Substitutions for λ-free higher-order terms

Vincent Trélat

March 17, 2025

Abstract

This theory provides a formalization of substitutions on λ-free higher-order terms, establishing a structured framework with the expected algebraic properties. It introduces a type construction for the rigorous definition and manipulation of substitutions. The main theorem of this theory proves the existence of fixed-point substitutions under acyclicity, a theorem that is too often regarded as trivial in the literature [1, 3].

Contents

1 Introduction

This theory is based on J. Blanchette's Formalization of Recursive Path Orders for Lambda-Free Higher-Order Terms [2] which defines λ-free higher-order terms.

1.1 Preliminary lemmas

The following lemma and definitions would be worth adding in the theory *Lambda-Free-RPOs.Lambda-Free-Term*.

lemma *sub-trans*: ‹*sub x y* ⟹ *sub y z* ⟹ *sub x z*›
  ⟨*proof*⟩

definition *subterms* :: ‹(*'s*, *'v*) *tm* ⇒ (*'s*, *'v*) *tm set*› **where**
  ‹*subterms t* ≡ {*u*. *sub u t*}›

definition *proper-subterms* :: ‹(*'s*, *'v*) *tm* ⇒ (*'s*, *'v*) *tm set*› **where**
  ‹*proper-subterms t* ≡ {*u*. *proper-sub u t*}›

The following lemmas are also helpful in the following and could be easily lifted higher in the hierarchy of theories.

lemmas *mult-Suc-left* =
  mult-Suc-right[*unfolded add.commute*[*of m* ‹*m*∗*n*› **for** *m n*]]
— Although this result is immediate, it might be worth adding it to Nat symmetrically.

lemma *inject-nat-in-fset-ninj*:
  ‹*finite S* ⟹ (*range* (*f*::*nat*⇒-) ⊆ *S*) ⟹ (∃ *x y*. *x* ≠ *y* ∧ *f x* = *f y*)›
  ⟨*proof*⟩

lemma *wfPD*: ‹*wfP P* ⟹ *wfp-on P A*›
— This destruction rule for *wfP* could be added to the theory *Open-Induction.Restricted-Predicates*
  ⟨*proof*⟩

lemma *set-decr-chain-empty*:
  fixes *u* :: ‹*nat* ⇒ *'a set*›
  assumes *pord*: ‹⋀*n*. *u n* ≠ ∅ ⟹ *u* (*n*+*1*) ⊂ *u n*›
    and *fin*: ‹⋀*n*. *finite* (*u n*)›
  shows ‹∃ *k*. *u k* = ∅›
— This lemma could easily be generalized to any partial order and any minimal element and integrated to the theory *Well-Quasi-Orders.Minimal-Elements*.
⟨*proof*⟩

lemma *distinct-in-fset*:
  ‹*finite E* ⟹ *card E* = *n* ⟹ *distinct xs* ⟹ *set xs* ⊆ *E* ⟹ *length xs* ≤ *n*›
  ⟨*proof*⟩

2 Substitutions

This section embeds substitutions in a proper type, lifting basic operations like substitution application (i.e. *substitution* as an operation on terms) and composition.

2.1 Substitutions for terms

Substitutions in *Lambda-Free-RPOs.Lambda-Free-RPOs* [2] are not defined as a type, they are implicitly used as functions from variables to terms. However, not all functions from variables to terms are substitutions, which motivates the introduction of a proper type *subst* fitting the specification of a substitution, namely that only finitely many variables are not mapped to themselves.

abbreviation $\mathcal{V}$ **where** ‹$\mathcal{V} \equiv$ *Hd* $\circ$ *Var*›

lemma *inj-*$\mathcal{V}$: ‹*inj* $\mathcal{V}$›
  ⟨*proof*⟩

lemma *fin-var-restr*:‹*finite* ($\mathcal{V}$ ‘ E) $\Longrightarrow$ *finite* E›
  ⟨*proof*⟩

definition *is-subst* :: ‹($'v \Rightarrow$ ($'s$, $'v$) *tm*) $\Rightarrow$ *bool*› **where**
  ‹*is-subst* $\sigma \equiv$ *finite* {t. *is-Hd* t $\wedge$ *is-Var* (*head* t) $\wedge$ *subst* σ $t \neq t$}›

