# Strong Normalization of Moggis's Computational Metalanguage 

Christian Doczkal<br>Saarland University

September 13, 2023


#### Abstract

Handling variable binding is one of the main difficulties in formal proofs. In this context, Moggi's computational metalanguage serves as an interesting case study. It features monadic types and a commuting conversion rule that rearranges the binding structure. Lindley and Stark have given an elegant proof of strong normalization for this calculus. The key construction in their proof is a notion of relational TT-lifting, using stacks of elimination contexts to obtain a Girard-Tait style logical relation.

I give a formalization of their proof in Isabelle/HOL-Nominal with a particular emphasis on the treatment of bound variables.


## Contents

1 Introduction ..... 1
2 The Calculus ..... 2
3 The Reduction Relation ..... 5
4 Stacks ..... 9
5 Reducibility for Terms and Stacks ..... 12
6 Properties of the Reducibility Relation ..... 13
7 Abstraction Preserves Reducibility ..... 18
8 Sequencing Preserves Reducibility ..... 19
8.1 Central lemma ..... 24
9 Fundamental Theorem ..... 27
9.1 Strong normalization theorem ..... 28

## 1 Introduction

This article contains a formalization of the strong normalization theorem for the $\lambda_{m l}$-calculus. The formalization is based on a proof by Lindley and Stark [LS05]. An informal description of the formalization can be found in [DS09]. This formalization extends the example proof of strong normalization for the simply-typed $\lambda$-calculus, which is included in the Isabelle distribution [Nom].

The parts of the original proof which have been left unchanged are not displayed in this document.
The next section deals with the formalization of syntax, typing, and substitution. Section 3 contains the formalization of the reduction relation. Section 4 defines stacks which are used to define the reducibility relation in Section 5. The following sections contain proofs about the reducibility relation, ending with the normalization theorem in Section 9.

## 2 The Calculus

```
atom-decl name
nominal-datatype trm=
        Var name
    | App trm trm
    |am《name»trm (\Lambda - - [0,120] 120)
    | To trm《name»trm(- to - in - [141,0,140] 140)
    | Ret trm ([-])
declare trm.inject[simp]
lemmas name-swap-bij = pt-swap-bij'\[OF pt-name-inst at-name-inst]
lemmas ex-fresh = exists-fresh'[OF fin-supp]
lemma alpha" :
    fixes }x\mathrm{ y :: name and t::trm
    assumes a: x\sharpt
    shows [y].t = [x].([(y,x)] • t)
proof -
    from a have aux: y#[(y,x)] •t
        by (subst fresh-bij[THEN sym, of - - [(x,y)]])
            (auto simp add: perm-swap calc-atm)
    thus ?thesis
        by(auto simp add: alpha perm-swap name-swap-bij fresh-bij)
qed
```

Even though our types do not involve binders, we still need to formalize them as nominal datatypes to obtain a permutation action. This is required to establish equivariance of the typing relation.

```
nominal-datatype ty =
```

        TBase
    | TFun ty ty (infix \(\rightarrow\) 200)
    | T ty
    Since we cannot use typed variables, we have to formalize typing contexts. Typing contexts are formalized as lists. A context is valid if no name occurs twice.

```
inductive
    valid :: (name }\times\mathrm{ ty) list }=>\mathrm{ bool
where
    v1[intro]: valid []
|v2[intro]: \llbracketvalid \Gamma;x\sharp\Gamma\rrbracket\Longrightarrow valid ((x,\sigma)#\Gamma)
equivariance valid
```

lemma fresh-ty:
fixes $x::$ name and $\tau:: t y$
shows $x \sharp \tau$
by (induct $\tau$ rule: ty.induct) (auto)
lemma fresh-context:
fixes $\Gamma::(n a m e \times t y) l i s t$
assumes $a$ : $x \sharp \Gamma$
shows $\neg(\exists \tau .(x, \tau) \in$ set $\Gamma)$
using $a$
by (induct $\Gamma$ ) (auto simp add: fresh-prod fresh-list-cons fresh-atm)

