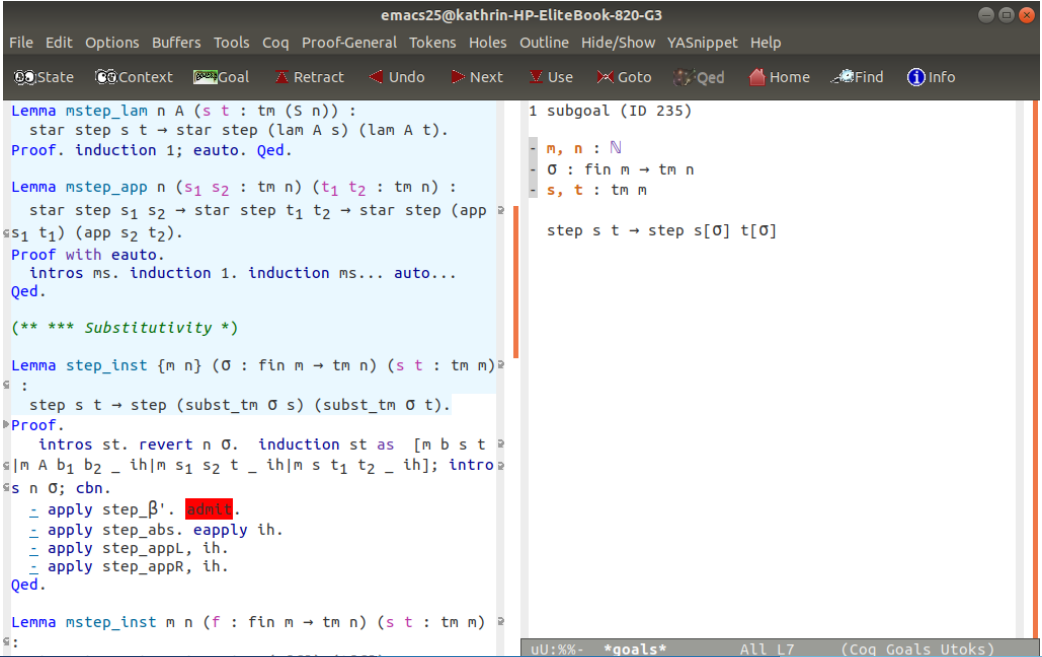
The σ_{SP}-calculus comes with a **confluent**, **terminating** [Curien et al. '96] **rewriting system**, e.g.

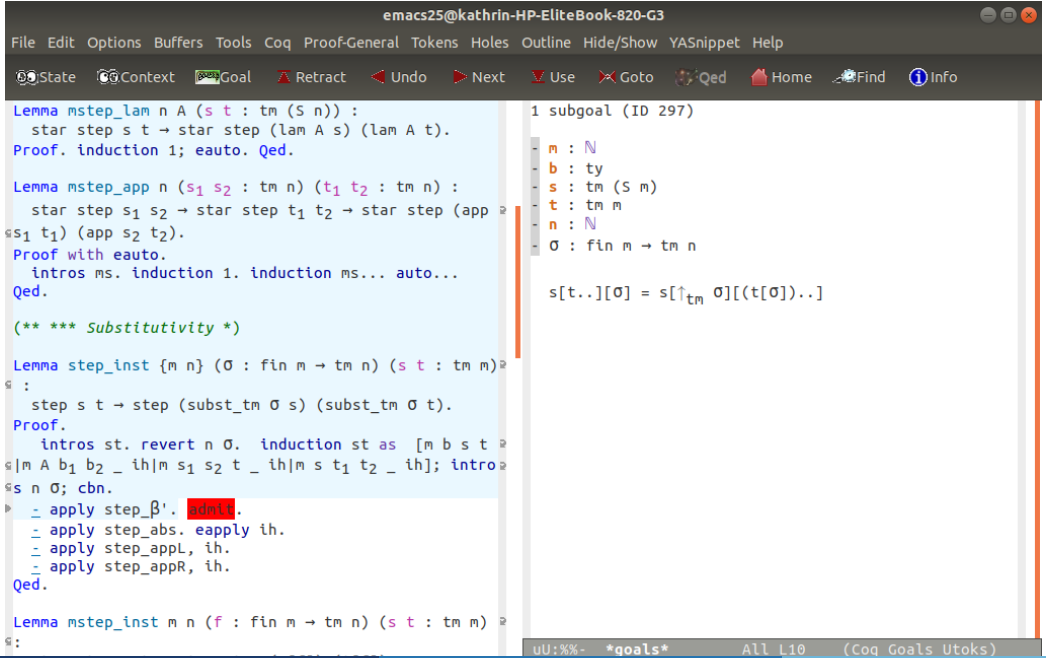
- $0(s \cdot \sigma) = s$
- $s[\mathrm{id}] = s$ and $s[\sigma][\tau] = s[\sigma \circ \_[\tau]]$

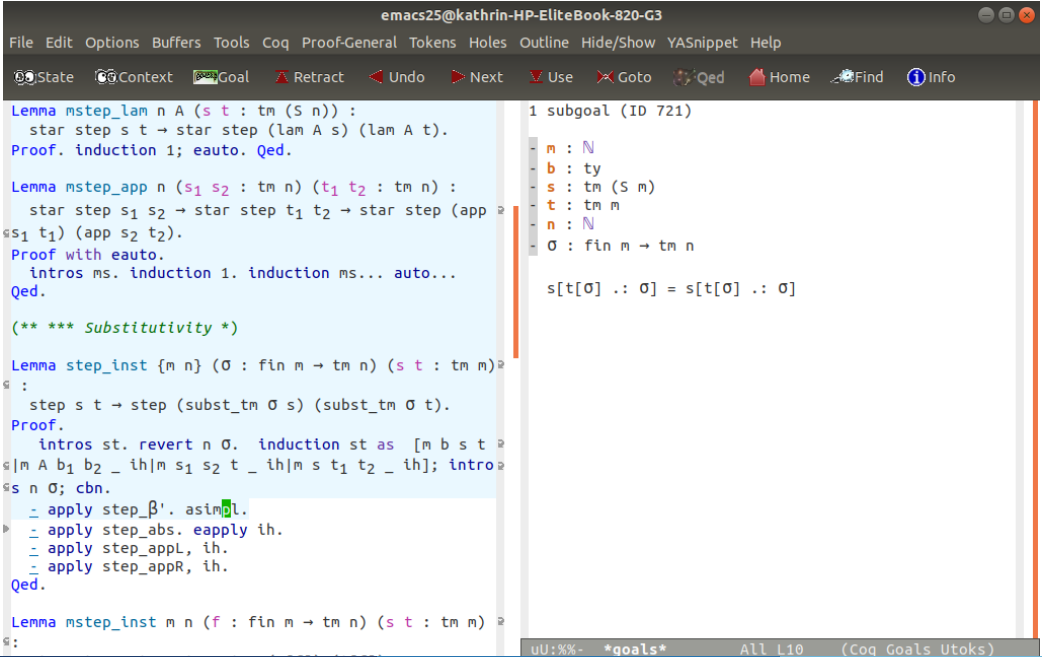
The rewriting system is complete for the de Bruijn algebra of the λ-calculus [Schäfer, Smolka, Tebbi '15].

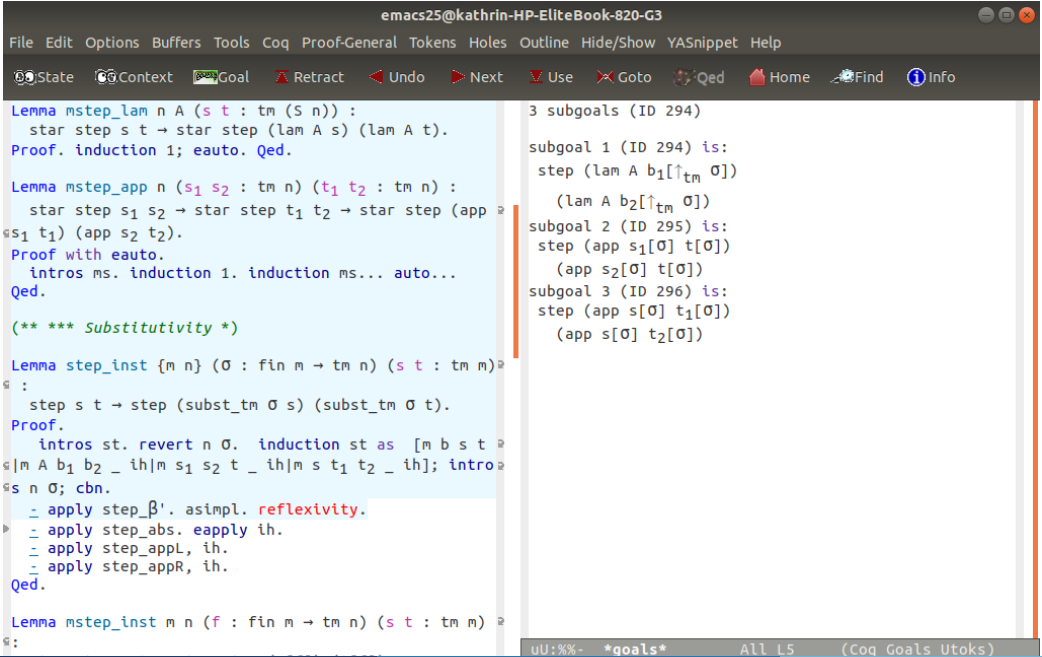
$\Rightarrow$ We can normalise terms with instantiation $\Rightarrow$ Solve equations between expressions


emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
(** ** Reduction and Values *)
Require Export ARS Coq.Program.Equality.
From Chapter9 Require Export stlc.
Set Implicit Arguments.
Unset Strict Implicit.
Ltac inv H := dependent destruction H.
Hint Constructors star.
(** *** Single-step reduction *)
Inductive step {n} : tm n → tm n → ℙ :=
| step_β A b t : step (app (lam A b) t) (b[t..])
| step_abs A b1 b2 : @step (S n) b1 b2 → step (lam A b1) (lam A b2)
| step_appL s1 s2 t : step s1 s2 → step (app s1 t) (app s2 t)
| step_appR s t1 t2 : step t1 t2 → step (app s t1) (app s t2).
Hint Constructors step.
Lemma step_β' n A b (t t': tm n):
t' = b[t..] → step (lam A b) t t'.
Proof. intros →. now constructor. Qed.
(** *** Multi-step reduction *)
U:%%- *goals* All L1 (Coq Goals Utoks)


emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
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 235)
- m, n : ℕ
- σ : fin m → tm n
- s, t : tm m
step s t → step s[σ] t[σ]
uU:%%- *goals* All L7 (Coq Goals Utoks)


emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
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)


emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
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)


emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
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. reflexivity.
- 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) :
3 subgoals (ID 294)
subgoal 1 (ID 294) is:
step (lam A b1[↑tm σ])
(lam A b2[↑tm σ])
subgoal 2 (ID 295) is:
step (app s1[σ] t[σ])
(app s2[σ] t[σ])
subgoal 3 (ID 296) is:
step (app s[σ] t1[σ])
(app s[σ] t2[σ])
uU:%%- *goals* All L5 (Coq Goals Utoks)

Why Parallel Substitutions?

Single-point substitutions: Separate instantiation for each variable, e.g.

$$\_[\_ \mapsto \_] : \mathrm{tm} \to (\mathbb{N} * \mathrm{tm}) \to \mathrm{tm}$$

Problems:

1. Under binders, we need instantiation on all variables with shifting
   $\Rightarrow$ Lifting functions $\uparrow_k^n$, requires interaction laws.
2. How to get an elegant equational theory, e.g. how to commute the different kinds of instantiation?
   $$s[x \mapsto t][y \mapsto u] = s[x \mapsto t][y \mapsto u[x \mapsto t]]?$$
   Depends on whether $x = y$! What is the normal form?

... but does this scale?

Another popular concrete representation is de Bruijn's nameless representation. De Bruijn indices are easy to understand and support the full range of induction principles needed to reason over terms. [..] while the notation clutter is **manageable for "toy" examples** of the size of the simply-typed lambda calculus, we have found it becomes **quite a heavy burden even for fairly small languages like** $F_{<:}$.

[Aydemir et al. '05]

Outline

Why Automation?

F-Ing Modules [Rossberg et al.]

"Our experience [...] was more painful than we had anticipated. Compared to the sample LN developments, ours was different in making use of various forms of derived n-ary (as well as basic unary binders) and in dealing with a larger number of syntactic categories. **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.The **number of required infrastructure lemmas** appears to be **quadratic in the number of variable classes**[...], the number of "substitution" operations needed per class [...] and the arity (unary and n-ary) of binding constructs. So we cannot, hand-on-heart, recommend the vanilla LN style for anything but small, kernel language developments."

Compositionality of Instantiation

Lemma 3.4 (Compositionality).

1. $(\Uparrow^* \xi) \circ (\Uparrow^* \zeta) \equiv \Uparrow^* (\xi \circ \zeta)$
2. $s\langle\xi\rangle\langle\zeta\rangle = s\langle\xi \circ \langle\zeta\rangle\rangle$
3. $(\Uparrow^* \xi) \circ (\Uparrow \tau) \equiv \Uparrow (\xi \circ \tau)$
4. $s\langle\xi\rangle[\tau] = s[\xi \circ \tau]$
5. $(\Uparrow \sigma) \circ (\Uparrow^* \zeta) \equiv \Uparrow (\sigma \circ \langle\zeta\rangle)$
6. $s[\sigma]\langle\zeta\rangle = s[\sigma \circ \langle\zeta\rangle]$
7. $(\Uparrow \sigma) \circ [\Uparrow \tau] \equiv \Uparrow (\sigma \circ [\tau])$
8. $s[\sigma][\tau] \equiv s[\sigma \circ [\tau]]$.