If type-checking on terms was enforced in *is-subst*, the above definition could be expressed as follows in a more concise way:
is-subst $\sigma \equiv$ *finite* {*subst* σ $t \neq t$}
Without type-checking, the definition must range over variables and not over terms since *App* x x is a valid term, even though it does not type check. If $x\sigma = x$, then $(x(x))\sigma = x(x)$. This inductively allows infinitely many fixpoints of the substitution σ.
With type-checking, the second definition would only add finitely many terms, namely type-correct applied terms of the form *App* y x where x and y are substitutable variables.

lemma *subst-*$\mathcal{V}$: ‹*is-subst* $\mathcal{V}$›
⟨*proof*⟩

typedef ($'s$, $'v$) *subst* = ‹{σ::($'v \Rightarrow$ ($'s$, $'v$) *tm*). *is-subst* σ}›
  ⟨*proof*⟩

setup-lifting *type-definition-subst*

lift-definition $\mathcal{V}'$ :: ‹($'s$, $'v$) *subst*› **is** ‹$\mathcal{V}$›
  ⟨*proof*⟩

Informally, $\mathcal{V}$ is almost the identity function since it casts variables to themselves (as terms), but it has the type $'v \Rightarrow ('s,\ 'v)\ tm$. $\mathcal{V}$ is thus lifted to $\mathcal{V}'$ that applies on substitutions. The fact that $\mathcal{V}'$ leaves ground terms unchanged follows from the definition of *subst* and is obtained by lifting. $\mathcal{V}'$ *is* the identity substitution.

lift-definition
  $subst\text{-}app :: \langle ('s,\ 'v)\ tm \Rightarrow ('s,\ 'v)\ subst \Rightarrow ('s,\ 'v)\ tm\rangle\ (\langle \text{-}\cdot\text{-}\rangle\ [56,55]\ 55)$ **is**
  $\langle \lambda\ x\ \sigma.\ subst\ \sigma\ x\rangle\langle proof\rangle$

lemma $sub\text{-}subst'$: $\langle sub\ x\ t \Longrightarrow sub\ (x\cdot\sigma)\ (t\cdot\sigma)\rangle$
— This lemma is a lifted version of $sub\ ?s\ ?t \Longrightarrow sub\ (subst\ ?\varrho\ ?s)\ (subst\ ?\varrho\ ?t)$.
  $\langle proof\rangle$

Application for substitutions (i.e. *substitution*) is lifted from the function *subst* and denoted as usual in the literature with a post-fix notation: *subst* $\sigma\ x$ is denoted by $x\cdot\sigma$.

lemma $subst\text{-}alt\text{-}def$:$\langle finite\ \{t.\ (\mathcal{V}\ t)\cdot\sigma \neq \mathcal{V}\ t\}\rangle$
$\langle proof\rangle$

The lemma above provides an alternative definition for substitution. Yet, there is a subtlety since Isabelle does not provide support for dependent types: one shall understand this lemma as the meta-implication "if σ is of type *subst* then the aforementioned set is finite". A true alternative definition should state an equivalence, however the converse implication makes no sense in Isabelle.

lemma $subst\text{-}eq\text{-}sub$:$\langle sub\ s\ t \Longrightarrow t\cdot\sigma = t\cdot\vartheta \Longrightarrow s\cdot\sigma = s\cdot\vartheta\rangle$
  $\langle proof\rangle$

Composition for substitutions is also lifted as follows.

lift-definition
  $rcomp :: \langle ('s,\ 'v)\ subst \Rightarrow ('s,\ 'v)\ subst \Rightarrow ('s,\ 'v)\ subst\rangle$
  (**infixl** $\langle\circ\rangle$ 55) **is** $\langle \lambda\ \sigma\ \vartheta.\ subst\ \vartheta \circ (subst\ \sigma \circ \mathcal{V})\rangle$
$\langle proof\rangle$

lemma $rcomp\text{-}subst\text{-}simp$:
  $\langle (x{::}('s,\ 'v)\ tm)\cdot(\sigma \circ \vartheta) = (x\cdot\sigma)\cdot\vartheta\rangle$
  $\langle proof\rangle$

lift-definition
  $set\text{-}image\text{-}subst :: \langle ('s,\ 'v)\ tm\ set \Rightarrow ('s,\ 'v)\ subst \Rightarrow ('s,\ 'v)\ tm\ set\rangle$
  (**infixl** $\langle\cdot\rangle$ 90) **is** $\langle \lambda\ S\ \sigma.\ subst\ \sigma\ `\ S\rangle\ \langle proof\rangle$

lemma $set\text{-}image\text{-}subst\text{-}collect$:
  $\langle S\cdot\sigma = \{x\cdot\sigma \mid x.\ x \in S\}\rangle$
  $\langle proof\rangle$