## inductive

typing $::($ name $\times$ ty $)$ list $\Rightarrow$ trm $\Rightarrow$ ty $\Rightarrow$ bool $(-\vdash-:-[60,60,60] 60)$
where
$t 1[$ intro $]: \llbracket$ valid $\Gamma ;(x, \tau) \in$ set $\Gamma \rrbracket \Longrightarrow \Gamma \vdash \operatorname{Var} x: \tau$
| t2[intro]: $\llbracket \Gamma \vdash s: \tau \rightarrow \sigma ; \Gamma \vdash t: \tau \rrbracket \Longrightarrow \Gamma \vdash$ App $s t: \sigma$
| t3[intro]: $\llbracket x \sharp \Gamma ;((x, \tau) \# \Gamma) \vdash t: \sigma \rrbracket \Longrightarrow \Gamma \vdash \Lambda x . t: \tau \rightarrow \sigma$
| $\mathrm{t}_{4}[$ intro $]: \llbracket \Gamma \vdash s: \sigma \rrbracket \Longrightarrow \Gamma \vdash[s]: T \sigma$
$\mid t 5[$ intro $]: \llbracket x \sharp(\Gamma, s) ; \Gamma \vdash s: T \sigma ;((x, \sigma) \# \Gamma) \vdash t: T \tau \rrbracket$
$\Longrightarrow \Gamma \vdash s$ to $x$ in $t: T \tau$
equivariance typing
nominal-inductive typing
by (simp-all add: abs-fresh fresh-ty)
Except for the explicit requirement that contexts be valid in the variable case and the freshness requirements in $t 3$ and $t 5$, this typing relation is a direct translation of the original typing relation in [LS05] to Curry-style typing.
fun
lookup :: (name $\times$ trm) list $\Rightarrow$ name $\Rightarrow$ trm
where
lookup [] $x=\operatorname{Var} x$
| lookup $((y, e) \# \vartheta) x=($ if $x=y$ then e else lookup $\vartheta x)$
lemma lookup-eqvt[eqvt]:
fixes pi::name prm
and $\quad \vartheta::($ name $\times$ trm $) ~ l i s t$
and $x:$ :name
shows $p i \cdot($ lookup $\vartheta x)=\operatorname{lookup}(p i \cdot \vartheta)(p i \cdot x)$
by (induct $\vartheta$ ) (auto simp add: eqvts)
nominal-primrec
psubst $::($ name $\times$ trm $)$ list $\Rightarrow$ trm $\Rightarrow$ trm $(-<->[95,95]$ 205)
where
$\vartheta<\operatorname{Var} x>=$ lookup $\vartheta x$
$\mid \vartheta<A p p s t>=A p p(\vartheta<s>)(\vartheta<t>)$
$\mid x \sharp \vartheta \Longrightarrow \vartheta<\Lambda x . s>=\Lambda x .(\vartheta<s>)$
| $\vartheta<[t]>=[\vartheta<t>]$
$\mid \llbracket x \sharp \vartheta ; x \sharp t \rrbracket \Longrightarrow \vartheta<t$ to $x$ in $s>=(\vartheta<t>)$ to $x$ in $(\vartheta<s>)$
by(finite-guess,$+($ simp add: abs-fresh $)+$, fresh-guess + )