Different Ways to Generalise Syntax

- **Code generation**
  - Ott [Sewell et al., '07]
  - LNgen [Aydemir, Weirich '10]
  - Autosubst 1 [Schäfer, Tebbi, Smolka '15]
- **Universes of Syntax with Binding**
  - Generic Syntax [Allais et al., '18]
  - General Theory of Syntax with Binding [Gheri, Popescu '19]
- **. . . or both?**
  - Needle/Knot [Keuchel et al., '16]

Autosubst 1 is limited in its expressive power.

A High-Level View on the Autosubst Compiler

Specification, e.g.

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

HOAS-like specification,
potentially multivariate and mutual inductive

⇓ Autosubst 2 compiler

Output file
Expressions
+ Instantiation $\_[\_]$
+ Reasoning

unscoped/scoped de Bruijn [de Bruijn '72]
+ parallel substitutions [de Bruijn '72]
+ σ-calculus [Abadi et al. '91]
+ notations and tactics

Generation of Code

HOAS Input

↓ Parsing/Analysis

Dependency Graph

↓ Proof Generation

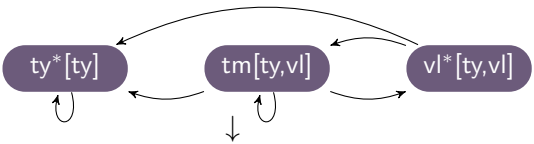
Abstract Proof Terms

↓ Code Generation

Coq Code

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

↓



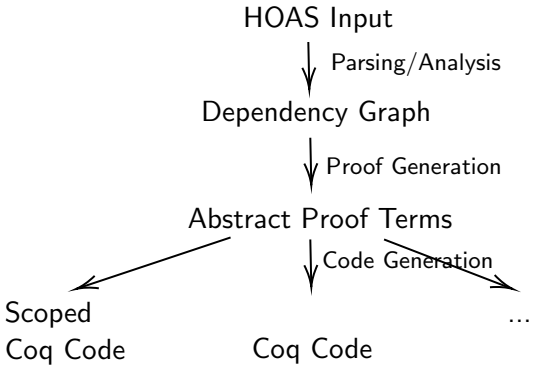
↓

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

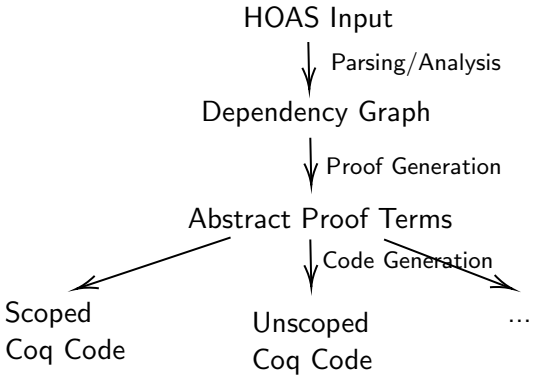
↓

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

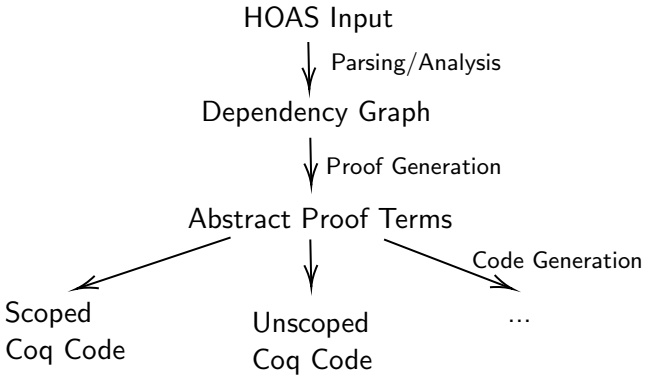
Generation of Code



Generation of Code



Generation of Code



What Syntax does Autosubst 2 Support?

- Unscoped and **scoped** syntax [Bird, Paterson '99]
- **Polyadic** binders, e.g. in a λ-calculus with pairs
- **First-class renamings**, e.g. needed for Kripke-style logical relations
- **External sorts** and sort constructors, e.g. for records
- **Many-sorted** syntax via vector substitutions [Stark, Kaiser, Schäfer '19], e.g. in System F
- **Mutual inductive** syntax, e.g. in Call-by-Push-Value
- **Variadic** syntax, e.g. in the multivariate λ-calculus or for patterns
- Simplified definitions for **first-order** sorts, e.g. for first-order logic
- **Modular** syntax

Principles: Restricted set of substitutions/instantiation covers all relevant indices

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

Example: Call-by-Value System F (F_{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

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:

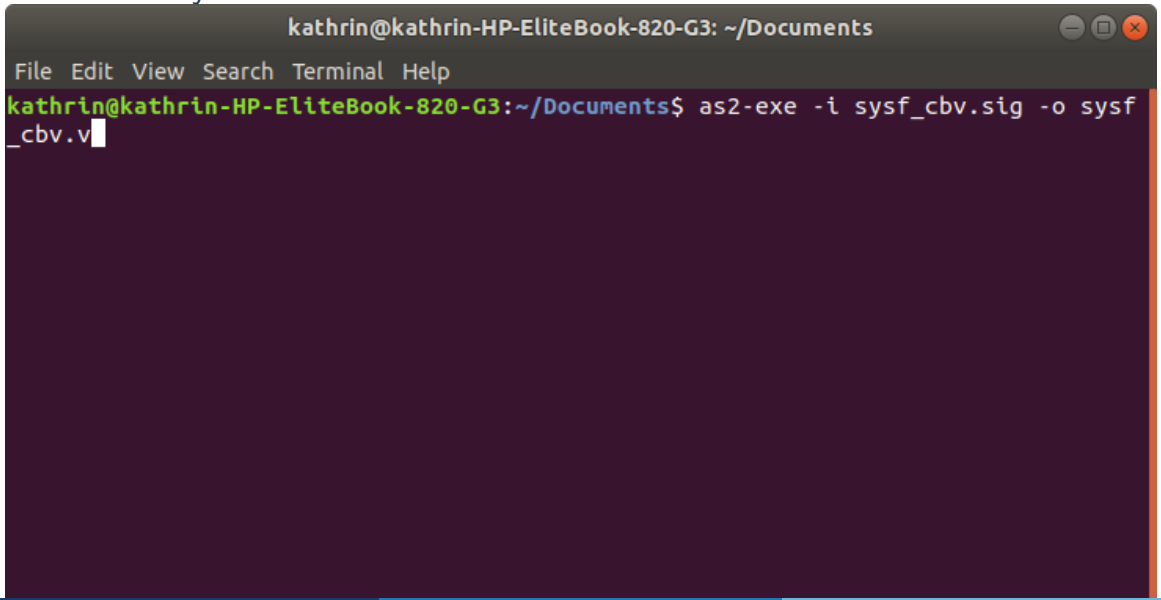
$$\_[\_] : \mathsf{ty} \to (\mathbb{N} \to \mathsf{ty}) \to \mathsf{ty}$$

Demo

Signature for System F in Higher-Order Abstract Syntax

$$
\begin{aligned}
&\texttt{ty} : \texttt{Type} \\
&\texttt{tm} : \texttt{Type} \\
&\texttt{vl} : \texttt{Type} \\
\\
&\texttt{arr} : \texttt{ty} \to \texttt{ty} \to \texttt{ty} \\
&\texttt{all} : (\texttt{ty} \to \texttt{ty}) \to \texttt{ty} \\
\\
&\texttt{app} : \texttt{tm} \to \texttt{tm} \to \texttt{tm} \\
&\texttt{tapp} : \texttt{tm} \to \texttt{ty} \to \texttt{tm} \\
&\texttt{vt} : \texttt{vl} \to \texttt{tm} \\
\\
&\texttt{lam} : \texttt{ty} \to (\texttt{vl} \to \texttt{tm}) \to \texttt{vl} \\
&\texttt{tlam} : (\texttt{ty} \to \texttt{tm}) \to \texttt{vl}
\end{aligned}
$$

Demo File System F



Demo File System F

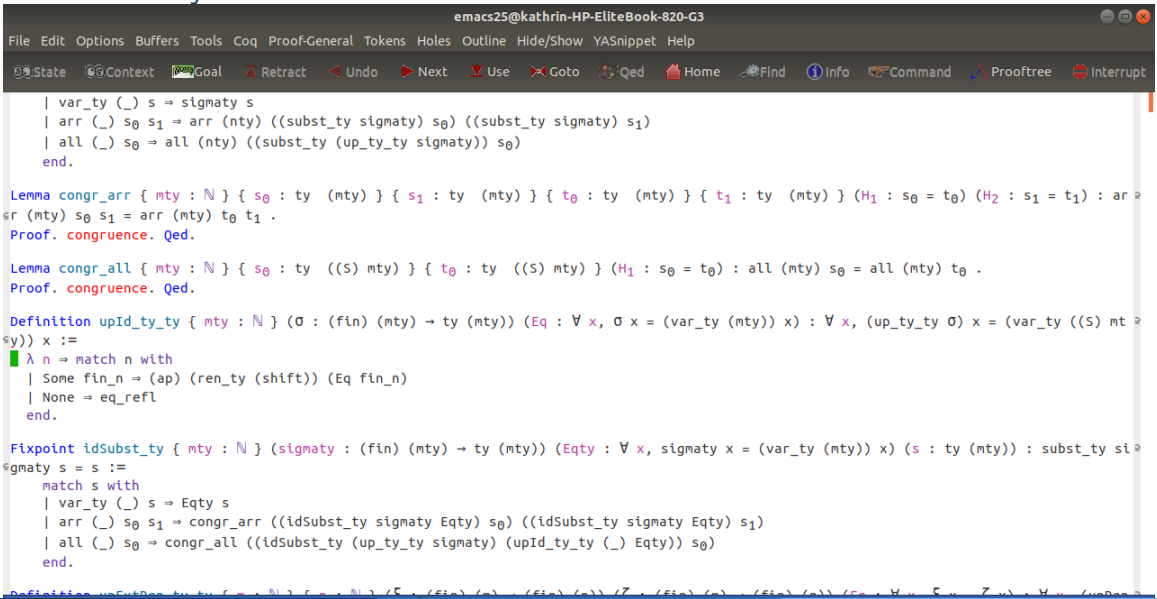
emacs25@kathrin-HP-EliteBook-820-G3

File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help

State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt

```
Require Export fintype. Require Export header_extensible.

Section ty.
Inductive ty (nty : ℕ) : Type :=
  | var_ty : (fin) (nty) → ty (nty)
  | arr : ty  (nty) → ty  (nty) → ty (nty)
  | all : ty  ((S) nty) → ty (nty).

Definition upRen_ty_ty { m : ℕ } { n : ℕ } (ξ : (fin) (m) → (fin) (n)) : (fin) ((S) (m)) → (fin) ((S) (n)) :=
  (up_ren) ξ.

Fixpoint ren_ty { mty : ℕ } { nty : ℕ } (xity : (fin) (mty) → (fin) (nty)) (s : ty (mty)) : ty (nty) :=
    match s with
    | var_ty (_) s ⇒ (var_ty (nty)) (xity s)
    | arr (_) s₀ s₁ ⇒ arr (nty) ((ren_ty xity) s₀) ((ren_ty xity) s₁)
    | all (_) s₀ ⇒ all (nty) ((ren_ty (upRen_ty_ty xity)) s₀)
    end.

Definition up_ty_ty { m : ℕ } { nty : ℕ } (σ : (fin) (m) → ty (nty)) : (fin) ((S) (m)) → ty ((S) nty) :=
  (scons) ((var_ty ((S) nty)) (var_zero)) ((funcomp) (ren_ty (shift)) σ).

Fixpoint subst_ty { mty : ℕ } { nty : ℕ } (sigmaty : (fin) (mty) → ty (nty)) (s : ty (mty)) : ty (nty) :=
    match s with
    | var_ty (_) s ⇒ sigmaty s
    | arr (_) s₀ s₁ ⇒ arr (nty) ((subst_ty sigmaty) s₀) ((subst_ty sigmaty) s₁)
    | all (_) s₀ ⇒ all (nty) ((subst_ty (up_ty_ty sigmaty)) s₀)
    end.
```

Demo File System F



Demo File System F

emacs25@kathrin-HP-EliteBook-820-G3

File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help

State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt

```
Proof. exact (compRenSubst_ty xity tauty (_) (λ n ⇒ eq_refl) s). Qed.

Lemma renComp'_ty { kty : ℕ } { lty : ℕ } { mty : ℕ } (xity : (fin) (mty) → (fin) (kty)) (tauty : (fin) (kty) → ty (lty)) : (funcomp) (subst_ty tauty) (ren_ty xity) = subst_ty ((funcomp) tauty xity) .
Proof. exact ((FunctionalExtensionality.functional_extensionality _ _ ) (λ n ⇒ renComp_ty xity tauty n)). Qed.

Lemma renRen_ty { kty : ℕ } { lty : ℕ } { mty : ℕ } (xity : (fin) (mty) → (fin) (kty)) (zetaty : (fin) (kty) → (fin) (lty)) (s : ty (mty)) : ren_ty zetaty (ren_ty xity s) = ren_ty ((funcomp) zetaty xity) s .
Proof. exact (compRenRen_ty xity zetaty (_) (λ n ⇒ eq_refl) s). Qed.

Lemma renRen'_ty { kty : ℕ } { lty : ℕ } { mty : ℕ } (xity : (fin) (mty) → (fin) (kty)) (zetaty : (fin) (kty) → (fin) (lty)) : (funcomp) (ren_ty zetaty) (ren_ty xity) = ren_ty ((funcomp) zetaty xity) .
Proof. exact ((FunctionalExtensionality.functional_extensionality _ _ ) (λ n ⇒ renRen_ty xity zetaty n)). Qed.

End ty.

Section tmvl.
Inductive tm (nty nvl : ℕ) : Type :=
  | app : tm  (nty) (nvl) → tm  (nty) (nvl) → tm (nty) (nvl)
  | tapp : tm  (nty) (nvl) → ty  (nty) → tm (nty) (nvl)
  | vt : vl  (nty) (nvl) → tm (nty) (nvl)
 with vl (nty nvl : ℕ) : Type :=
  | var_vl : (fin) (nvl) → vl (nty) (nvl)
  | lam : ty  (nty) → tm  (nty) ((S) nvl) → vl (nty) (nvl)
  | tlam : tm  ((S) nty) (nvl) → vl (nty) (nvl).

Lemma congr_app { mty mvl : ℕ } { s₀ : tm  (mty) (mvl) } { s₁ : tm  (mty) (mvl) } { t₀ : tm  (mty) (mvl) } { t₁ : tm  (mty) (mvl) } (H₁ : s₀ = t₀) (H₂ : s₁ = t₁) : app (mty) (mvl) s₀ s₁ = app (mty) (mvl) t₀ t₁ .
```

And in Practice?