2.2 Substitutions as a monoid

First, we state two introduction lemmas for allowing extensional reasoning on substitutions. The first one is on terms and the second one is for terms that are variables.

lemma *subst-ext-tmI*:
  fixes σ::‹$('s,\ 'v)\ \mathit{subst}$› **and** ϑ::‹$('s,\ 'v)\ \mathit{subst}$›
  shows ‹$\forall\ (x::('s,\ 'v)\ \mathit{tm}).\ (x\cdot\sigma) = (x\cdot\vartheta) \Longrightarrow \sigma = \vartheta$›
⟨*proof*⟩

lemma *subst-ext-tmI′*:
  fixes σ::‹$('s,\ 'v)\ \mathit{subst}$› **and** ϑ::‹$('s,\ 'v)\ \mathit{subst}$›
  shows ‹$\forall\ x.\ (\mathcal{V}\ x)\cdot\sigma = (\mathcal{V}\ x)\cdot\vartheta \Longrightarrow \sigma = \vartheta$›
  ⟨*proof*⟩

lemmas *subst-extI* = *subst-ext-tmI subst-ext-tmI′*

The three following lemmas state that $\mathcal{V}'$ is the neutral element for composition. Although uniqueness follows from the definition of a neutral element, the proof of this claim is given below.

lemma $\mathcal{V}'$*-id-tm* [*simp*]:
  fixes x::‹$(\text{-},\text{-})\ \mathit{tm}$›
  shows ‹$(x\cdot\mathcal{V}') = x$›
  ⟨*proof*⟩

lemma $\mathcal{V}'$*-neutral-rcomp*[*simp*]:
  ‹$\sigma \circ \mathcal{V}' = \sigma$›
  ‹$\mathcal{V}' \circ \sigma = \sigma$›
  ⟨*proof*⟩

lemma *unique-*$\mathcal{V}'$:
  ‹$(\bigwedge\sigma.\ \sigma \circ \eta = \sigma) \Longrightarrow \eta = \mathcal{V}'$›
  ‹$(\bigwedge\sigma.\ \eta \circ \sigma = \sigma) \Longrightarrow \eta = \mathcal{V}'$›
⟨*proof*⟩

lemma $\mathcal{V}'$*-iff*:‹$\sigma = \mathcal{V}' \longleftrightarrow (\forall\ x.\ (\mathcal{V}\ x)\cdot\sigma = (\mathcal{V}\ x))$›
  ⟨*proof*⟩

lemma *rcomp-assoc*[*simp*]:
  fixes σ::‹$('s,\ 'v)\ \mathit{subst}$›
    and ϑ::‹$('s,\ 'v)\ \mathit{subst}$›
    and γ::‹$('s,\ 'v)\ \mathit{subst}$›
  shows ‹$(\sigma \circ \vartheta) \circ \gamma = \sigma \circ (\vartheta \circ \gamma)$›
  ⟨*proof*⟩

Knowing that the composition of substitutions ($\circ$) is associative and has a neutral element $\mathcal{V}'$, we may embed substitutions in an algebraic structure with a monoid structure and enjoy Isabelle's lemmas on monoids.

global-interpretation *subst-monoid*: *monoid rcomp* $\mathcal{V}'$
  ⟨*proof*⟩

3 Acyclic substitutions

3.1 Definitions and auxiliary lemmas

The iteration on substitutions is defined below and is followed by several algebraic properties.