lemma psubst-eqvt[eqvt]:
fixes $p i::$ name $p r m$
shows $p i \cdot(\vartheta<t>)=(p i \cdot \vartheta)<(p i \cdot t)>$
by (nominal-induct t avoiding: $\vartheta$ rule:trm.strong-induct) (auto simp add: eqvts fresh-bij)

```
abbreviation
    subst \(:: \operatorname{trm} \Rightarrow\) name \(\Rightarrow \operatorname{trm} \Rightarrow \operatorname{trm}(-[-::=-][200,100,100]\) 200 \()\)
where
    \(t\left[x::=t^{\prime}\right] \equiv\left(\left[\left(x, t^{\prime}\right)\right]\right)<t>\)
lemma subst[simp]:
shows \((\operatorname{Var} x)[y::=v]=(\) if \(x=y\) then \(v\) else Var \(x)\)
    and \((A p p s t)[y::=v]=A p p(s[y::=v])(t[y::=v])\)
    and \(x \sharp(y, v) \Longrightarrow(\Lambda x . t)[y::=v]=\Lambda x . t[y::=v]\)
    and \(x \sharp(s, y, v) \Longrightarrow(s\) to \(x\) in \(t)[y::=v]=s[y::=v]\) to \(x\) in \(t[y::=v]\)
    and \(([s])[y::=v]=[s[y::=v]]\)
by (simp-all add: fresh-list-cons fresh-list-nil)
lemma subst-rename:
    assumes \(a\) : \(y \sharp t\)
    shows \(([(y, x)] \cdot t)[y::=v]=t[x::=v]\)
using \(a\)
by (nominal-induct t avoiding: x y v rule: trm.strong-induct)
    (auto simp add: calc-atm fresh-atm abs-fresh fresh-prod fresh-aux)
lemmas subst-rename \({ }^{\prime}=\) subst-rename \([\) THEN sym \(]\)
lemma forget: \(x \sharp t \Longrightarrow t[x::=v]=t\)
by(nominal-induct t avoiding: x v rule: trm.strong-induct)
    (auto simp add: abs-fresh fresh-atm)
lemma fresh-fact:
    fixes \(x\) ::name
    assumes \(x\) : \(x \sharp v \quad x \sharp t\)
    shows \(x \sharp t[y::=v]\)
using \(x\)
by (nominal-induct \(t\) avoiding: \(x\) y rule: trm.strong-induct)
        (auto simp add: abs-fresh fresh-atm)
lemma fresh-fact':
    fixes \(x\) ::name
    assumes \(x: x \sharp v\)
    shows \(x \sharp t[x::=v]\)
using \(x\)
by(nominal-induct t avoiding: x v rule: trm.strong-induct)
    (auto simp add: abs-fresh fresh-atm)
lemma subst-lemma:
    assumes \(a: x \neq y\)
    and \(\quad b: x \sharp u\)
    shows \(s[x::=v][y::=u]=s[y::=u][x::=v[y::=u]]\)
using \(a b\)
by(nominal-induct s avoiding: \(x\) y u v rule: trm.strong-induct)
        (auto simp add: fresh-fact forget)
lemma id-subs:
    shows \(t[x::=\) Var \(x]=t\)
by(nominal-induct t avoiding: x rule:trm.strong-induct) auto
```