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 et al. '19]	3950	3750
Modular development of preservation/weak head normalisation/strong normalisation for a modular λ-calculus with boolean and arithmetic expressions[Forster and S '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, S '19]

Syntax

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

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

Outline

POPLmark 15 Year Retrospective Panel

About Program Proceedings

This panel will bring together experts in mechanized metatheory with the broader POPL community for a discussion of the POPLmark challenge 15 years later, with an eye toward the future. It will discuss lessons from POPLmark and the evolution of mechanized metatheory since POPLmark as relevant to the field of mechanized metatheory, to the design of future benchmark suites and challenges, and to the POPL community more broadly.

Format

- 10 minute background presentation (Benjamin Pierce)
- 70 minutes Q&A (planned and audience questions)
- 10 minutes conclusion

Content

Here are just a few examples of what we hope to discuss:

- What did we learn about different proof assistants and different binding styles from the challenge itself?
- What happened in the years immediately following the POPLmark challenge?
- What about POPLmark led to its impact? What can we learn from that in designing future benchmark suites for mechanized metatheory, and for programming languages in general?
- What has changed since 2005, and what new challenges has this brought with it?
- What problems raised in POPLmark were underaddressed? How can we address them?

You can send us your own questions in advance using this form, and you can ask questions in person during the second half of the Q&A.

Panel

Benjamin C. Pierce — Panelist
University of Pennsylvania
United States

Peter Sewell — Panelist
University of Cambridge

Xavier Leroy — Panelist
Collège de France
France

Robby Findler — Panelist
Northwestern University, USA
United States

Scott Owens — Panelist
University of Kent, UK
United Kingdom

Brigitte Pientka — Panelist
McGill University

Modular Syntax

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."*

A Practical Approach to Modular Syntax [Forster, S '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 – 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

Variadic Syntax

Has the Full POPLMark Challenge Reached the Masses Yet?[1]

POPLMark Part A

Summary of the encoding techniques and tools used by the available submissions:

	Alpha Prolog	Coq	Twelf	ATS	Isabelle/HOL	Matita	Abella
de Bruijn		Vouillon, Charguéraud (a)			Berghofer		
HOAS			CMU				Gacek
Weak HOAS		Ciaffaglione and Scagnetto					
Hybrid				Xi			
Locally nameless		Chlipala, Leroy, Charguéraud (b)				Ricciotti	
Named variables		Stump					
Nested abstract syntax		Hirschowitz and Maggesi					
Nominal	Fairbairn				Urban et al.		

[1] Source: https://www.seas.upenn.edu/ plclub/poplmark/

Has the Full POPLMark Challenge Reached the Masses Yet?[1]

POPLMark Part B

Summary of the encoding techniques and tools used by the available submissions:

	Alpha Prolog	Coq	Twelf	ATS	Isabelle/HOL	Matita	Abella
de Bruijn		Vouillon, [redacted]			Berghofer		
HOAS			CMU				[redacted]
Weak HOAS		[redacted]					
Hybrid				[redacted]			
Locally nameless		[redacted]				[redacted]	
Named variables		[redacted]					
Nested abstract syntax		[redacted]					
Nominal	[redacted]				[redacted]		

[1] Source: https://www.seas.upenn.edu/ plclub/poplmark/

Variadic Syntax

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

$$s, t \in tm_k ::= \mathsf{var}\, x \mid \mathsf{app}\, s^k\, \{t_1^k .. t_n^k\} \mid \lambda_n.s^{n+k} \qquad x \in \mathsf{fin\,k}$$

$$\begin{array}{lccccccc} \mathsf{s} & & .. & 0 & & 1 & & 2 \\ & & & \vert & & \vert & & \vert \quad \ldots \\ \lambda_n.s & 0 & .. & \mathsf{n} & 1+\mathsf{n} & & 2+\mathsf{n} & \end{array}$$

- Other examples: Pattern matching, recursive let-bindings

From Monadic to Variadic Primitives

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

$+$ definition of instantiation $+$ adaption of reasoning principles

Showing Preservation With Parallel Substitutions

Goal: If $\Gamma \vdash s : A$ and $s \succ t$, then $\Gamma \vdash t : A$.

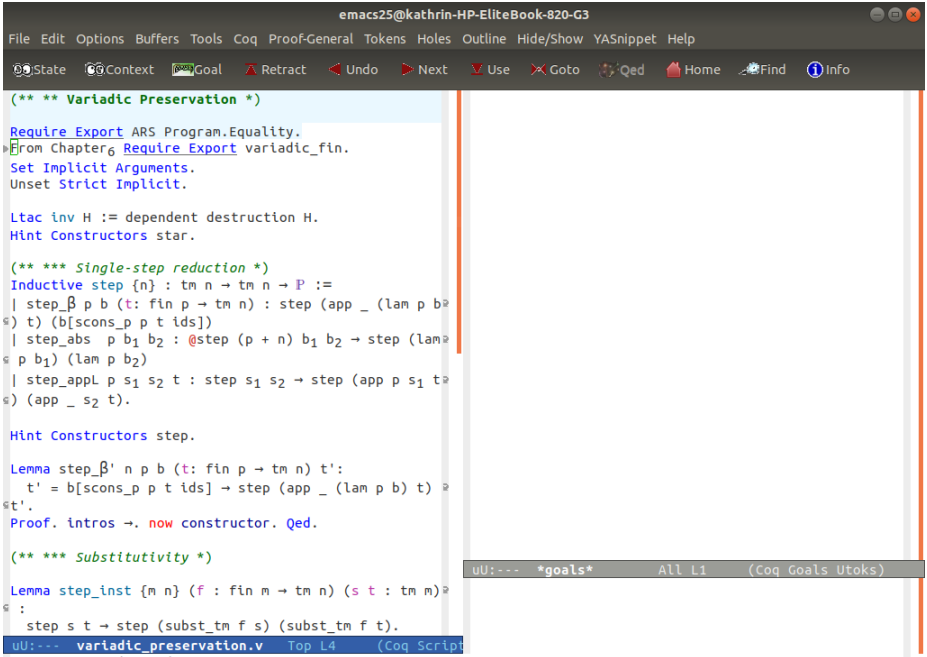
- Show "substitutivity of typing" with **context renaming** and **context morphism** lemmas [Goguen, McKinna '97]:

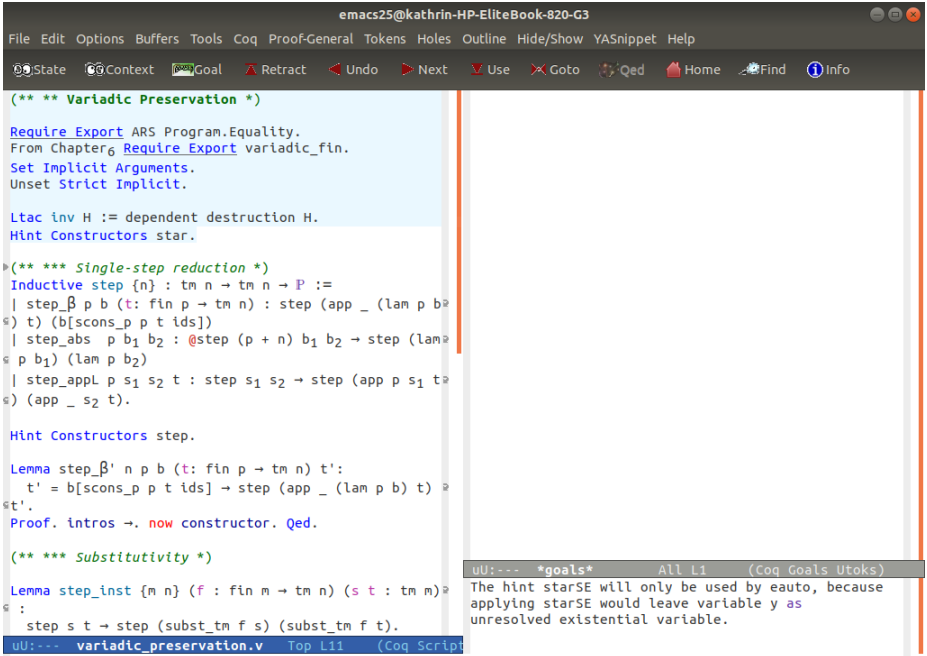
$$\frac{\Gamma \vdash s : A \qquad \forall x. \Gamma\, x = \Delta\, (\xi\, x)}{\Delta \vdash s\langle \xi \rangle : A}$$

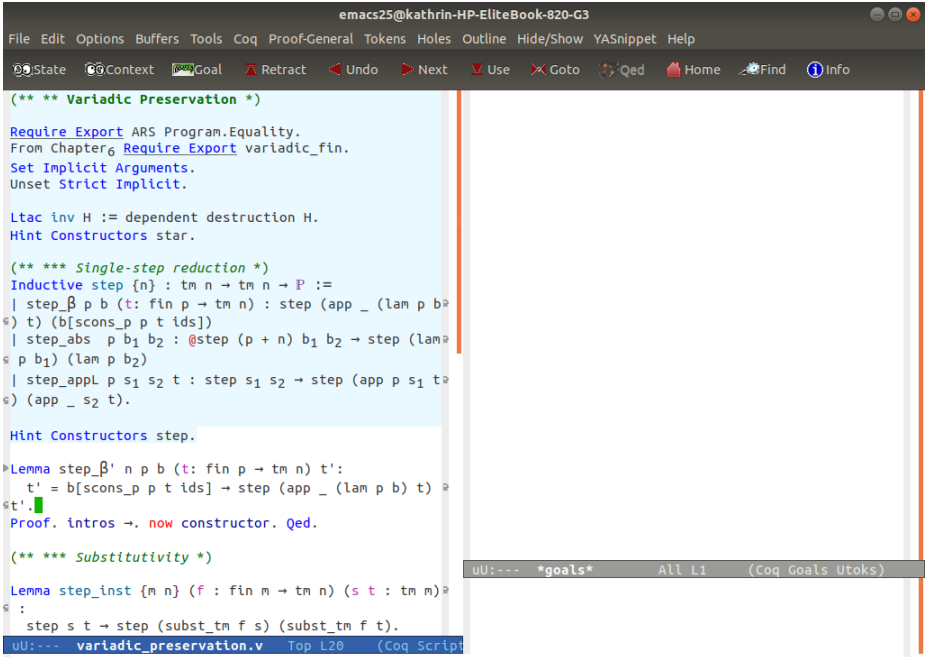
$$\frac{\Gamma \vdash s : A \qquad \forall x. \Delta \vdash \sigma\, x : \Gamma\, x}{\Delta \vdash s[\sigma] : A}$$

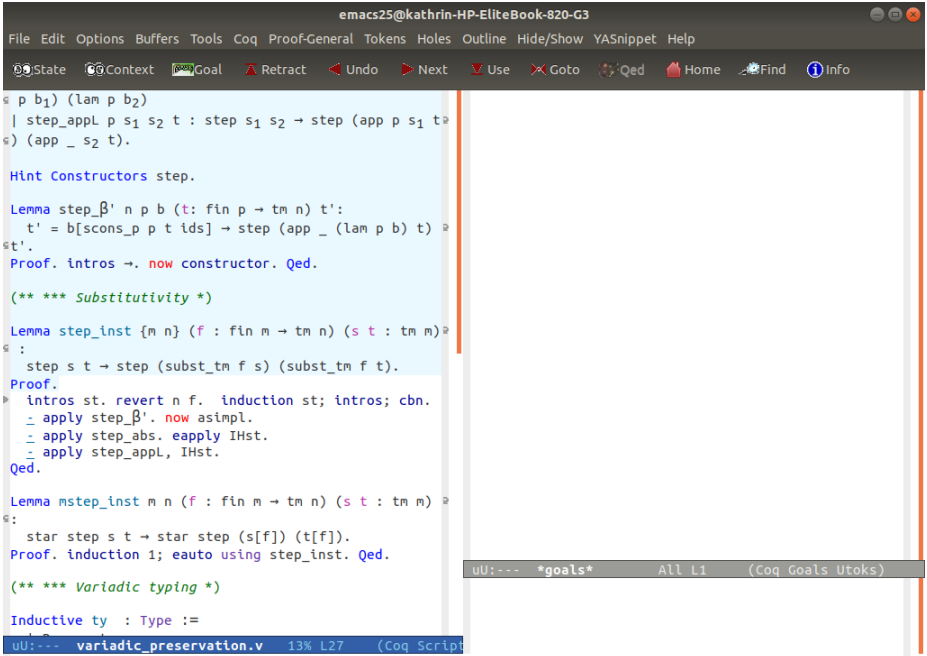
- Show preservation.