In order to show these properties, we give three different definitions for iterated substitutions. In short, the first one is simply the iteration of composition using Isabelle's ($\frown\frown$) operator. This can be understood as follows:

$$\sigma^n \triangleq (\underbrace{\sigma \circ (\sigma \circ (\ldots (\sigma}_{n\text{ times}} \circ \mathcal{V}') \ldots))$$

Using properties from the monoid structure, this can be written as

$$\sigma^n \triangleq \underbrace{\sigma \circ \cdots \circ \sigma}_{n\text{ times}}$$

The two other definitions are inductively defined using those two schemes:

$$\sigma^{n+1} = \sigma \circ \sigma^n$$
$$\sigma^{n+1} = \sigma^n \circ \sigma$$

We prove that these three definitions are equivalent and use them in the proofs of the properties that follow.

definition *iter-rcomp* :: ‹(*'s*, *'v*) *subst* ⇒ *nat* ⇒ (*'s*, *'v*) *subst*›
  (‹-⁻› [*200*, *0*] *1000*) **where** ‹$\sigma^n \equiv ((\circ)\ \sigma\ \frown\frown\ n)\ \mathcal{V}'$›

lemma *iter-rcomp-Suc-right*:‹$\sigma^{Suc\ n} = \sigma^n \circ \sigma$›
  ⟨*proof*⟩

lemma *iter-rcomp-Suc-left*:‹$\sigma^{Suc\ n} = \sigma \circ \sigma^n$›
  ⟨*proof*⟩

fun *iter-rcomp'* :: ‹(*'s*, *'v*) *subst* ⇒ *nat* ⇒ (*'s*, *'v*) *subst*›
  where
  ‹*iter-rcomp'* σ *0* = $\mathcal{V}'$›
| ‹*iter-rcomp'* σ (*Suc n*) = σ ∘ (*iter-rcomp'* σ *n*)›
lemma *iter-rcomp-eq-iter-rcomp'*:‹σ^n = *iter-rcomp'* σ *n*›
  ⟨*proof*⟩

fun *iter-rcomp''* :: ‹(*'s*, *'v*) *subst* ⇒ *nat* ⇒ (*'s*, *'v*) *subst*›

```
  where
  ‹iter-rcomp'' σ 0 = 𝒱'›
| ‹iter-rcomp'' σ (Suc n) = (iter-rcomp'' σ n) ∘ σ›

lemma iter-rcomp-eq-iter-rcomp'':‹σ^n = iter-rcomp'' σ n›
  ⟨proof⟩

lemmas iter-rcomp'-eq-iter-rcomp'' =
  iter-rcomp-eq-iter-rcomp'[symmetric, simplified iter-rcomp-eq-iter-rcomp'']
```

The following lemmas show some algebraic properties on iterations of substitutions, namely that for any σ, the function $n \mapsto \sigma^n$ i.e. *iter-rcomp* σ is a magma homomorphism between $(\mathbb{N}, +)$ and $(\mathit{subst}, \circ)$. Since $\sigma^{0} \equiv \mathcal{V}'$, it is even a (commutative) monoid homomorphism.

```
lemma iter-comp-add-morphism: ‹(σ^n) ∘ (σ^k) = σ^(n+k)›
  ⟨proof⟩

lemmas iter-comp-com-add-morphism =
  iter-comp-add-morphism[
    of σ n k for σ n k,
    simplified add.commute,
    unfolded iter-comp-add-morphism[of σ k n, symmetric]]
```

There is a similar property with multiplication, stated as follows:

$$\forall \sigma, n, k. (\sigma^n)^k = \sigma^{n \times k}$$

This is shown by the following lemma. The next one shows commutativity.

```
lemma iter-comp-mult-morphism: ‹(σ^n)^k = σ^(n*k)›
  ⟨proof⟩

lemmas iter-comp-com-mult-morphism =
  iter-comp-mult-morphism[
    of σ n k for σ k n,
    simplified mult.commute,
    unfolded iter-comp-mult-morphism[of σ k n, symmetric]]
```