In addition to the facts on simple substitution we also need some facts on parallel substitution. In particular we want to be able to extend a parallel substitution with a simple one.

```
lemma lookup-fresh:
    fixes \(z:: n a m e\)
    assumes \(z \sharp \vartheta \quad z \sharp x\)
    shows \(z \sharp\) lookup \(\vartheta x\)
using assms
by (induct rule: lookup.induct)
    (auto simp add: fresh-list-cons)
lemma lookup-fresh':
    assumes \(a\) : \(z \sharp \vartheta\)
    shows lookup \(\vartheta z=\operatorname{Var} z\)
using \(a\)
by (induct rule: lookup.induct)
    (auto simp add: fresh-list-cons fresh-prod fresh-atm)
lemma psubst-fresh-fact:
    fixes \(x\) :: name
    assumes \(a: x \sharp t\) and \(b: x \sharp \vartheta\)
    shows \(x \sharp \vartheta<t>\)
using \(a b\)
by (nominal-induct t avoiding: \(\vartheta\) x rule:trm.strong-induct)
        (auto simp add: lookup-fresh abs-fresh)
lemma psubst-subst:
    assumes \(a\) : \(x \sharp \vartheta\)
    shows \(\vartheta<t>[x::=s]=((x, s) \# \vartheta)<t>\)
    using \(a\)
by (nominal-induct \(t\) avoiding: \(\vartheta x\) s rule: trm.strong-induct)
    (auto simp add: fresh-list-cons fresh-atm forget
    lookup-fresh lookup-fresh' fresh-prod psubst-fresh-fact)
```


