AUTOSUBST: Automation for de Bruijn Substitutions

https://www.ps.uni-saarland.de/autosubst

Steven Schäfer Tobias Tebbi Gert Smolka

Department of Computer Science
Saarland University, Germany

August 13, 2014

Overview

- Autosubst Overview
  - Full de Bruijn Substitutions.
  - Relevant Substitutions.
  - Simplifying Substitution Expressions.
- Proof Techniques.
- Implementation.

Syntax with Binding

$$\lambda f \,.\, f \,(\lambda x \,.\, f\, x)$$

- Represent & reason about syntax modulo α-equivalence.
- Representations Techniques:
    - de Bruijn
    - named
    - locally nameless
    - nominal
    - higher-order
    - …

Syntax with Binding

$$\lambda\, 0\, (\lambda\, 1\, 0)$$

- Represent & reason about syntax modulo α-equivalence.
- Representations Techniques:
  - de Bruijn
  - named
  - locally nameless
  - nominal
  - higher-order
  - …

Parallel Substitutions [de Bruijn 1972]

- *Substitutions* $\sigma, \tau : \text{var} \to \text{term}$
- *Renamings* $\xi, \zeta : \text{var} \to \text{var}$
- Substitution application for λ-calculus ($s, t ::= x \mid s\,t \mid \lambda\, s$)

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

$$(\Uparrow\sigma)(0) = 0$$
$$(\Uparrow\sigma)(x+1) = \sigma(x).[{+1}]$$

- Not structurally recursive
- Renaming $s.[\xi]$ can be defined by structural recursion. Use renamings to define $\Uparrow\sigma$. [Adams 2006]

Relevant Substitutions [Abadi Cardelli Curien Lévy 1991]

- ids ("Identity Substitution")

$$\mathsf{ids}(x) = x$$

- $+1$ ("Lift")

$$(+1)(x) = x + 1$$

- $\sigma \gg \tau$ ("Composition", σ then τ)

$$(\sigma \gg \tau)(x) = \sigma(x).[\tau]$$

- $s \mathrel{.:} \sigma$ ("Cons")

$$(s \mathrel{.:} \sigma)(0) = s$$
$$(s \mathrel{.:} \sigma)(x + 1) = \sigma(x)$$

Relevant Substitutions [Abadi Cardelli Curien Lévy 1991]

$$(s .: \sigma)(0) = s$$
$$(s .: \sigma)(x+1) = \sigma(x) \qquad \mathsf{ids}(x) = x$$
$$(\sigma \gg \tau)(x) = \sigma(x).[\tau] \qquad ({+1})(x) = x + 1$$

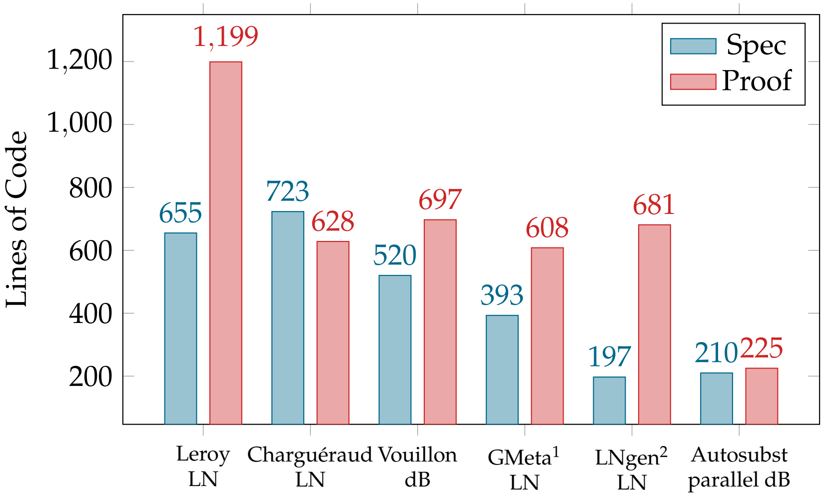
- Note that

$$\Uparrow\sigma = 0 .: (\sigma \gg {+1})$$
$$(\lambda\, s)\, t \text{ reduces to } s[t .: \mathsf{ids}]$$

- This yields a model of the σ-calculus by Abadi et. al.
- In Coq: normalize substitution expressions using `asimpl`, solve equations using `autosubst`.