Some simplification rules are added to the rules to help automatize subsequent proofs.

```
lemma iter-rcomp-𝒱'[simp]: ‹𝒱'^m = 𝒱'›
  ⟨proof⟩

lemma iter-rcomp-0[simp]: ‹σ^0 = 𝒱'›
  ⟨proof⟩

lemma iter-rcomp-1[simp]: ‹σ^(Suc 0) = σ›
  ⟨proof⟩
```

definition *dom* :: ‹$('s, 'v)$ *subst* $\Rightarrow$ $('s, 'v)$ *tm set*› **where**
‹*dom* $\sigma \equiv \{\mathcal{V}\ x \mid x.\ (\mathcal{V}\ x) \cdot \sigma \neq \mathcal{V}\ x\}$›

definition *ran* :: ‹$('s, 'v)$ *subst* $\Rightarrow$ $('s, 'v)$ *tm set*› **where**
‹*ran* $\sigma \equiv (\lambda x.\ x \cdot \sigma)$ ‘ *dom* σ›

lemma *no-sub-in-dom-subst-eq*: ‹$(\forall\ x \in$ *dom* $\sigma.\ \neg$ *sub* $x\ t) \Longrightarrow t = t \cdot \sigma$›
⟨*proof*⟩

lemma *subst-eq-on-domI*:
‹$(\forall x.\ x \in$ *dom* $\sigma \lor x \in$ *dom* $\vartheta \longrightarrow x \cdot \sigma = x \cdot \vartheta) \Longrightarrow \sigma = \vartheta$›
⟨*proof*⟩

lemma *subst-finite-dom*:‹*finite* (*dom* σ)›
⟨*proof*⟩

lemma $\mathcal{V}'$*-emp-dom*: ‹*dom* $\mathcal{V}' = \emptyset$›
⟨*proof*⟩

lemma *var-not-in-dom* [*simp*]: ‹$\mathcal{V}\ x \notin$ *dom* $\sigma \Longrightarrow ((\mathcal{V}\ x) \cdot \sigma^n) = \mathcal{V}\ x$›
⟨*proof*⟩

lemma *ran-alt-def*:‹*ran* $\sigma = \{(\mathcal{V}\ x) \cdot \sigma \mid x.\ (\mathcal{V}\ x) \cdot \sigma \neq \mathcal{V}\ x\}$›
⟨*proof*⟩

definition *is-ground-subst* :: ‹$('s, 'v)$ *subst* $\Rightarrow$ *bool*› **where**
‹*is-ground-subst* $\sigma \equiv$ (*ground* ‘ *ran* σ) $= \{$*True*$\}$›

lemma *is-ground-subst-alt-def*:
‹*is-ground-subst* $\sigma \longleftrightarrow$ (*ran* $\sigma \neq \emptyset) \land (\forall x.\ (\mathcal{V}\ x) \cdot \sigma \neq \mathcal{V}\ x \longrightarrow$ *ground* $((\mathcal{V}\ x) \cdot \sigma))$›
⟨*proof*⟩

lemma *ground-subst-grounds*:‹*is-ground-subst* $\sigma \Longrightarrow x \in$ *dom* $\sigma \Longrightarrow$ *ground* $(x \cdot \sigma)$›
⟨*proof*⟩

lemma *iter-on-ground*:‹*ground* $(x \cdot \sigma) \Longrightarrow n > 0 \Longrightarrow x \cdot \sigma^n = x \cdot \sigma$›
⟨*proof*⟩

lemma *true-subst-nempty-vars*:
‹$\sigma \neq \mathcal{V} \Longrightarrow \{t.$ *is-Hd* $t \land$ *is-Var* (*head* t) $\land$ *subst* $\sigma\ t \neq t\} \neq \{\}$›
⟨*proof*⟩

lemma *true-subst-nemp-im*:‹*ran* $\sigma = \{\} \Longrightarrow \sigma = \mathcal{V}'$›
⟨*proof*⟩

lemma *ground-subst-imp-no-var-mapped-on-var*:
‹*is-ground-subst* $\sigma \Longrightarrow (\forall x\ y.\ x \neq y \longrightarrow (\mathcal{V}\ x) \cdot \sigma \neq (\mathcal{V}\ y))$›
⟨*proof*⟩

lemma *ran-$\mathcal{V}'$-empty*:‹*ran* $\mathcal{V}' = \emptyset$›
  ⟨*proof*⟩

lemma *non-ground-$\mathcal{V}'$*: ‹$\neg$ *is-ground-subst* $\mathcal{V}'$›
  ⟨*proof*⟩