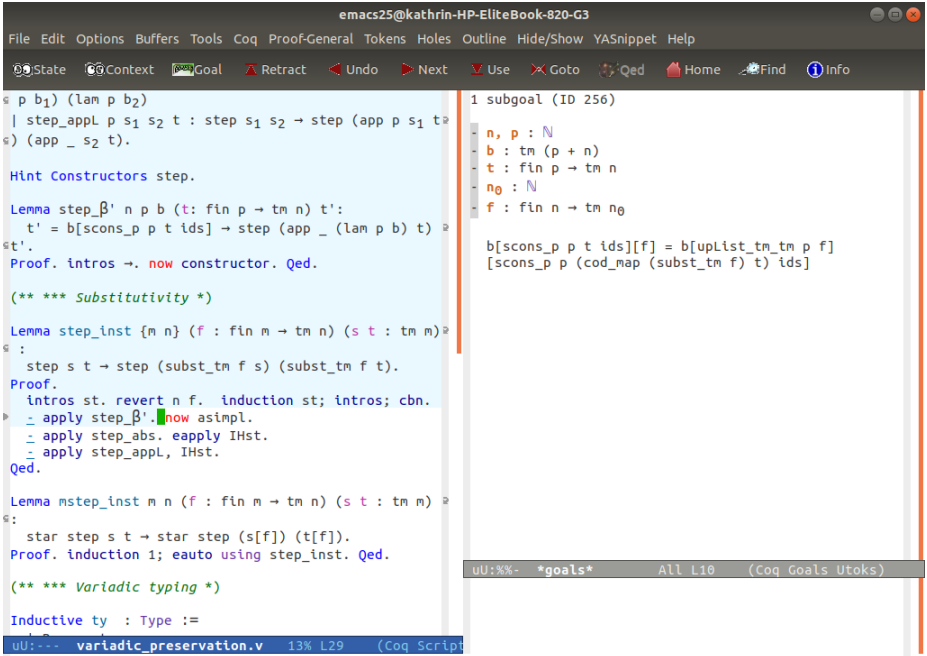
Demo: Type Safety for the Variadic λ-Calculus

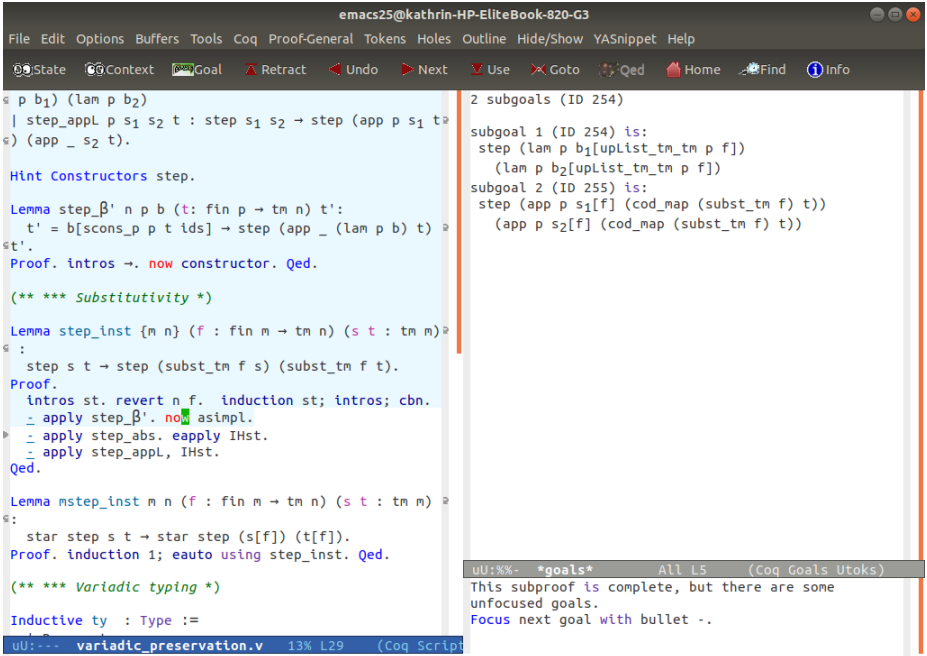

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
(** ** Variadic Preservation *)
Require Export ARS Program.Equality.
From Chapter6 Require Export variadic_fin.
Set Implicit Arguments.
Unset Strict Implicit.
Ltac inv H := dependent destruction H.
Hint Constructors star.
(** *** Single-step reduction *)
Inductive step {n} : tm n → tm n → Prop :=
| step_β p b (t: fin p → tm n) : step (app _ (lam p b) t) (b[scons_p p t ids])
| step_abs p b1 b2 : @step (p + n) b1 b2 → step (lam p b1) (lam p b2)
| step_appL p s1 s2 t : step s1 s2 → step (app p s1 t) (app _ s2 t).
Hint Constructors step.
Lemma step_β' n p b (t: fin p → tm n) t':
t' = b[scons_p p t ids] → step (app _ (lam p b) t) t'.
Proof. intros →. now constructor. Qed.
(** *** Substitutivity *)
Lemma step_inst {m n} (f : fin m → tm n) (s t : tm m) :
step s t → step (subst_tm f s) (subst_tm f t).
uU:--- variadic_preservation.v Top L4 (Coq Script
uU:--- *goals* All L1 (Coq Goals Utoks)

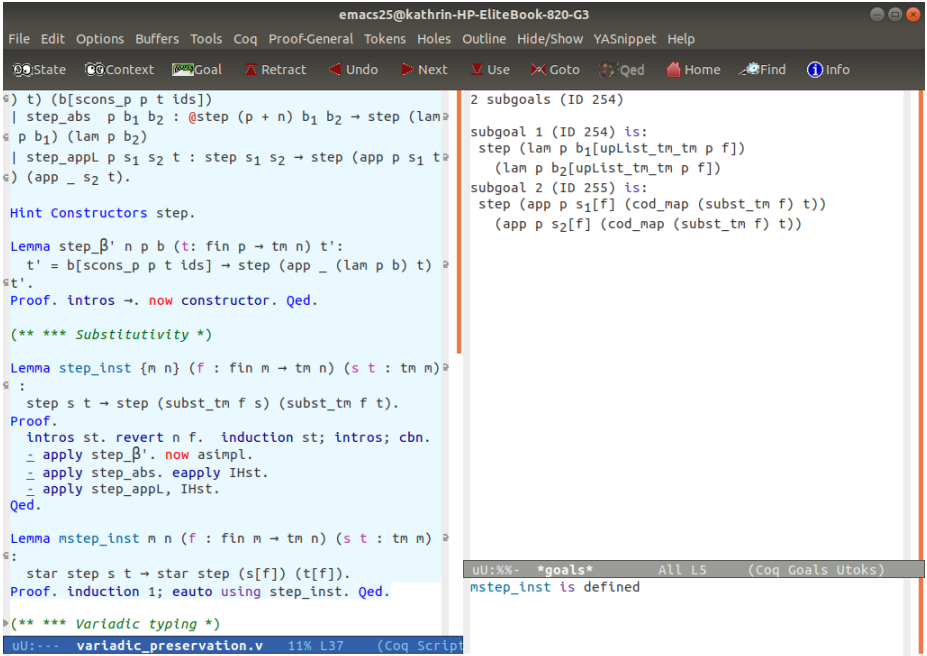

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
(** ** Variadic Preservation *)
Require Export ARS Program.Equality.
From Chapter6 Require Export variadic_fin.
Set Implicit Arguments.
Unset Strict Implicit.
Ltac inv H := dependent destruction H.
Hint Constructors star.
(** *** Single-step reduction *)
Inductive step {n} : tm n → tm n → Prop :=
| step_β p b (t: fin p → tm n) : step (app _ (lam p b) t) (b[scons_p p t ids])
| step_abs p b1 b2 : @step (p + n) b1 b2 → step (lam p b1) (lam p b2)
| step_appL p s1 s2 t : step s1 s2 → step (app p s1 t) (app _ s2 t).
Hint Constructors step.
Lemma step_β' n p b (t: fin p → tm n) t':
t' = b[scons_p p t ids] → step (app _ (lam p b) t) t'.
Proof. intros →. now constructor. Qed.
(** *** Substitutivity *)
Lemma step_inst {m n} (f : fin m → tm n) (s t : tm m) :
step s t → step (subst_tm f s) (subst_tm f t).
uU:--- variadic_preservation.v Top L11 (Coq Script
uU:--- *goals* All L1 (Coq Goals Utoks)
The hint starSE will only be used by eauto, because
applying starSE would leave variable y as
unresolved existential variable.

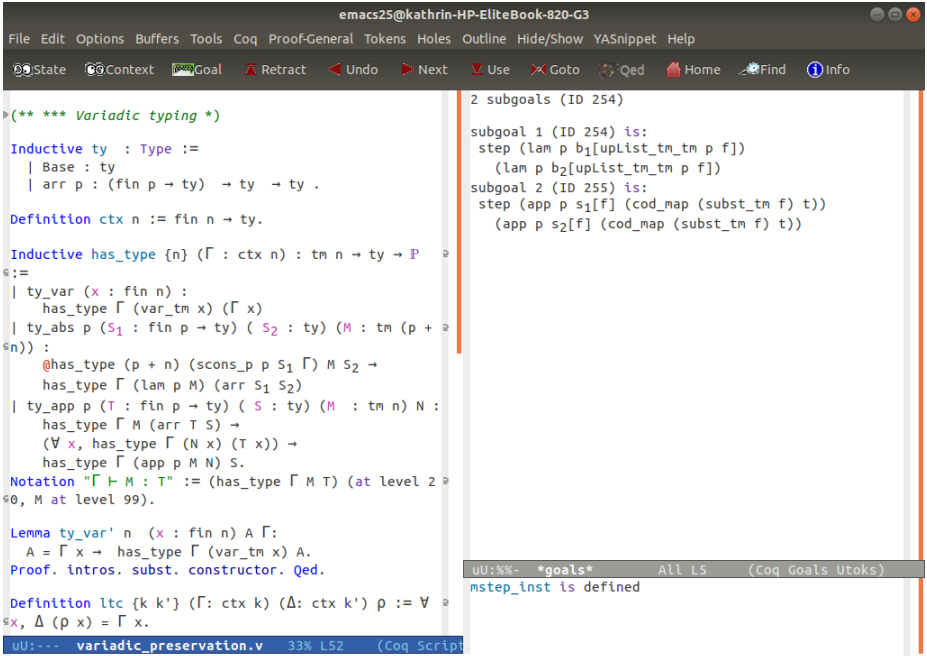

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
(** ** Variadic Preservation *)
Require Export ARS Program.Equality.
From Chapter6 Require Export variadic_fin.
Set Implicit Arguments.
Unset Strict Implicit.
Ltac inv H := dependent destruction H.
Hint Constructors star.
(** *** Single-step reduction *)
Inductive step {n} : tm n → tm n → Prop :=
| step_β p b (t: fin p → tm n) : step (app _ (lam p b) t) (b[scons_p p t ids])
| step_abs p b1 b2 : @step (p + n) b1 b2 → step (lam p b1) (lam p b2)
| step_appL p s1 s2 t : step s1 s2 → step (app p s1 t) (app _ s2 t).
Hint Constructors step.
Lemma step_β' n p b (t: fin p → tm n) t':
t' = b[scons_p p t ids] → step (app _ (lam p b) t) t'.
Proof. intros →. now constructor. Qed.
(** *** Substitutivity *)
Lemma step_inst {m n} (f : fin m → tm n) (s t : tm m) :
step s t → step (subst_tm f s) (subst_tm f t).
uU:--- variadic_preservation.v Top L20 (Coq Script
uU:--- *goals* All L1 (Coq Goals Utoks)

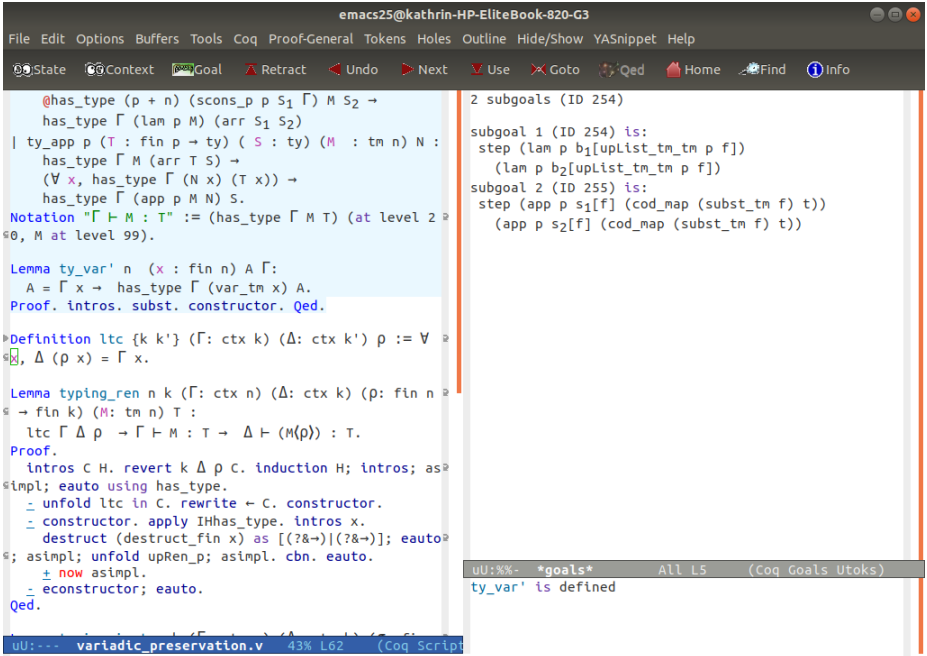

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
p b1) (lam p b2)
| step_appL p s1 s2 t : step s1 s2 → step (app p s1 t
) (app _ s2 t).
Hint Constructors step.
Lemma step_β' n p b (t: fin p → tm n) t':
t' = b[scons_p p t ids] → step (app _ (lam p b) t)
t'.
Proof. intros →. now constructor. Qed.
(** *** Substitutivity *)
Lemma step_inst {m n} (f : fin m → tm n) (s t : tm m)
:
step s t → step (subst_tm f s) (subst_tm f t).
Proof.
intros st. revert n f. induction st; intros; cbn.
- apply step_β'. now asimpl.
- apply step_abs. eapply IHst.
- apply step_appL, IHst.
Qed.
Lemma mstep_inst m n (f : fin m → tm n) (s t : tm m)
:
star step s t → star step (s[f]) (t[f]).
Proof. induction 1; eauto using step_inst. Qed.
(** *** Variadic typing *)
Inductive ty : Type :=
uU:--- variadic_preservation.v 13% L27 (Coq Script
uU:--- *goals* All L1 (Coq Goals Utoks)