DEMO: SUBSTITUTIVITY

POPLMARK[Aydemir et al. 2005] Comparison (1a + 2a)



[1] [Lee Oliveira Cho Yi 2012]

[2] [Aydemir Weirich 2010]

CASE STUDIES

Lines of Code, per `coqwc`

	Spec	Proof
POPLmark[3]: $F_{<:}$ Preservation & Progress	210	225
$F_{<:}$ Preservation & Progress	185	146
Normalization for CBV System F	99	54
Strong Normalization for System F	153	96
Type Preservation for (predicative) CC_ω	214	229

[3] Aydemir et. al. 2005, mostly following the paper proofs

DEMO: STRONG NORMALIZATION OF SYSTEM F

Example: Type Preservation

- In order to show type preservation…

$$\frac{\Gamma \vdash s : A \qquad s \rhd t}{\Gamma \vdash t : A}$$

- …we need a substitution lemma.

$$\frac{\Gamma, A \vdash s : B \qquad \Gamma \vdash t : A}{\Gamma \vdash s.[t \mathrel{.:} \mathsf{ids}] : B}$$

Substitution Lemmas

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

$$\sigma : \Delta \to \Gamma := \forall x,\ \Delta \vdash \sigma(x) : \Gamma(x)$$

↑ maximal generalization

$$\frac{\Gamma, A, \Delta \vdash s : B \qquad \Gamma \vdash t : A}{\Gamma, \Delta \vdash s.[|\Delta| \mapsto t] : B}$$

$$x \mapsto t := \Uparrow^{x}(t \mathrel{.:} \mathsf{ids})$$

↑ minimal generalization

$$\frac{\Gamma, A \vdash s : B \qquad \Gamma \vdash t : A}{\Gamma \vdash s.[t \mathrel{.:} \mathsf{ids}] : B}$$

USING THE GENERALIZED SUBSTITUTION LEMMA

$$\frac{\Gamma \vdash s : A \qquad \sigma : \Delta \to \Gamma}{\Delta \vdash s.[\sigma] : A} \qquad\qquad \begin{array}{c}\sigma : \Delta \to \Gamma := \\ \forall x,\ \Delta \vdash \sigma(x) : \Gamma(x)\end{array}$$

- Subsumes weakening, substitution, contraction, exchange…

$$\begin{array}{rcll} {}_{+}1 & : & \Gamma, A \to \Gamma & \\ (s \mathbin{.:} \mathsf{ids}) & : & \Gamma \to \Gamma, A & \qquad \text{if } \Gamma \vdash s : A \end{array}$$

- Subsumes narrowing in $F_{<:}$
- Subsumes context conversion in Type Theory

Coq Implementation

- Generated substitution application must fulfill

$$\mathsf{ids}(x).[\sigma] = \sigma(x)$$
$$s.[\mathsf{ids}] = s$$
$$s.[\sigma].[\tau] = s.[\sigma \gg \tau]$$

- Equations between functions $\Rightarrow$ assume functional extensionality.
- Synthesize proofs using Ltac.
- Synthesize substitution application using Ltac.

LTAC METAPROGRAMMING

- Use fix/destruct to generate recursive definitions.
- Annotate goal to remember the shape of terms.