3.2 Acyclicity

A substitution is said to be *acyclic* if no variable x in the domain of σ occurs as a subterm of $x{\cdot}\sigma^n$ for any $0 < n$.

definition *is-acyclic* :: ‹$('s,\ 'v)$ *subst* $\Rightarrow$ *bool*› **where**
  ‹*is-acyclic* $\sigma \equiv (\forall\ x \in \mathit{dom}\ \sigma.\ \forall\ n > 0.\ x \notin \mathit{subterms}\ (x{\cdot}\sigma^n))$›

lemma *is-acyclicE*:‹ *is-acyclic* $\sigma \Longrightarrow x \in \mathit{dom}\ \sigma \Longrightarrow n > 0 \Longrightarrow x \notin \mathit{subterms}$ $(x{\cdot}\sigma^n)$›
  ⟨*proof*⟩

lemma *non-acyclic-$\mathcal{V}'$*: ‹*is-acyclic* $\mathcal{V}'$›
  ⟨*proof*⟩

lemma *acyclic-iter-dom-eq*:‹*is-acyclic* $\sigma \Longrightarrow \mathit{dom}\ \sigma = \mathit{dom}\ \sigma^n$› **if** ‹$n > 0$› **for** n
    ⟨*proof*⟩

lemma *acyclic-iter*: ‹*is-acyclic* $\sigma \Longrightarrow n > 0 \Longrightarrow$ *is-acyclic* σ^n›
  ⟨*proof*⟩

3.3 Fixed-point substitution

We define the fixed-point substitution of a substitution σ as the substitution σ^i where $i = \inf\{k \in \mathbb{N} \mid \sigma^k = \sigma^{k+1}\}$.

definition *fp-subst* :: ‹$('s,\ 'v)$ *subst* $\Rightarrow$ $('s,\ 'v)$ *subst*› (‹-*› *1000*) **where**
  ‹$\sigma^\star \equiv$ *iter-rcomp* σ (*LEAST* n . $\sigma^n = \sigma^{n+1}$)›

lemma *ground-subst-is-fp*:‹*is-ground-subst* $\sigma \Longrightarrow \sigma^\star = \sigma$›
— Ground substitutions have no effect and are therefore fixed-points substitutions. The converse is not true.
⟨*proof*⟩

In the following, we prove that fixed-point substitutions are well-defined for acyclic substitutions. To help visualise how the proofs are carried out, for any terms x and y and any substitution σ, we denote the fact that x is substituted by y after one application of σ, i.e. that *sub* y $(x{\cdot}\sigma)$, by $x \rightarrow_\sigma y$.

Remark. *In fact, automata could be used to model substitutions with variables in the domain as the initial states and variables outside of the domain and constants as final states. The transitions would be given by the successive substitutions. Acyclic substitutions would be represented by a DAG.*

lemma *dom-sub-subst*: ‹$x \in dom\ \sigma \Longrightarrow sub\ x\ (t \cdot \sigma) \Longrightarrow \exists\, y \in dom\ \sigma.\ sub\ x\ (y \cdot \sigma)$›
— If $x \rightarrow_\sigma t$ for $x \in dom\ \sigma$ and a term t, then there is a variable $y \in dom\ \sigma$ such that $x \rightarrow_\sigma y$.
⟨*proof*⟩

lemma *dom-sub-subst-contrapos*:
— For x in the domain, if there is no z in the domain such that $x \rightarrow_\sigma z$, then there is not term t such that $x \rightarrow_\sigma t$.
‹$x \in dom\ \sigma \Longrightarrow \forall\, z \in dom\ \sigma.\ \neg\ sub\ x\ (z \cdot \sigma) \Longrightarrow \forall\, t.\ \neg\ sub\ x\ (t \cdot \sigma)$›
⟨*proof*⟩

lemma *dom-sub-subst-iter*:
‹$x \in dom\ \sigma \Longrightarrow \forall\, z \in dom\ \sigma.\ \neg\ sub\ x\ (z \cdot \sigma^n) \Longrightarrow \neg\ sub\ x\ (t \cdot \sigma^n)$›
⟨*proof*⟩