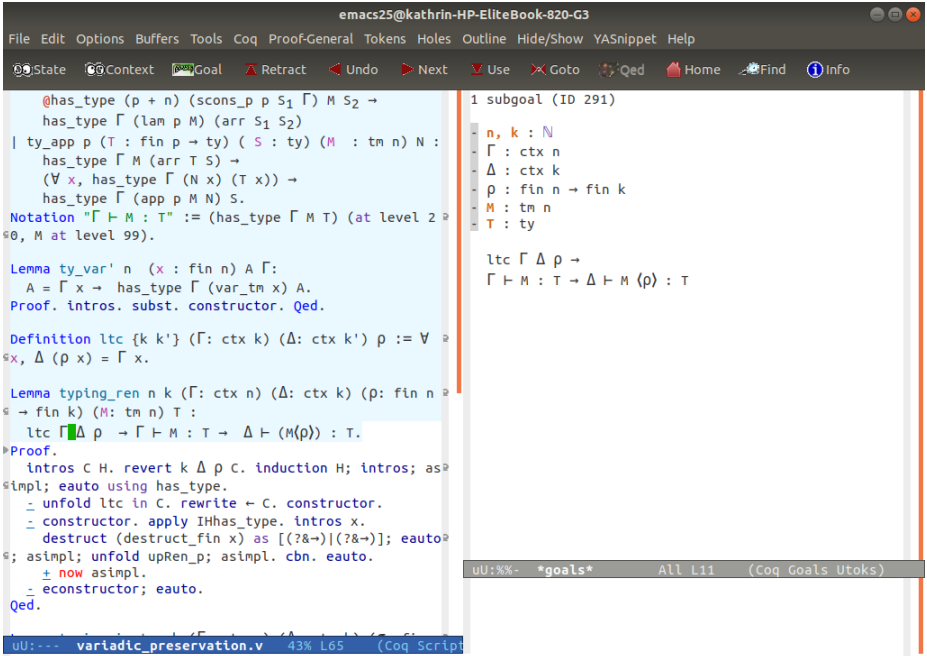

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
p b1) (lam p b2)
| step_appL p s1 s2 t : step s1 s2 → step (app p s1 t
) (app _ s2 t).
Hint Constructors step.
Lemma step_β' n p b (t: fin p → tm n) t':
t' = b[scons_p p t ids] → step (app _ (lam p b) t)
t'.
Proof. intros →. now constructor. Qed.
(** *** Substitutivity *)
Lemma step_inst {m n} (f : fin m → tm n) (s t : tm m)
:
step s t → step (subst_tm f s) (subst_tm f t).
Proof.
intros st. revert n f. induction st; intros; cbn.
- apply step_β'. now asimpl.
- apply step_abs. eapply IHst.
- apply step_appL, IHst.
Qed.
Lemma mstep_inst m n (f : fin m → tm n) (s t : tm m)
:
star step s t → star step (s[f]) (t[f]).
Proof. induction 1; eauto using step_inst. Qed.
(** *** Variadic typing *)
Inductive ty : Type :=
uU:--- variadic_preservation.v 13% L29 (Coq Script
1 subgoal (ID 256)
n, p : ℕ
b : tm (p + n)
t : fin p → tm n
n0 : ℕ
f : fin n → tm n0
b[scons_p p t ids][f] = b[upList_tm_tm p f]
[scons_p p (cod_map (subst_tm f) t) ids]
uU:%%- *goals* All L10 (Coq Goals Utoks)

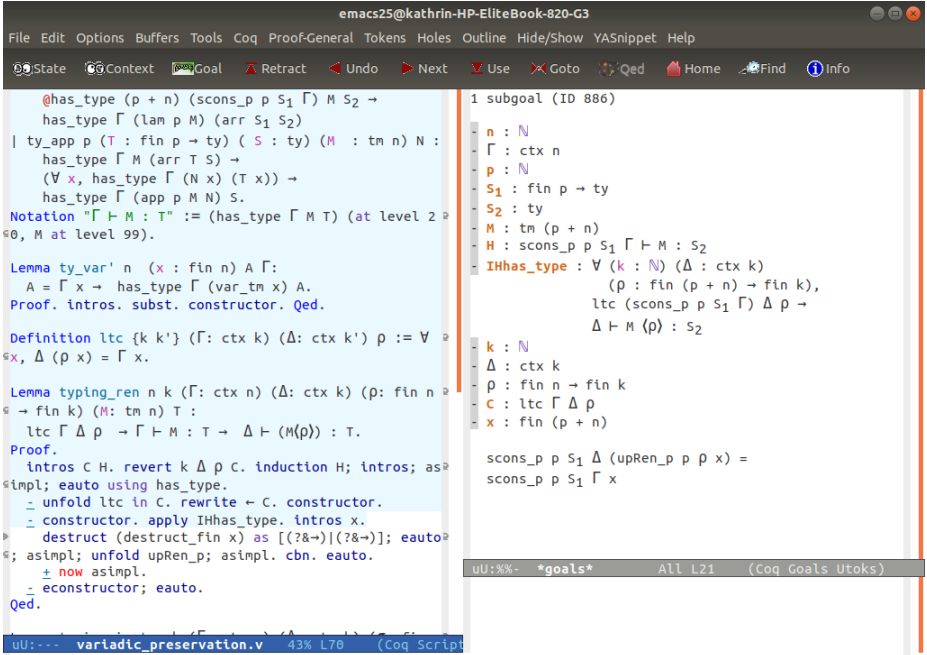

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
p b1) (lam p b2)
| step_appL p s1 s2 t : step s1 s2 → step (app p s1 t
) (app _ s2 t).
Hint Constructors step.
Lemma step_β' n p b (t: fin p → tm n) t':
t' = b[scons_p p t ids] → step (app _ (lam p b) t)
t'.
Proof. intros →. now constructor. Qed.
(** *** Substitutivity *)
Lemma step_inst {m n} (f : fin m → tm n) (s t : tm m)
:
step s t → step (subst_tm f s) (subst_tm f t).
Proof.
intros st. revert n f. induction st; intros; cbn.
- apply step_β'. now asimpl.
- apply step_abs. eapply IHst.
- apply step_appL, IHst.
Qed.
Lemma mstep_inst m n (f : fin m → tm n) (s t : tm m)
:
star step s t → star step (s[f]) (t[f]).
Proof. induction 1; eauto using step_inst. Qed.
(** *** Variadic typing *)
Inductive ty : Type :=
uU:--- variadic_preservation.v 13% L29 (Coq Script
2 subgoals (ID 254)
subgoal 1 (ID 254) is:
step (lam p b1[upList_tm_tm p f])
(lam p b2[upList_tm_tm p f])
subgoal 2 (ID 255) is:
step (app p s1[f] (cod_map (subst_tm f) t))
(app p s2[f] (cod_map (subst_tm f) t))
uU:%%- *goals* All L5 (Coq Goals Utoks)
This subproof is complete, but there are some
unfocused goals.
Focus next goal with bullet -.

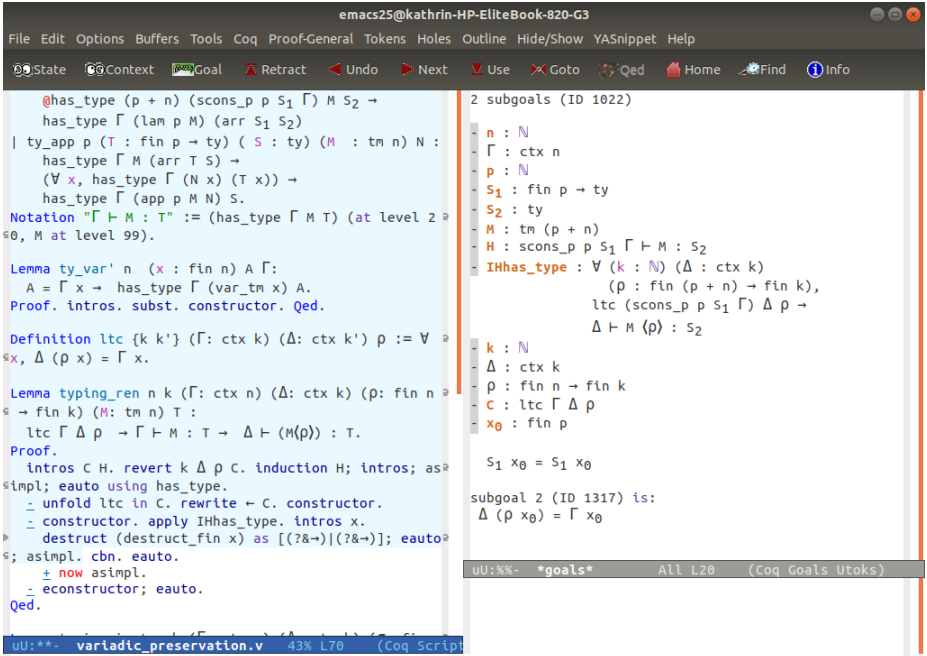

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
) t) (b[scons_p p t ids])
| step_abs p b1 b2 : @step (p + n) b1 b2 → step (lam
p b1) (lam p b2)
| step_appL p s1 s2 t : step s1 s2 → step (app p s1 t
) (app _ s2 t).
Hint Constructors step.
Lemma step_β' n p b (t: fin p → tm n) t':
t' = b[scons_p p t ids] → step (app _ (lam p b) t)
t'.
Proof. intros →. now constructor. Qed.
(** *** Substitutivity *)
Lemma step_inst {m n} (f : fin m → tm n) (s t : tm m)
:
step s t → step (subst_tm f s) (subst_tm f t).
Proof.
intros st. revert n f. induction st; intros; cbn.
- apply step_β'. now asimpl.
- apply step_abs. eapply IHst.
- apply step_appL, IHst.
Qed.
Lemma mstep_inst m n (f : fin m → tm n) (s t : tm m)
:
star step s t → star step (s[f]) (t[f]).
Proof. induction 1; eauto using step_inst. Qed.
(** *** Variadic typing *)
uU:--- variadic_preservation.v 11% L37 (Coq Script
2 subgoals (ID 254)
subgoal 1 (ID 254) is:
step (lam p b1[upList_tm_tm p f])
(lam p b2[upList_tm_tm p f])
subgoal 2 (ID 255) is:
step (app p s1[f] (cod_map (subst_tm f) t))
(app p s2[f] (cod_map (subst_tm f) t))
uU:%%- *goals* All L5 (Coq Goals Utoks)
mstep_inst is defined

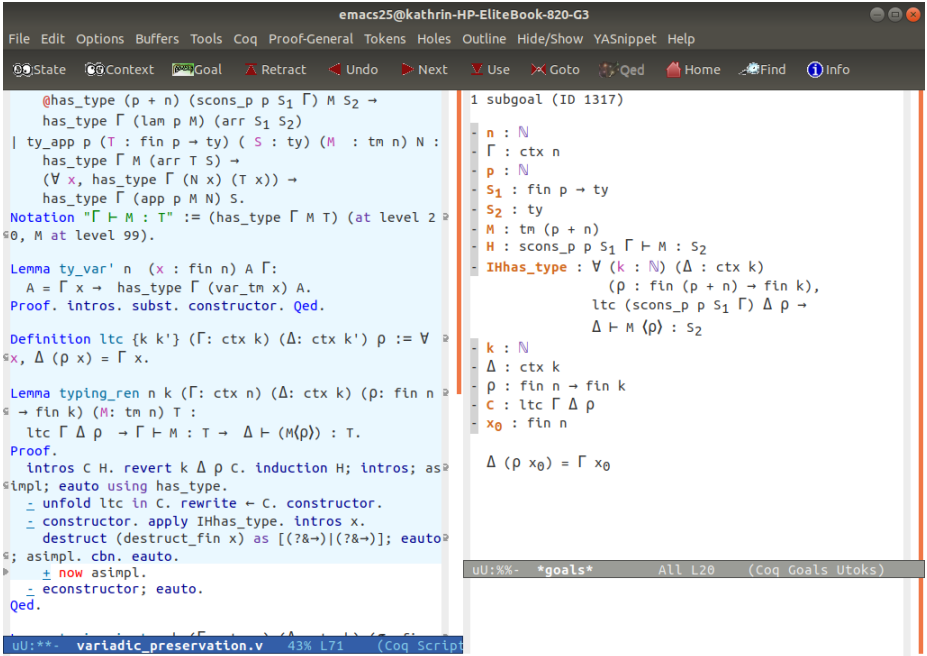

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
(** *** Variadic typing *)
Inductive ty : Type :=
| Base : ty
| arr p : (fin p → ty) → ty → ty .
Definition ctx n := fin n → ty.
Inductive has_type {n} (Γ : ctx n) : tm n → ty → ℙ :=
| ty_var (x : fin n) :
has_type Γ (var_tm x) (Γ x)
| ty_abs p (S1 : fin p → ty) (S2 : ty) (M : tm (p + n)) :
@has_type (p + n) (scons_p p S1 Γ) M S2 →
has_type Γ (lam p M) (arr S1 S2)
| ty_app p (T : fin p → ty) (S : ty) (M : tm n) N :
has_type Γ M (arr T S) →
(∀ x, has_type Γ (N x) (T x)) →
has_type Γ (app p M N) S.
Notation "Γ ⊢ M : T" := (has_type Γ M T) (at level 20, M at level 99).
Lemma ty_var' n (x : fin n) A Γ:
A = Γ x → has_type Γ (var_tm x) A.
Proof. intros. subst. constructor. Qed.
Definition ltc {k k'} (Γ: ctx k) (Δ: ctx k') ρ := ∀ x, Δ (ρ x) = Γ x.
uU:--- variadic_preservation.v 33% L52 (Coq Script
2 subgoals (ID 254)
subgoal 1 (ID 254) is:
step (lam p b1[upList_tm_tm p f])
(lam p b2[upList_tm_tm p f])
subgoal 2 (ID 255) is:
step (app p s1[f] (cod_map (subst_tm f) t))
(app p s2[f] (cod_map (subst_tm f) t))
uU:%%- *goals* All L5 (Coq Goals Utoks)
mstep_inst is defined