## 3 The Reduction Relation

With substitution in place, we can now define the reduction relation on $\lambda_{m l^{-}}$ terms. To derive strong induction and case rules, all the rules must be vccompatible (cf. [Urb08]). This requires some additional freshness conditions. Note that in this particular case the additional freshness conditions only serve the technical purpose of automatically deriving strong reasoning principles. To show that the version with freshness conditions defines the same relation as the one without the freshness conditions, we also state this version and prove equality of the two relations.
This requires quite some effort and is something that is certainly undesirable in nominal reasoning. Unfortunately, handling the reduction rule r10 which rearranges the binding structure, appeared to be impossible without going through this.

```
inductive std-reduction \(::\) trm \(\Rightarrow\) trm \(\Rightarrow\) bool \((-\rightsquigarrow-[80,80]\) 80)
where
    std-r1[intro!]:s \(\rightsquigarrow s^{\prime} \Longrightarrow A p p s t \rightsquigarrow A p p s^{\prime} t\)
\(\mid\) std-r2 [intro!]:t \(\rightsquigarrow t^{\prime} \Longrightarrow A p p s t \rightsquigarrow A p p s t^{\prime}\)
```

```
\(\mid\) std-r\}[intro!]:App \((\Lambda x . t) s \rightsquigarrow t[x::=s]\)
| std-r4[intro!]:t \(\rightsquigarrow t^{\prime} \Longrightarrow \Lambda x . t \rightsquigarrow \Lambda x \cdot t^{\prime}\)
| std-r \(5[\) intro! \(]: x \sharp t \Longrightarrow \Lambda x . A p p t(\) Var \(x) \rightsquigarrow t\)
\(\mid\) std-r6[intro!]: \(\llbracket s s^{\prime} \rrbracket \Longrightarrow s\) to \(x\) in \(t \rightsquigarrow s^{\prime}\) to \(x\) in \(t\)
| std-r \(\begin{array}{r}\text { r intro! }]: \llbracket ~ \\ \rightsquigarrow\end{array} t^{\prime} \rrbracket \Longrightarrow s\) to \(x\) in \(t \rightsquigarrow s\) to \(x\) in \(t^{\prime}\)
| std-r8[intro!]:[s] to \(x\) in \(t \rightsquigarrow t[x::=s]\)
| std-r9[intro!]: \(x \sharp s \Longrightarrow s\) to \(x\) in \([\) Var \(x] \rightsquigarrow s\)
| std-r10[intro!]: \(\llbracket x \sharp y ; x \sharp u \rrbracket\)
    \(\Longrightarrow(s\) to \(x\) in \(t)\) to \(y\) in \(u \rightsquigarrow s\) to \(x\) in (t to \(y\) in \(u)\)
\(\mid s t d-r 11\left[\right.\) intro!]: \(s \rightsquigarrow s^{\prime} \Longrightarrow[s] \rightsquigarrow\left[s{ }^{\prime}\right]\)
inductive
    reduction \(::\) trm \(\Rightarrow\) trm \(\Rightarrow\) bool \((-\mapsto-[80,80] 80)\)
where
    r1[intro!]:s \(\mapsto s^{\prime} \Longrightarrow A p p s t \mapsto A p p s^{\prime} t\)
\(\mid r 2\left[\right.\) intro!]: \(t \mapsto t^{\prime} \Longrightarrow A p p\) s \(t \mapsto A p p s t^{\prime}\)
\(\mid r 3[\) intro! \(]: x \sharp s \Longrightarrow A p p(\Lambda x . t) s \mapsto t[x::=s]\)
| \(r_{4}[\) intro! \(]: t \mapsto t^{\prime} \Longrightarrow \Lambda x . t \mapsto \Lambda x \cdot t^{\prime}\)
| \(r 5\) [intro! ]: \(x \sharp t \Longrightarrow \Lambda x\). App \(t(\) Var \(x) \mapsto t\)
\(\mid r 6[\) intro! \(]: \llbracket x \sharp\left(s, s^{\prime}\right) ; s \mapsto s^{\prime} \rrbracket \Longrightarrow s\) to \(x\) in \(t \mapsto s^{\prime}\) to \(x\) in \(t\)
| \(r\) 7 [intro! ! : \(\llbracket x \sharp s ; t \mapsto t^{\prime} \rrbracket \Longrightarrow s\) to \(x\) in \(t \mapsto s\) to \(x\) in \(t^{\prime}\)
\(\mid r 8[\) intro! \(]: x \sharp s \Longrightarrow[s]\) to \(x\) in \(t \mapsto t[x::=s]\)
\(\mid r 9[\) intro! \(]: x \sharp s \Longrightarrow s\) to \(x\) in \([\) Var \(x] \mapsto s\)
|r10[intro!]: 【x甘 \(x, s, u) ; y \sharp(s, t) \rrbracket\)
    \(\Longrightarrow(s\) to \(x\) in \(t)\) to \(y\) in \(u \mapsto s\) to \(x\) in ( \(t\) to \(y\) in \(u)\)
\(\mid r 11[\) intro! \(]: s \mapsto s^{\prime} \Longrightarrow[s] \mapsto\left[s^{\prime}\right]\)
equivariance reduction
nominal-inductive reduction
    by (auto simp add: abs-fresh fresh-fact' fresh-prod fresh-atm)
```

