Confluence of a Good Calculus with Explicit Substitutions Formalized

Flávio L. C. de Moura and Leandro Oliveira Rezende

September 20, 2018

Universidade de Brasília

Explicit Substitutions

- Starting point: λ-calculus

$$\mathcal{T} ::= x \mid \lambda x.\mathcal{T} \mid \mathcal{T}\mathcal{T}$$

$$(\lambda x.t)\ u \to_\beta t\{x/u\}$$

Explicit Substitutions

- Starting point: λ-calculus

$$\mathcal{T} ::= x \mid \lambda x.\mathcal{T} \mid \mathcal{T}\mathcal{T}$$

$$(\lambda x.t)\ u \to_\beta t\{x/u\}$$

- Extending the λ-calculus with an explicit substitution operator: calculi with explicit substitutions

$$\mathcal{T} ::= x \mid \lambda x.\mathcal{T} \mid \mathcal{T}\mathcal{T} \mid \mathcal{T}[x/\mathcal{T}]$$

Explicit Substitutions

- Starting point: λ-calculus

$$\mathcal{T} ::= x \mid \lambda x.\mathcal{T} \mid \mathcal{T}\mathcal{T}$$

$$(\lambda x.t)\ u \rightarrow_{\beta} t\{x/u\}$$

- Extending the λ-calculus with an explicit substitution operator: calculi with explicit substitutions

$$\mathcal{T} ::= x \mid \lambda x.\mathcal{T} \mid \mathcal{T}\mathcal{T} \mid \mathcal{T}[x/\mathcal{T}]$$

$$(\lambda x.t)\ u \qquad \rightarrow \quad t[x/u]$$

Explicit Substitutions

- Starting point: λ-calculus

$$\mathcal{T} ::= x \mid \lambda x.\mathcal{T} \mid \mathcal{T}\mathcal{T}$$

$$(\lambda x.t)\ u \to_\beta t\{x/u\}$$

- Extending the λ-calculus with an explicit substitution operator: calculi with explicit substitutions

$$\mathcal{T} ::= x \mid \lambda x.\mathcal{T} \mid \mathcal{T}\mathcal{T} \mid \mathcal{T}[x/\mathcal{T}]$$

$$\begin{array}{lcl}
(\lambda x.t)\ u & \to & t[x/u] \\
x[x/u] & \to & u \\
y[x/u] & \to & y \\
(\lambda y.t')[x/u] & \to & \lambda y.t'[x/u] \\
(t'\ t'')[x/u] & \to & t'[x/u]\ t''[x/u]
\end{array}$$

The $\lambda\sigma$-calculus

Terms	a, b	$::=$	$\underline{\mathbf{1}} \mid \lambda a \mid a\ b \mid a[s]$
Substituições	s, t	$::=$	$id \mid \uparrow \mid s.t \mid s \circ t$

The $\lambda\mathtt{ex}$-calculus

- Developed by Delia Kesner [1]

$$\mathcal{T} ::= x \mid \lambda x.\mathcal{T} \mid \mathcal{T}\mathcal{T} \mid \mathcal{T}[x/\mathcal{T}]$$

$$
\begin{array}{llll}
t[x/u][y/v] & =_C & t[y/v][x/u], & \text{if } y \notin \mathtt{fv}(u) \text{ and } x \notin \mathtt{fv}(v) \\
\\
(\lambda x.t)\ u & \to_{\mathtt{B}} & t[x/u] & \\
x[x/u] & \to_{\mathtt{Var}} & u & \\
t[x/u] & \to_{\mathtt{Gc}} & t, & \text{if } x \notin \mathtt{fv}(t) \\
(t\ \ v)[x/u] & \to_{\mathtt{App}} & t[x/u]\ v[x/u] & \\
(\lambda y.t)[x/u] & \to_{\mathtt{Lamb}} & \lambda y.t[x/u] & \\
t[x/u][y/v] & \to_{\mathtt{Comp}} & t[y/v][x/u[y/v]], & \text{if } y \in \mathtt{fv}(u)
\end{array}
$$

The λex-calculus

- **Goal**: Formal proof of confluence (with code extraction).
  - Proof assistant: Coq.
  - Framework: Locally nameless representation (Arthur Charguéraud).
    + No need for α-conversion (bound variables are De Bruijn indexes).
    + No need referential contexts (named free variables).
    – The whole library is non-constructive.