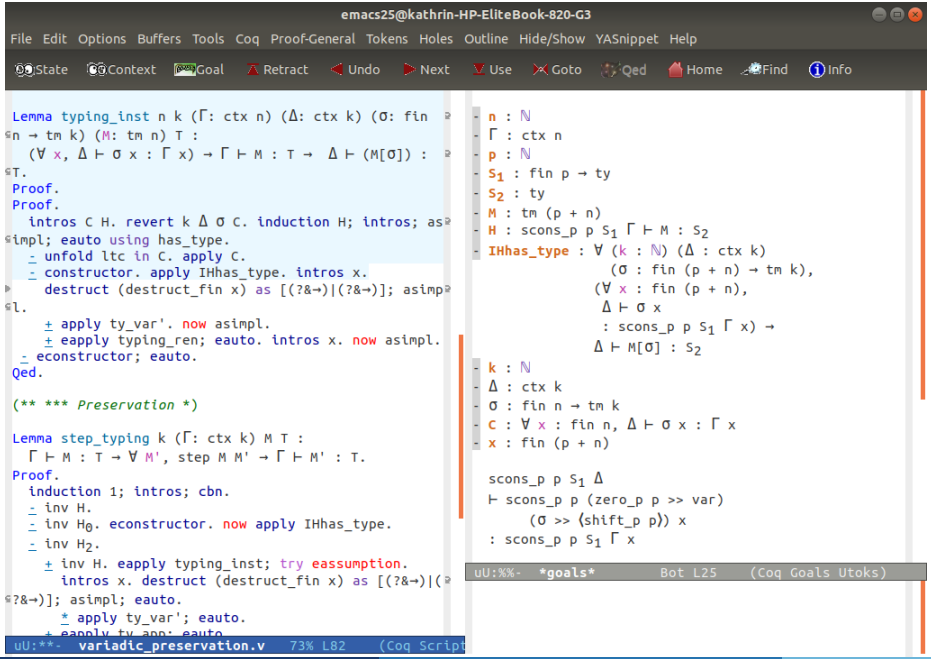

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
@has_type (ρ ⋅ n) (scons_p p s1 Γ) M s2 →
has_type Γ (lam p M) (arr s1 s2)
| ty_app p (T : fin p → ty) (S : ty) (M : tm n) N :
has_type Γ M (arr T S) →
(∀ x, has_type Γ (N x) (T x)) →
has_type Γ (app p M N) S.
Notation "Γ ⊢ M : T" := (has_type Γ M T) (at level 20, M at level 99).
Lemma ty_var' n (x : fin n) A Γ:
A = Γ x → has_type Γ (var_tm x) A.
Proof. intros. subst. constructor. Qed.
Definition ltc {k k'} (Γ: ctx k) (Δ: ctx k') ρ := ∀ x, Δ (ρ x) = Γ x.
Lemma typing_ren n k (Γ: ctx n) (Δ: ctx k) (ρ: fin n → fin k) (M: tm n) T :
ltc Γ Δ ρ → Γ ⊢ M : T → Δ ⊢ (M⟨ρ⟩) : T.
Proof.
intros C H. revert k Δ ρ C. induction H; intros; asimpl; eauto using has_type.
- unfold ltc in C. rewrite ← C. constructor.
- constructor. apply IHhas_type. intros x.
destruct (destruct_fin x) as [(?&→)|(?&→)]; eauto; asimpl; unfold upRen_p; asimpl. cbn. eauto.
+ now asimpl.
- econstructor; eauto.
Qed.
uU:--- variadic_preservation.v 43% L62 (Coq Script
2 subgoals (ID 254)
subgoal 1 (ID 254) is:
step (lam p b1[upList_tm_tm p f])
(lam p b2[upList_tm_tm p f])
subgoal 2 (ID 255) is:
step (app p s1[f] (cod_map (subst_tm f) t))
(app p s2[f] (cod_map (subst_tm f) t))
uU:%%- *goals* All L5 (Coq Goals Utoks)
ty_var' is defined

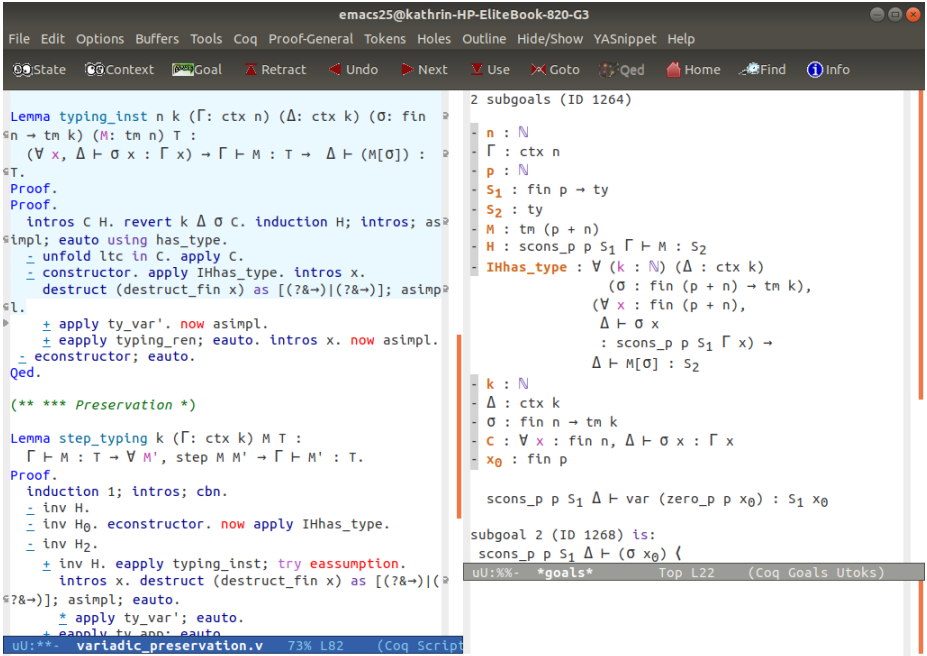

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
@has_type (ρ + n) (scons_p p S1 Γ) M S2 →
has_type Γ (lam p M) (arr S1 S2)
| ty_app p (T : fin p → ty) (S : ty) (M : tm n) N :
has_type Γ M (arr T S) →
(∀ x, has_type Γ (N x) (T x)) →
has_type Γ (app p M N) S.
Notation "Γ ⊢ M : T" := (has_type Γ M T) (at level 2
0, M at level 99).
Lemma ty_var' n (x : fin n) A Γ:
A = Γ x → has_type Γ (var_tm x) A.
Proof. intros. subst. constructor. Qed.
Definition ltc {k k'} (Γ: ctx k) (Δ: ctx k') ρ := ∀
x, Δ (ρ x) = Γ x.
Lemma typing_ren n k (Γ: ctx n) (Δ: ctx k) (ρ: fin n
→ fin k) (M: tm n) T :
ltc Γ Δ ρ → Γ ⊢ M : T → Δ ⊢ (M⟨ρ⟩) : T.
Proof.
intros C H. revert k Δ ρ C. induction H; intros; as
impl; eauto using has_type.
- unfold ltc in C. rewrite ← C. constructor.
- constructor. apply IHhas_type. intros x.
destruct (destruct_fin x) as [(?&→)|(?&→)]; eauto
; asimpl; unfold upRen_p; asimpl. cbn. eauto.
+ now asimpl.
- econstructor; eauto.
Qed.
uU:--- variadic_preservation.v 43% L65 (Coq Script
1 subgoal (ID 291)
n, k : ℕ
Γ : ctx n
Δ : ctx k
ρ : fin n → fin k
M : tm n
T : ty
ltc Γ Δ ρ →
Γ ⊢ M : T → Δ ⊢ M ⟨ρ⟩ : T
uU:%%- *goals* All L11 (Coq Goals Utoks)

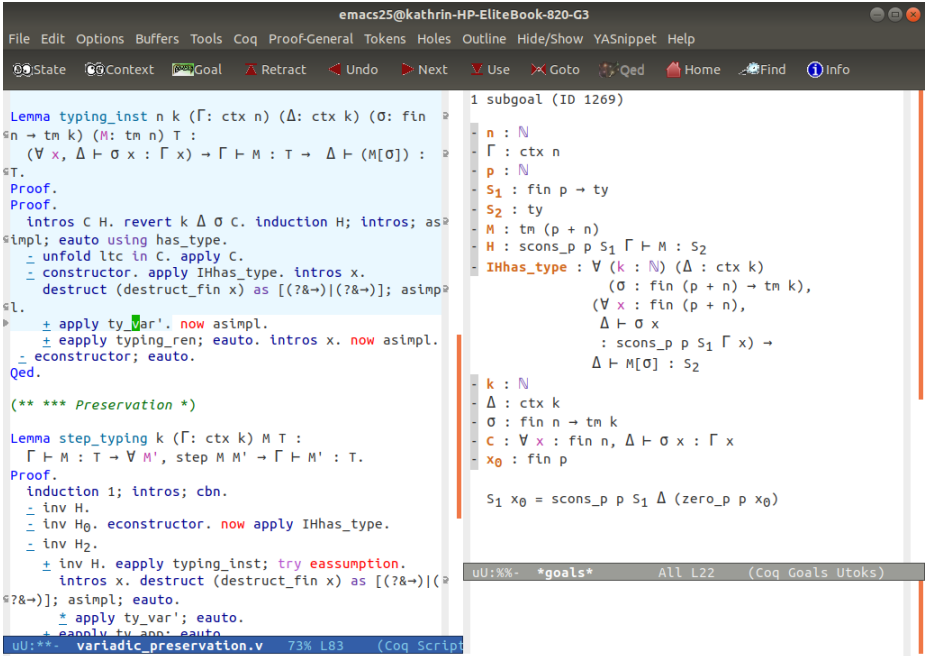

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
@has_type (p + n) (scons_p p S1 Γ) M S2 →
has_type Γ (lam p M) (arr S1 S2)
| ty_app p (T : fin p → ty) (S : ty) (M : tm n) N :
has_type Γ M (arr T S) →
(∀ x, has_type Γ (N x) (T x)) →
has_type Γ (app p M N) S.
Notation "Γ ⊢ M : T" := (has_type Γ M T) (at level 2
0, M at level 99).
Lemma ty_var' n (x : fin n) A Γ:
A = Γ x → has_type Γ (var_tm x) A.
Proof. intros. subst. constructor. Qed.
Definition ltc {k k'} (Γ: ctx k) (Δ: ctx k') ρ := ∀
x, Δ (ρ x) = Γ x.
Lemma typing_ren n k (Γ: ctx n) (Δ: ctx k) (ρ: fin n
→ fin k) (M: tm n) T :
ltc Γ Δ ρ → Γ ⊢ M : T → Δ ⊢ (M⟨ρ⟩) : T.
Proof.
intros C H. revert k Δ ρ C. induction H; intros; as
impl; eauto using has_type.
- unfold ltc in C. rewrite ← C. constructor.
- constructor. apply IHhas_type. intros x.
destruct (destruct_fin x) as [(?&→)|(?&→)]; eauto
; asimpl; unfold upRen_p; asimpl. cbn. eauto.
+ now asimpl.
- econstructor; eauto.
Qed.
uU:--- variadic_preservation.v 43% L70 (Coq Script
1 subgoal (ID 886)
n : ℕ
Γ : ctx n
p : ℕ
S1 : fin p → ty
S2 : ty
M : tm (p + n)
H : scons_p p S1 Γ ⊢ M : S2
IHhas_type : ∀ (k : ℕ) (Δ : ctx k)
(ρ : fin (p + n) → fin k),
ltc (scons_p p S1 Γ) Δ ρ →
Δ ⊢ M ⟨ρ⟩ : S2
k : ℕ
Δ : ctx k
ρ : fin n → fin k
C : ltc Γ Δ ρ
x : fin (p + n)
scons_p p S1 Δ (upRen_p p ρ x) =
scons_p p S1 Γ x
uU:%%- *goals* All L21 (Coq Goals Utoks)