Definition annot $s\ a := s$.

$$\frac{a : \mathsf{bool}}{\mathsf{nat}} \overset{\text{change}}{\rightsquigarrow} \frac{a : \mathsf{bool}}{\mathsf{annot\ nat}\ a} \overset{\text{destruct}}{\rightsquigarrow} \frac{}{\mathsf{annot\ nat\ true}}$$

RELATED COQ DEVELOPMENTS

LNgen[Aydemir10] : External tool, LN

GMeta[Lee12] : Generic term type, LN + dB

DBgen[Polonowski13] : External tool, dB

DBlib[Pottier13] : Library, dB

CFGV[Abhishek14] : Generic term type, named

Thank you for your attention!

Try Autosubst today

https://www.ps.uni-saarland.de/autosubst

Questions?

THE RULES OF THE σ-CALCULUS

- The defining equations shown before.

$$(s\,t).[\sigma] = s.[\sigma]\,t.[\sigma] \qquad (\lambda s).[\sigma] = \lambda s.[0 \mathrel{.:} \sigma \gg {+1}] \qquad 0.[s \mathrel{.:} \sigma] = s$$

- Monoid action laws for composition and substitution application

$$s.[\mathsf{ids}] = s \qquad \mathsf{ids} \gg \sigma = \sigma \qquad (\sigma \gg \tau) \gg \theta = \sigma \gg \tau \gg \theta$$
$$s.[\sigma].[\tau] = s.[\sigma \gg \tau] \qquad \sigma \gg \mathsf{ids} = \sigma$$

- Interaction between lift and cons

$${+1} \gg s \mathrel{.:} \sigma = \sigma \qquad 0.[\sigma] \mathrel{.:} {+1} \gg \sigma = \sigma \qquad (s \mathrel{.:} \sigma) \gg \tau = s.[\tau] \mathrel{.:} \sigma \gg \tau$$

- This is a convergent rewriting system and complete for the untyped λ-calculus

Proving the generalized substitution lemma

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

$$\begin{gathered}\sigma : \Delta \to \Gamma := \\ \forall x,\ \Delta \vdash \sigma(x) : \Gamma(x)\end{gathered}$$

- We need

$$\frac{\sigma : \Delta \to \Gamma}{\Uparrow\sigma : \Delta, A \to \Gamma, A}$$

- Specialize to renaming

$$\begin{aligned}&\xi : \Delta \to \Gamma := \\ &\quad \forall x \in \mathrm{Dom}(\Gamma), \xi_x \in \mathrm{Dom}(\Delta) \land \Delta(\xi_x) = \Gamma(x).[\xi]\end{aligned}$$

Heterogenous Substitutions (1)

- Consider many-sorted syntax, e.g., System F

$$A, B ::= X \mid A \to B \mid \forall A$$
$$s, t ::= x \mid s\,t \mid \lambda A\,s \mid s\,A \mid \Lambda\,s$$

- Substitution application

$$(s\,A).[\sigma] = s.[\sigma]\,A$$
$$(\Lambda\,s).[\sigma] = \Lambda\,s.[\sigma \gg\!\!| \,{+1}]$$

Heterogenous Substitutions (2)

- Heterogenous composition $(\sigma \mathbin{⨾} \tau)_x = \sigma_x.\llbracket\tau\rrbracket$
- Substitute types in terms.

$$x.\llbracket\theta\rrbracket = x \qquad (\lambda A\, s).\llbracket\theta\rrbracket = \lambda A.[\theta]\, s.\llbracket\theta\rrbracket$$
$$(s\, t).\llbracket\theta\rrbracket = s.\llbracket\theta\rrbracket\, t.\llbracket\theta\rrbracket \qquad (\Lambda\, s).\llbracket\theta\rrbracket = \Lambda\, s.\llbracket{\Uparrow}\theta\rrbracket$$
$$(s\, A).\llbracket\theta\rrbracket = s.\llbracket\theta\rrbracket\, A.[\theta]$$

- Structurally recursive.
- (Some) laws for heterogenous substitutions

$$\xi \mathbin{⨾} \theta = \xi$$
$$s.[\sigma].\llbracket\theta\rrbracket = s.\llbracket\theta\rrbracket.[\sigma \mathbin{⨾} \theta]$$