The λex-calculus

$$\mathcal{T} ::= x \mid \mathcal{T}\mathcal{T} \mid \lambda x.\mathcal{T} \mid \mathcal{T}[x/\mathcal{T}]$$

```
Inductive pterm : Set :=
  | pterm_bvar : nat → pterm
  | pterm_fvar : var → pterm
  | pterm_app : pterm → pterm → pterm
  | pterm_abs : pterm → pterm
  | pterm_sub : pterm → pterm → pterm.

Inductive term : pterm → Prop :=
  | term_var : ∀ x, term (pterm_fvar x)
  | term_app : ∀ t1 t2,term t1 → term t2 →
                    term (pterm_app t1 t2)
  | term_abs : ∀ L t1,(∀ x, x \notin L → term (t1 ^ x)) →
                                      term (pterm_abs t1)
  | term_sub : ∀ L t1 t2, (∀ x, x \notin L → term (t1 ^ x)) →
                          term t2 → term (pterm_sub t1 t2).
```

The λex-calculus

$$t[x/u][y/v] =_C t[y/v][x/u], \text{ if } y \notin \mathtt{fv}(u) \text{ and } x \notin \mathtt{fv}(v)$$

```
Inductive eqc : Rel pterm :=
| eqc_def: ∀ t u v, term u → term v → eqc (t[u][v]) ((& t)[v][u]).

Definition eqc_ctx (t u: pterm) := ES_contextual_closure eqc t u.

Definition eqC (t : pterm) (u : pterm) := refltrans eqc_ctx t u.
Notation "t =C u" := (eqC t u) (at level 66).
```

The λex-calculus

$$(\lambda x.t)\ u \rightarrow_{\mathtt{B}} t[x/u]$$

```
Inductive rule_b : Rel pterm :=
  reg_rule_b : ∀ (t u:pterm),
    rule_b (pterm_app(pterm_abs t) u) (t[u]).

Definition b_ctx t u := ES_contextual_closure rule_b t u.
Notation "t →B u" := (b_ctx t u) (at level 66).
```

The $\lambda\mathtt{ex}$-calculus

$$\begin{array}{llll}
x[x/u] & \rightarrow_{\mathtt{Var}} & u & \\
t[x/u] & \rightarrow_{\mathtt{Gc}} & t, & \text{if } x \notin \mathtt{fv}(t) \\
(t\ v)[x/u] & \rightarrow_{\mathtt{App}} & t[x/u]\ v[x/u] & \\
(\lambda y.t)[x/u] & \rightarrow_{\mathtt{Lamb}} & \lambda y.t[x/u] & \\
t[x/u][y/v] & \rightarrow_{\mathtt{Comp}} & t[y/v][x/u[y/v]], & \text{if } y \in \mathtt{fv}(u)
\end{array}$$

```
Inductive sys_x : Rel pterm :=
| reg_rule_var : ∀ t, sys_x (pterm_bvar 0 [t]) t
| reg_rule_gc : ∀ t u, sys_x (t[u]) t
| reg_rule_app : ∀ t1 t2 u,
  sys_x ((pterm_app t1 t2)[u]) (pterm_app (t1[u]) (t2[u]))
| reg_rule_lamb : ∀ t u,
  sys_x ((pterm_abs t)[u]) (pterm_abs ((& t)[u]))
| reg_rule_comp : ∀ t u v, has_free_index 0 u →
  sys_x (t[u][v]) (((& t)[v])[ u[ v ] ]).
```

The $\lambda\mathtt{ex}$-calculus

```
Corollary lex_is_confluent: Confl lex.
Proof.
  apply Zprop_implies_Confl.
  apply Zlex.
Qed.
```

Confluence and the Z Property

Definition (Z Property)

Let $(A, \to)$ be an abstract rewriting system (ARS). The system $(A, \to)$ has the Z property, if there exists a map $f : A \to A$ such that:

$$\begin{tikzcd}
a \arrow[r] & b \arrow[dl, dotted, two heads] \\
f(a) \arrow[r, dotted, two heads] & f(b)
\end{tikzcd}$$

```
Definition Zprop {A:Type} (R: Rel A) := ∃ f:A → A, ∀ a b, R a b →
((refltrans R) b (f a) ∧ (refltrans R) (f a) (f b)).
```

Confluence and the Z Property

Definition (Confluence)

An ARS $(A, \to)$ is confluent if

$$\begin{tikzcd} & a \arrow[dl, twoheadrightarrow] \arrow[dr, twoheadrightarrow] & \\ b \arrow[dr, dotted, twoheadrightarrow] & & c \arrow[dl, dotted, twoheadrightarrow] \\ & d & \end{tikzcd}$$

```
Definition Confl {A:Type} (R: Rel A) := ∀ a b c, (refltrans R) a b →
(refltrans R) a c → (∃ d, (refltrans R) b d ∧ (refltrans R) c d).
```

Confluence and the Z Property

$$
\begin{tikzcd}
a \arrow[r] & b \arrow[dl, dotted, two heads] \\
f(a) \arrow[r, dotted, two heads] & f(b)
\end{tikzcd}
\quad \Longrightarrow \quad
\begin{tikzcd}
 & a \arrow[dl] \arrow[dr] & \\
b \arrow[dr, dotted, two heads] & & c \arrow[dl, dotted, two heads] \\
 & d &
\end{tikzcd}
$$

Theorem *Zprop_implies_Confl* $\{A{:}\mathtt{Type}\}$: $\forall$ *R*: *Rel A*, *Zprop R* $\rightarrow$ *Confl R*.

D. Kesner.
A Theory of Explicit Substitutions with Safe and Full Composition.
5(3:1):1–29, 2009.