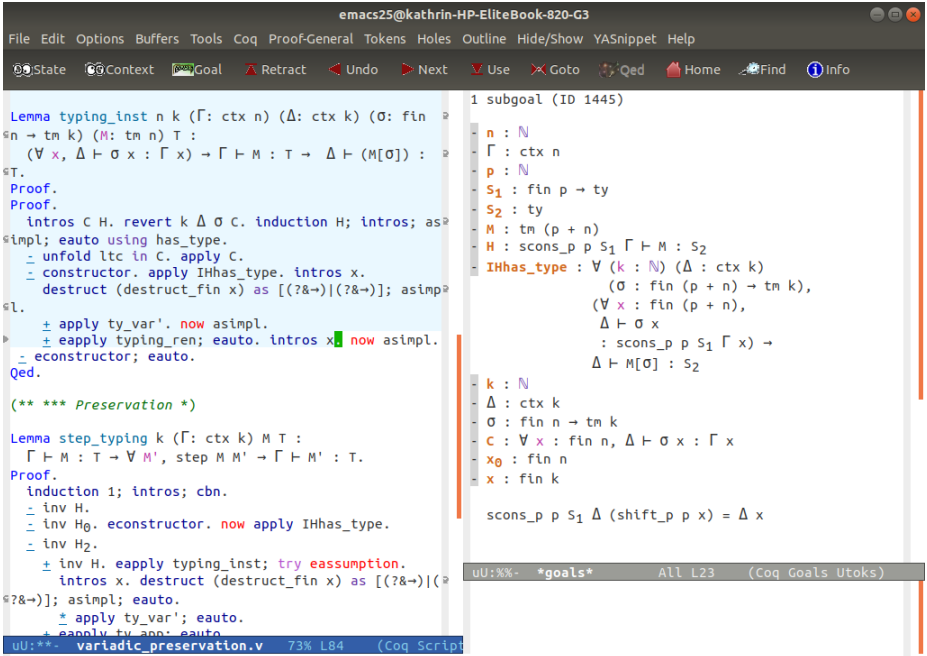

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
@has_type (ρ + n) (scons_p p S1 Γ) M S2 →
has_type Γ (lam p M) (arr S1 S2)
| ty_app p (T : fin p → ty) (S : ty) (M : tm n) N :
has_type Γ M (arr T S) →
(∀ x, has_type Γ (N x) (T x)) →
has_type Γ (app p M N) S.
Notation "Γ ⊢ M : T" := (has_type Γ M T) (at level 2
0, M at level 99).
Lemma ty_var' n (x : fin n) A Γ:
A = Γ x → has_type Γ (var_tm x) A.
Proof. intros. subst. constructor. Qed.
Definition ltc {k k'} (Γ: ctx k) (Δ: ctx k') ρ := ∀
x, Δ (ρ x) = Γ x.
Lemma typing_ren n k (Γ: ctx n) (Δ: ctx k) (ρ: fin n
→ fin k) (M: tm n) T :
ltc Γ Δ ρ → Γ ⊢ M : T → Δ ⊢ (M⟨ρ⟩) : T.
Proof.
intros C H. revert k Δ ρ C. induction H; intros; as
impl; eauto using has_type.
- unfold ltc in C. rewrite ← C. constructor.
- constructor. apply IHhas_type. intros x.
destruct (destruct_fin x) as [(?&→)|(?&→)]; eauto
; asimpl. cbn. eauto.
+ now asimpl.
- econstructor; eauto.
Qed.
uU:**- variadic_preservation.v 43% L70 (Coq Script
2 subgoals (ID 1022)
- n : ℕ
- Γ : ctx n
- p : ℕ
- S1 : fin p → ty
- S2 : ty
- M : tm (p + n)
- H : scons_p p S1 Γ ⊢ M : S2
- IHhas_type : ∀ (k : ℕ) (Δ : ctx k)
(ρ : fin (p + n) → fin k),
ltc (scons_p p S1 Γ) Δ ρ →
Δ ⊢ M ⟨ρ⟩ : S2
- k : ℕ
- Δ : ctx k
- ρ : fin n → fin k
- C : ltc Γ Δ ρ
- x0 : fin p
S1 x0 = S1 x0
subgoal 2 (ID 1317) is:
Δ (ρ x0) = Γ x0
uU:%%- *goals* All L20 (Coq Goals Utoks)

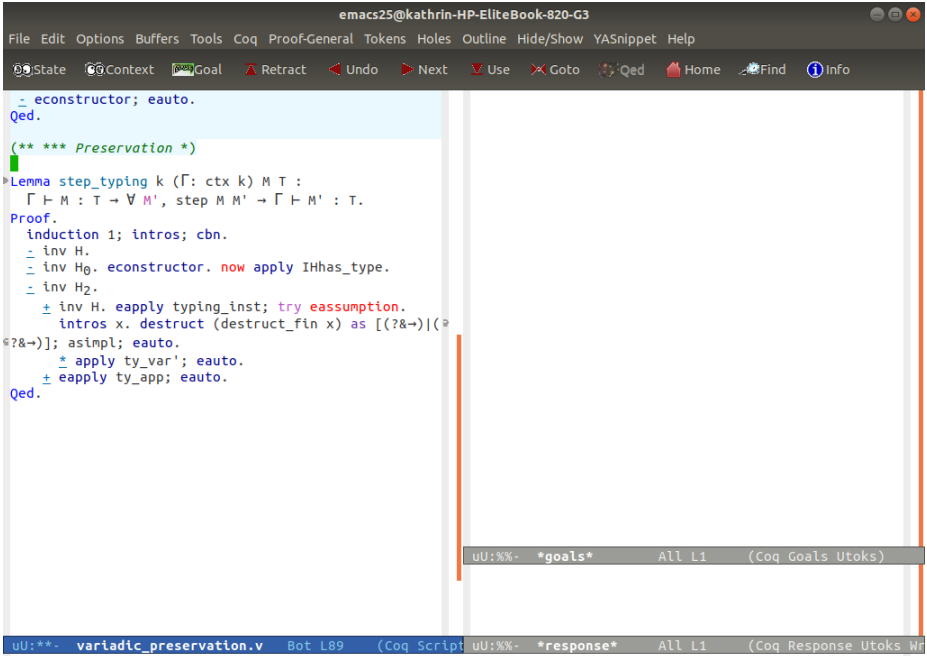

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
@has_type (ρ + n) (scons_p p S1 Γ) M S2 →
has_type Γ (lam p M) (arr S1 S2)
| ty_app p (T : fin p → ty) (S : ty) (M : tm n) N :
has_type Γ M (arr T S) →
(∀ x, has_type Γ (N x) (T x)) →
has_type Γ (app p M N) S.
Notation "Γ ⊢ M : T" := (has_type Γ M T) (at level 2
0, M at level 99).
Lemma ty_var' n (x : fin n) A Γ:
A = Γ x → has_type Γ (var_tm x) A.
Proof. intros. subst. constructor. Qed.
Definition ltc {k k'} (Γ: ctx k) (Δ: ctx k') ρ := ∀
x, Δ (ρ x) = Γ x.
Lemma typing_ren n k (Γ: ctx n) (Δ: ctx k) (ρ: fin n
→ fin k) (M: tm n) T :
ltc Γ Δ ρ → Γ ⊢ M : T → Δ ⊢ (M⟨ρ⟩) : T.
Proof.
intros C H. revert k Δ ρ C. induction H; intros; as
impl; eauto using has_type.
- unfold ltc in C. rewrite ← C. constructor.
- constructor. apply IHhas_type. intros x.
destruct (destruct_fin x) as [(?&→)|(?&→)]; eauto
; asimpl. cbn. eauto.
+ now asimpl.
- econstructor; eauto.
Qed.
uU:**- variadic_preservation.v 43% L71 (Coq Script
1 subgoal (ID 1317)
- n : ℕ
- Γ : ctx n
- p : ℕ
- S1 : fin p → ty
- S2 : ty
- M : tm (p + n)
- H : scons_p p S1 Γ ⊢ M : S2
- IHhas_type : ∀ (k : ℕ) (Δ : ctx k)
(ρ : fin (p + n) → fin k),
ltc (scons_p p S1 Γ) Δ ρ →
Δ ⊢ M ⟨ρ⟩ : S2
- k : ℕ
- Δ : ctx k
- ρ : fin n → fin k
- C : ltc Γ Δ ρ
- x0 : fin n
Δ (ρ x0) = Γ x0
uU:%%- *goals* All L20 (Coq Goals Utoks)

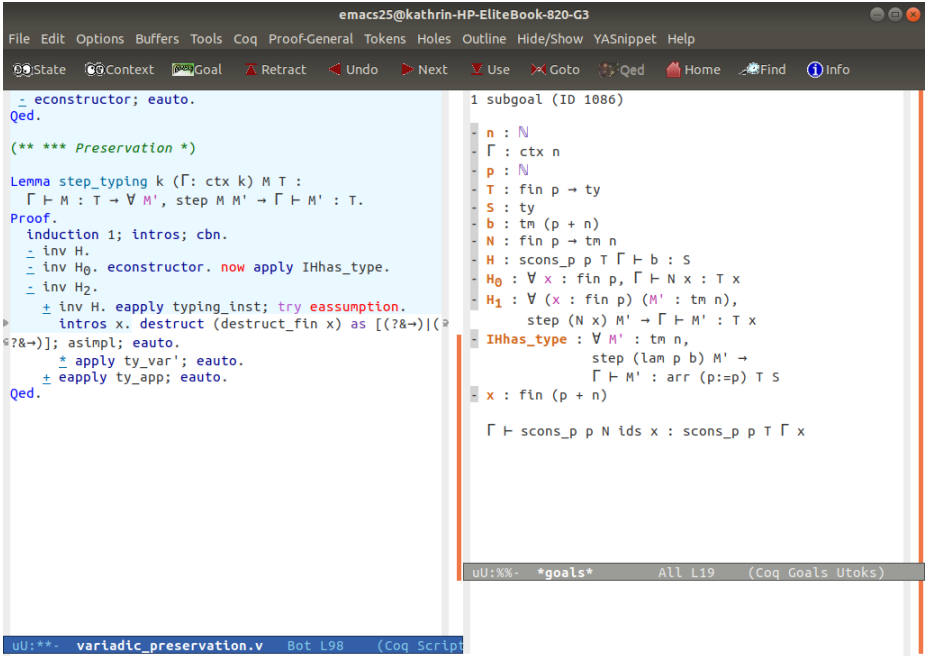

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
Lemma typing_inst n k (Γ: ctx n) (Δ: ctx k) (σ: fin n → tm k) (M: tm n) T :
(∀ x, Δ ⊢ σ x : Γ x) → Γ ⊢ M : T → Δ ⊢ (M[σ]) : T.
Proof.
Proof.
intros C H. revert k Δ σ C. induction H; intros; asimpl; eauto using has_type.
- unfold ltc in C. apply C.
- constructor. apply IHhas_type. intros x.
destruct (destruct_fin x) as [(?&→)|(?&→)]; asimpl.
+ apply ty_var'. now asimpl.
+ eapply typing_ren; eauto. intros x. now asimpl.
- econstructor; eauto.
Qed.
(** *** Preservation *)
Lemma step_typing k (Γ: ctx k) M T :
Γ ⊢ M : T → ∀ M', step M M' → Γ ⊢ M' : T.
Proof.
induction 1; intros; cbn.
- inv H.
- inv H0. econstructor. now apply IHhas_type.
- inv H2.
+ inv H. eapply typing_inst; try eassumption.
intros x. destruct (destruct_fin x) as [(?&→)|(?&→)]; asimpl; eauto.
* apply ty_var'; eauto.
+ eapply ty_app; eauto.
uU:**- variadic_preservation.v 73% L82 (Coq Script
n : ℕ
Γ : ctx n
p : ℕ
S1 : fin p → ty
S2 : ty
M : tm (p + n)
H : scons_p p S1 Γ ⊢ M : S2
IHhas_type : ∀ (k : ℕ) (Δ : ctx k)
(σ : fin (p + n) → tm k),
(∀ x : fin (p + n),
Δ ⊢ σ x
: scons_p p S1 Γ x) →
Δ ⊢ M[σ] : S2
k : ℕ
Δ : ctx k
σ : fin n → tm k
C : ∀ x : fin n, Δ ⊢ σ x : Γ x
x : fin (p + n)
scons_p p S1 Δ
⊢ scons_p p (zero_p p >> var)
(σ >> (shift_p p)) x
: scons_p p S1 Γ x
uU:%%- *goals* Bot L25 (Coq Goals Utoks)