lemma
assumes ‹$x \in dom\ \sigma$› ‹$\forall\, y \in dom\ \sigma.\ sub\ y\ t \longrightarrow \neg\ sub\ x\ (y \cdot \sigma)$›
shows *not-sub-subst-if*:‹$\neg\ sub\ x\ (t \cdot \sigma)$›
— For x in the domain and any term t, if there is no variable y occurring in t such that $x \rightarrow_\sigma y$, then $x \not\rightarrow_\sigma t$.
⟨*proof*⟩

lemma *dom-sub-subst-iter-Suc*:
‹$x \in dom\ \sigma \Longrightarrow sub\ x\ (t \cdot \sigma^{n+1}) \Longrightarrow$
$\exists\, y\ z.\ y \in dom\ \sigma \wedge z \in dom\ \sigma \wedge sub\ x\ (z \cdot \sigma) \wedge sub\ z\ (y \cdot \sigma^n)$›
— If $x \rightarrow_\sigma^{n+1} t$, then there are variables y and z such that $y \rightarrow_\sigma^n z \rightarrow_\sigma x$.
⟨*proof*⟩

lemma *sub-Suc-n-sub-n-sub*:
‹$(\exists\ x \in dom\ \sigma.\ sub\ z\ (x \cdot \sigma^{n+1})) \longleftrightarrow$
$(\exists\ x\ y.\ x \in dom\ \sigma \wedge y \in dom\ \sigma \wedge sub\ z\ (y \cdot \sigma) \wedge sub\ y\ (x \cdot \sigma^n))$› **if** ‹$z \in dom\ \sigma$›
⟨*proof*⟩

The following theorem is the main result of this theory and states that for acyclic substitutions, the fixed-point substitution exists and is well defined. The main idea of the proof is to define a non-negative quantity and show that successively applying the substitution makes it decrease.

For any such iteration n, we define the set of variables that will be substituted by the next iteration of the substitution and denote it by S_n. Formally, S_n is defined as follows:

$$S_n := \{z \in \text{dom}\ \sigma \mid \exists x \in \text{dom}\ \sigma.\ x \rightarrow_\sigma^n z\}$$

There is a clear recurrence relation between S_{n+1} and S_n, namely that the variables in S_{n+1} are exactly the variables in S_n that are not sources in S_n, i.e. that have a predecessor – for the subterm relation – in S_n. This is formalized as follows:

$$S_{n+1} = S_n - \{z \in S_n \mid \forall x \in S_n.\ x \not\to_\sigma z\}$$

This implies that the sequence $(S_n)_{n\in\mathbb{N}}$ is strictly monotone for inclusion. Since it is bounded and has its values in finite sets, it is convergent and there is a rank k from which it is constant and equal to the infimum of the range, the empty set.

theorem *fp-subst*: ‹*is-acyclic* $\sigma \Longrightarrow \exists\, n.\ \sigma^n = \sigma^{n+1}$›
⟨*proof*⟩

lemma *fp-subst-comp-stable*: ‹*is-acyclic* $\sigma \Longrightarrow (\sigma^\star) \circ (\sigma^\star) = \sigma^\star$›
⟨*proof*⟩

lemma *fp-subst-stable-iter*: ‹*is-acyclic* $\sigma \Longrightarrow n > 0 \Longrightarrow (\sigma^\star)^n = \sigma^\star$›
⟨*proof*⟩

lemma *fp-subst-stable-fp*: ‹*is-acyclic* $\sigma \Longrightarrow (\sigma^\star)^\star = \sigma^\star$›
⟨*proof*⟩

end

References

[1] H. Barbosa, P. Fontaine, and A. Reynolds. Congruence closure with free variables. In A. Legay and T. Margaria, editors, *TACAS 2017*, volume 10206 of *LNCS*, pages 214–230, 2017.

[2] J. C. Blanchette, U. Waldmann, and D. Wand. Formalization of recursive path orders for lambda-free higher-order terms. *Archive of Formal Proofs*, September 2016. https://isa-afp.org/entries/Lambda_Free_RPOs.html, Formal proof development.

[3] S. Tourret, P. Fontaine, D. E. Ouraoui, and H. Barbosa. Lifting congruence closure with free variables to λ-free higher-order logic via SAT encoding. In F. Bobot and T. Weber, editors, *SMT 2020*, volume 2854 of *CEUR Workshop Proceedings*, pages 3–14. CEUR-WS.org, 2020.