Extending the Locally Nameless Representation with an Explicit Substitution Operator

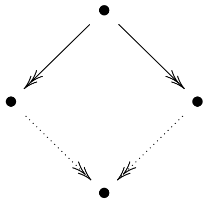
Flávio L. C. de Moura and Leandro Oliveira Rezende

Seminário GTC/UnB

February 8, 2020

Confluence of ARS

- An Abstract Rewriting System (ARS) is a pair (A, R) where A is a set and R is a binary relation over A.



The Z Property implies Confluence

- **Z Property**: Let $(A, \to)$ be an ARS. If $a \to b$ then there exists a mapping $f : A \to A$ such that the following diagram holds:

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

- Z-property implies confluence [vO07]: We showed the formalization of this result in the previous GTC seminar.

Weak Z Property

Definition

Let $(A, \to)$ be an ARS and $\to_x$ another relation on A. A mapping f satisfies the *weak Z property* for $\to$ by $\to_x$ if $a \to b$ implies $b \twoheadrightarrow_x f(a)$ and $f(a) \twoheadrightarrow_x f(b)$.

$$\begin{tikzcd} a \arrow[r] & b \arrow[dl, "x"] \\ f(a) \arrow[r, twoheadrightarrow, "x"'] & f(b) \end{tikzcd}$$

Compositional Z Property

Definition

Let $(A, \to)$ be an ARS such that $\to = \to_1 \cup \to_2$. If there exists mappings $f_1, f_2 : A \to A$ such that:

1. f_1 is Z for $\to_1$
2. $a \to_1 b$ implies $f_2(a) \twoheadrightarrow f_2(b)$
3. $a \twoheadrightarrow f_2(a)$, for any $a \in Im(f_1)$
4. $f_2 \circ f_1$ is weakly Z for $\to_2$ by $\to$

then $f_2 \circ f_1$ is Z for $(A, \to)$, and hence $(A, \to)$ is confluent.

$$\begin{tikzcd}
a \arrow[rr, "1"] & & b \arrow[dll, dotted, two heads, "1"'] \\
f_1(a) \arrow[rr, dotted, two heads, "1"] \arrow[d, dotted, two heads] & & f_1(b) \\
f_2(f_1(a)) \arrow[rr, dotted, two heads] & & f_2(f_1(b))
\end{tikzcd}
\qquad
\begin{tikzcd}
a \arrow[rr, "2"] & & b \arrow[dll, dotted, two heads] \\
f_2(f_1(a)) \arrow[rr, dotted, two heads] & & f_2(f_1(b))
\end{tikzcd}$$

- Compositional Z implies Confluence [NF16]
  - We formalized this result in Coq (Coq session).

The λ-calculus (with Explicit Substitutions)

- Starting point: λ-calculus

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

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

The λ-calculus (with 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

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

The λ-calculus (with 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

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

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

The λ-calculus (with 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

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

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

The Formalization in Coq

- Bound variables are De Bruijn indexes, and
- Free variables are named variables.
  - This framework was built in Coq for λ-calculi without explicit substitutions by Charguéraud [Cha11].

```
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.
```

- The expressions generated by this grammar are called pre-terms.
- But just a proper subset of the pre-terms are important: terms.
- We formalized three different notions of terms (with an explicit substitution operator) and their equivalence.

The notion of terms

```
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).
```

- where (*t1* ^ *x*) is obtained from *t1* by replacing all its occurrences of the index 0 for *x*, *x* being a free variable.
- (*t1* ^ *x*) is a particular case of {*k* ~> *u*} *t1*, is obtained from *t1* by replacing all its occurrences of the index *k* for *u*.

Alternative notion of term

- The local closure of an expression indicates the value of the indices that may appear in it.

```
Fixpoint lc_at (k:nat) (t:pterm) : Prop :=
  match t with
  | pterm_bvar i ⇒ i < k
  | pterm_fvar x ⇒ True
  | pterm_app t1 t2 ⇒ lc_at k t1 ∧ lc_at k t2
  | pterm_abs t1 ⇒ lc_at (S k) t1
  | pterm_sub t1 t2 ⇒ (lc_at (S k) t1) ∧ lc_at k t2
  end.
Theorem term_equiv_lc_at: ∀ t, term t ↔ lc_at 0 t.
```

Future work

- Complete the proof that the formalized calculus satisfies the Z property.
- Merge this formalization with the one that has the other properties (PSN and one-step β-simulation).
- Extract the code of the corresponding calculus with explicit substitutions.

A. Charguéraud.
The Locally Nameless Representation.
Journal of Automated Reasoning, pages 1–46, 2011.

Koji Nakazawa and Ken-etsu Fujita.
Compositional Z: confluence proofs for permutative conversion.
104(6):1205–1224, 2016.

Vincent van Oostrom.
Z - draft: For your mind only.
2007.