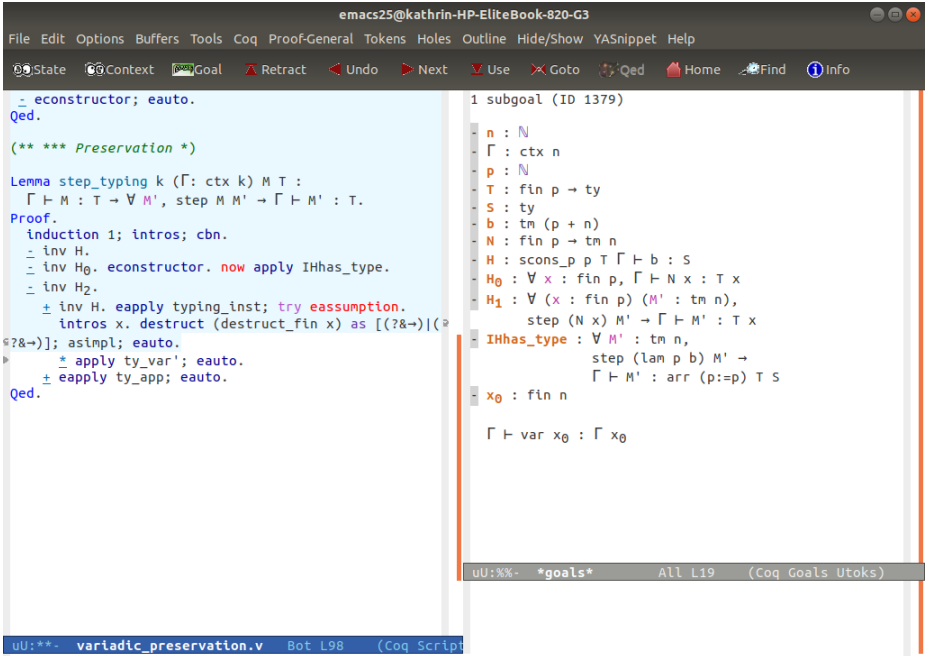

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
Lemma typing_inst n k (Γ: ctx n) (Δ: ctx k) (σ: fin n → tm k) (M: tm n) T :
(∀ x, Δ ⊢ σ x : Γ x) → Γ ⊢ M : T → Δ ⊢ (M[σ]) : T.
Proof.
Proof.
intros C H. revert k Δ σ C. induction H; intros; asimpl; eauto using has_type.
- unfold ltc in C. apply C.
- constructor. apply IHhas_type. intros x.
destruct (destruct_fin x) as [(?&→)|(?&→)]; asimpl.
+ apply ty_var'. now asimpl.
+ eapply typing_ren; eauto. intros x. now asimpl.
- econstructor; eauto.
Qed.
(** *** Preservation *)
Lemma step_typing k (Γ: ctx k) M T :
Γ ⊢ M : T → ∀ M', step M M' → Γ ⊢ M' : T.
Proof.
induction 1; intros; cbn.
- inv H.
- inv H0. econstructor. now apply IHhas_type.
- inv H2.
+ inv H. eapply typing_inst; try eassumption.
intros x. destruct (destruct_fin x) as [(?&→)|(?&→)]; asimpl; eauto.
* apply ty_var'; eauto.
+ eapply ty_app; eauto.
uU:**- variadic_preservation.v 73% L82 (Coq Script
2 subgoals (ID 1264)
- n : ℕ
- Γ : ctx n
- p : ℕ
- S1 : fin p → ty
- S2 : ty
- M : tm (p + n)
- H : scons_p p S1 Γ ⊢ M : S2
- IHhas_type : ∀ (k : ℕ) (Δ : ctx k) (σ : fin (p + n) → tm k), (∀ x : fin (p + n), Δ ⊢ σ x : scons_p p S1 Γ x) → Δ ⊢ M[σ] : S2
- k : ℕ
- Δ : ctx k
- σ : fin n → tm k
- C : ∀ x : fin n, Δ ⊢ σ x : Γ x
- x0 : fin p
scons_p p S1 Δ ⊢ var (zero_p p x0) : S1 x0
subgoal 2 (ID 1268) is:
scons_p p S1 Δ ⊢ (σ x0) ⟨
uU:%%- *goals* Top L22 (Coq Goals Utoks)

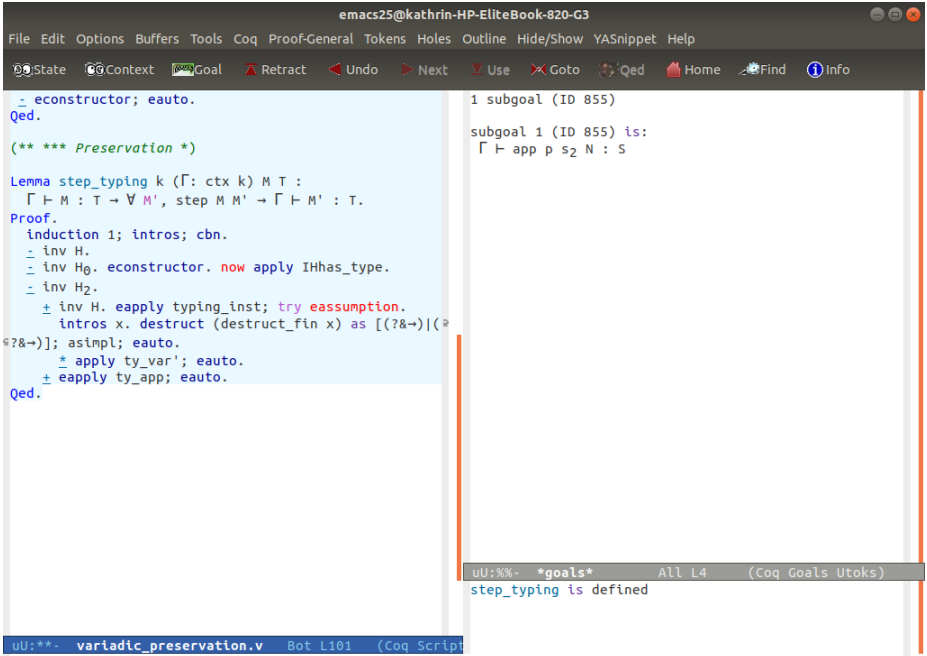

emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
Lemma typing_inst n k (Γ: ctx n) (Δ: ctx k) (σ: fin n → tm k) (M: tm n) T :
(∀ x, Δ ⊢ σ x : Γ x) → Γ ⊢ M : T → Δ ⊢ (M[σ]) : T.
Proof.
Proof.
intros C H. revert k Δ σ C. induction H; intros; asimpl; eauto using has_type.
- unfold ltc in C. apply C.
- constructor. apply IHhas_type. intros x.
destruct (destruct_fin x) as [(?&→)|(?&→)]; asimpl.
+ apply ty_Var'. now asimpl.
+ eapply typing_ren; eauto. intros x. now asimpl.
- econstructor; eauto.
Qed.
(** *** Preservation *)
Lemma step_typing k (Γ: ctx k) M T :
Γ ⊢ M : T → ∀ M', step M M' → Γ ⊢ M' : T.
Proof.
induction 1; intros; cbn.
- inv H.
- inv H0. econstructor. now apply IHhas_type.
- inv H2.
+ inv H. eapply typing_inst; try eassumption.
intros x. destruct (destruct_fin x) as [(?&→)|(?&→)]; asimpl; eauto.
* apply ty_var'; eauto.
+ eapply ty_app; eauto.
uU:**- variadic_preservation.v 73% L83 (Coq Script
1 subgoal (ID 1269)
n : ℕ
Γ : ctx n
p : ℕ
S1 : fin p → ty
S2 : ty
M : tm (p + n)
H : scons_p p S1 Γ ⊢ M : S2
IHhas_type : ∀ (k : ℕ) (Δ : ctx k) (σ : fin (p + n) → tm k), (∀ x : fin (p + n), Δ ⊢ σ x : scons_p p S1 Γ x) → Δ ⊢ M[σ] : S2
k : ℕ
Δ : ctx k
σ : fin n → tm k
C : ∀ x : fin n, Δ ⊢ σ x : Γ x
x0 : fin p
S1 x0 = scons_p p S1 Δ (zero_p p x0)
uU:%%- *goals* All L22 (Coq Goals Utoks)


emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
Lemma typing_inst n k (Γ: ctx n) (Δ: ctx k) (σ: fin n → tm k) (M: tm n) T :
(∀ x, Δ ⊢ σ x : Γ x) → Γ ⊢ M : T → Δ ⊢ (M[σ]) : T.
Proof.
Proof.
intros C H. revert k Δ σ C. induction H; intros; asimpl; eauto using has_type.
- unfold ltc in C. apply C.
- constructor. apply IHhas_type. intros x.
destruct (destruct_fin x) as [(?&→)|(?&→)]; asimpl.
+ apply ty_var'. now asimpl.
+ eapply typing_ren; eauto. intros x. now asimpl.
- econstructor; eauto.
Qed.
(** *** Preservation *)
Lemma step_typing k (Γ: ctx k) M T :
Γ ⊢ M : T → ∀ M', step M M' → Γ ⊢ M' : T.
Proof.
induction 1; intros; cbn.
- inv H.
- inv H0. econstructor. now apply IHhas_type.
- inv H2.
+ inv H. eapply typing_inst; try eassumption.
intros x. destruct (destruct_fin x) as [(?&→)|(?&→)]; asimpl; eauto.
* apply ty_var'; eauto.
+ eapply ty_app; eauto.
uU:**- variadic_preservation.v 73% L84 (Coq Script
1 subgoal (ID 1445)
n : ℕ
Γ : ctx n
p : ℕ
S1 : fin p → ty
S2 : ty
M : tm (p + n)
H : scons_p p S1 Γ ⊢ M : S2
IHhas_type : ∀ (k : ℕ) (Δ : ctx k) (σ : fin (p + n) → tm k), (∀ x : fin (p + n), Δ ⊢ σ x : scons_p p S1 Γ x) → Δ ⊢ M[σ] : S2
k : ℕ
Δ : ctx k
σ : fin n → tm k
C : ∀ x : fin n, Δ ⊢ σ x : Γ x
x0 : fin n
x : fin k
scons_p p S1 Δ (shift_p p x) = Δ x
uU:%%- *goals* All L23 (Coq Goals Utoks)


emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
- econstructor; eauto.
Qed.
(** *** Preservation *)
Lemma step_typing k (Γ: ctx k) M T :
Γ ⊢ M : T → ∀ M', step M M' → Γ ⊢ M' : T.
Proof.
induction 1; intros; cbn.
- inv H.
- inv H0. econstructor. now apply IHhas_type.
- inv H2.
+ inv H. eapply typing_inst; try eassumption.
intros x. destruct (destruct_fin x) as [(?&→)|(?
?&→)]; asimpl; eauto.
* apply ty_var'; eauto.
+ eapply ty_app; eauto.
Qed.
uU:%%- *goals* All L1 (Coq Goals Utoks)
uU:**- variadic_preservation.v Bot L89 (Coq Script
uU:%%- *response* All L1 (Coq Response Utoks Wr


emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
- econstructor; eauto.
Qed.
(** *** Preservation *)
Lemma step_typing k (Γ: ctx k) M T :
Γ ⊢ M : T → ∀ M', step M M' → Γ ⊢ M' : T.
Proof.
induction 1; intros; cbn.
- inv H.
- inv H0. econstructor. now apply IHhas_type.
- inv H2.
+ inv H. eapply typing_inst; try eassumption.
intros x. destruct (destruct_fin x) as [(?&→)|(?&→)]; asimpl; eauto.
* apply ty_var'; eauto.
+ eapply ty_app; eauto.
Qed.
uU:**- variadic_preservation.v Bot L98 (Coq Script
1 subgoal (ID 1086)
n : ℕ
Γ : ctx n
p : ℕ
T : fin p → ty
S : ty
b : tm (p + n)
N : fin p → tm n
H : scons_p p T Γ ⊢ b : S
H0 : ∀ x : fin p, Γ ⊢ N x : T x
H1 : ∀ (x : fin p) (M' : tm n),
step (N x) M' → Γ ⊢ M' : T x
IHhas_type : ∀ M' : tm n,
step (lam p b) M' →
Γ ⊢ M' : arr (p:=p) T S
x : fin (p + n)
Γ ⊢ scons_p p N ids x : scons_p p T Γ x
uU:%%- *goals* All L19 (Coq Goals Utoks)


emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
- econstructor; eauto.
Qed.
(** *** Preservation *)
Lemma step_typing k (Γ: ctx k) M T :
Γ ⊢ M : T → ∀ M', step M M' → Γ ⊢ M' : T.
Proof.
induction 1; intros; cbn.
- inv H.
- inv H0. econstructor. now apply IHhas_type.
- inv H2.
+ inv H. eapply typing_inst; try eassumption.
intros x. destruct (destruct_fin x) as [(?&→)|(?
?&→)]; asimpl; eauto.
* apply ty_var'; eauto.
+ eapply ty_app; eauto.
Qed.
uU:**- variadic_preservation.v Bot L98 (Coq Script
1 subgoal (ID 1379)
n : ℕ
Γ : ctx n
p : ℕ
T : fin p → ty
S : ty
b : tm (p + n)
N : fin p → tm n
H : scons_p p T Γ ⊢ b : S
H0 : ∀ x : fin p, Γ ⊢ N x : T x
H1 : ∀ (x : fin p) (M' : tm n),
step (N x) M' → Γ ⊢ M' : T x
IHhas_type : ∀ M' : tm n,
step (lam p b) M' →
Γ ⊢ M' : arr (p:=p) T S
x0 : fin n
Γ ⊢ var x0 : Γ x0
uU:%%- *goals* All L19 (Coq Goals Utoks)


emacs25@kathrin-HP-EliteBook-820-G3
File Edit Options Buffers Tools Coq Proof-General Tokens Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info
- econstructor; eauto.
Qed.
(** *** Preservation *)
Lemma step_typing k (Γ: ctx k) M T :
Γ ⊢ M : T → ∀ M', step M M' → Γ ⊢ M' : T.
Proof.
induction 1; intros; cbn.
- inv H.
- inv H0. econstructor. now apply IHhas_type.
- inv H2.
+ inv H. eapply typing_inst; try eassumption.
intros x. destruct (destruct_fin x) as [(?&→)|(?
?&→)]; asimpl; eauto.
* apply ty_var'; eauto.
+ eapply ty_app; eauto.
Qed.
uU:**- variadic_preservation.v Bot L101 (Coq Script
1 subgoal (ID 855)
subgoal 1 (ID 855) is:
Γ ⊢ app p s2 N : S
uU:%%- *goals* All L4 (Coq Goals Utoks)
step_typing is defined

Outline

What are Future Gaps to Bridge?

- We wish for:
  - More expressive modular syntax
  - Automatic generation of context renaming and context morphism lemmas [Keuchel '19]
  - Support for recursive functions [Allais et al. '19, Kaiser et al. '18]
- How do different solutions compare beyond type safety?
  - POPLMark Reloaded Challenge [Abel et al. '19]:
    Strong normalisation with Kripke-style logical relations
  - Similar problems for anti-renaming lemma

Wrap-up

- De Bruijn syntax is regular enough to allow for code generation
- Tool support, for example in the form of Autosubst 2, can relieve users from binder boilerplate

Available online:

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