In order to show adequacy, the extra freshness conditions in the rules $r 3, r 6$, $r^{r}, r 8, r 9$, and $r 10$ need to be discharged.

```
lemma \(r 3^{\prime}[\) intro! \(]: A p p(\Lambda x . t) s \mapsto t[x::=s]\)
proof -
    obtain \(x^{\prime}:\) :name where \(s: x^{\prime} \sharp s\) and \(t: x^{\prime} \sharp t\)
    using ex-fresh \([\) of ( \(s, t)]\) by (auto simp add: fresh-prod)
    from \(t\) have \(A p p(\Lambda x \cdot t) s=\operatorname{App}\left(\Lambda x^{\prime} .\left(\left[\left(x, x^{\prime}\right)\right] \cdot t\right)\right) s\)
    by (simp add: alpha'")
    also from \(s\) have \(\ldots \mapsto\left(\left[\left(x, x^{\prime}\right)\right] \cdot t\right)\left[x^{\prime}::=s\right]\)..
    also have \(\ldots=t[x::=s]\) using \(t\)
    by (auto simp add: subst-rename') (metis perm-swap)
    finally show ?thesis.
qed
declare r3[rule del]
```

lemma $r 6^{\prime}[$ intro :
fixes $s::$ trm
assumes $r: s \mapsto s^{\prime}$
shows $s$ to $x$ in $t \mapsto s^{\prime}$ to $x$ in $t$
using assms

```
proof -
    obtain x'::name where s: }\mp@subsup{x}{}{\prime}\sharp(s,\mp@subsup{s}{}{\prime})\mathrm{ and }t:\mp@subsup{x}{}{\prime}\sharp
        using ex-fresh[of ( }s,\mp@subsup{s}{}{\prime},t)]\mathrm{ by (auto simp add: fresh-prod)
    from t have s to x in t=s to x' in ([(x,\mp@subsup{x}{}{\prime})] \cdot t)
        by (simp add: alpha'')
    also from s r have ...\mapsto 的 to 和 in ([(x, x')] • t) ..
    also from t have ... = s' to x in t
        by (simp add: alpha'/)
    finally show ?thesis.
qed
declare r6[rule del]
lemma r7'[intro]:
    fixes t:: trm
    assumes }t\mapsto\mp@subsup{t}{}{\prime
    shows s to x in t\mapstos to x in t'
using assms
proof -
    obtain x'::name where f: x'\sharpt 
        using ex-fresh[of (t,\mp@subsup{t}{}{\prime},s,x)] by(auto simp add:fresh-prod)
    hence a:s to x in t=s to \mp@subsup{x}{}{\prime}}\mathrm{ in ([(x,x')] - t)
        by (auto simp add: alpha'")
    from assms have ([(x,\mp@subsup{x}{}{\prime})]\cdott)\mapsto[(x,\mp@subsup{x}{}{\prime})]\cdot\mp@subsup{t}{}{\prime}
        by (simp add: eqvts)
    hence r:s to }\mp@subsup{x}{}{\prime}\mathrm{ in ([(x,x')] • t) }\mapsto\mathrm{ s to }\mp@subsup{x}{}{\prime}\mathrm{ in ([(x, (')] • t')
        using f by auto
    from f have s to x in t' =s to \mp@subsup{x}{}{\prime}}\mathrm{ in ([(x,x')] - t')
        by (auto simp add: alpha'')
    with a r show ?thesis by (simp del: trm.inject)
qed
declare r7[rule del]
lemma r8'[intro!]: [s] to x in t\mapstot[x::=s]
proof -
    obtain x'::name where s: }\mp@subsup{x}{}{\prime}\sharps\mathrm{ and }t:\mp@subsup{x}{}{\prime}\sharp
        using ex-fresh[of (s,t)] by (auto simp add: fresh-prod)
    from t have [s] to x in t=[s] to x' in ([(x,x')] - t)
        by (simp add: alpha')
    also from s have ...\mapsto([(x, x)] • t)[\mp@subsup{x}{}{\prime}::=s] ..
    also have ... = t[x::=s] using t
        by (auto simp add: subst-rename') (metis perm-swap)
    finally show ?thesis.
qed
declare r8[rule del]
lemmar r9'[intro!]: s to x in [Var x]\mapstos
proof -
    obtain x}\mp@subsup{x}{}{\prime}::name where f:\mp@subsup{x}{}{\prime}\sharps\quad\mp@subsup{x}{}{\prime}\sharp
        using ex-fresh[of (s,x)] by(auto simp add:fresh-prod)
    hence s to }\mp@subsup{x}{}{\prime}\mathrm{ in [Var x]}\mapstos\mathrm{ by auto
    moreover have s to }\mp@subsup{x}{}{\prime}\mathrm{ in ([Var x ])=s to x in ([Var x])
        by (auto simp add: alpha fresh-atm swap-simps)
    ultimately show ?thesis by simp
qed
declare rg[rule del]
```

While discharging these freshness conditions is easy for rules involving only one binder it unfortunately becomes quite tedious for the assoc rule r10. This is due to the complex binding structure of this rule which includes four binding occurrences of two different names. Furthermore, the binding structure changes from the left to the right: On the left hand side, $x$ is only bound in $t$, whereas on the right hand side the scope of $x$ extends over the whole term $t$ to $y$ in $u$.

```
lemma \(r 10^{\prime}\) [intro!]:
    assumes \(x f: x \sharp y \quad x \sharp u\)
    shows (s to \(x\) in t) to \(y\) in \(u \mapsto s\) to \(x\) in ( \(t\) to \(y\) in \(u\) )
proof -
    obtain \(y^{\prime}::\) name - suitably fresh
        where \(y: y^{\prime} \sharp s \quad y^{\prime} \sharp x \quad y^{\prime} \sharp t \quad y^{\prime} \sharp u\)
        using ex-fresh[of \(\left.\left(s, x, t, u,\left[\left(x, x^{\prime}\right)\right] \cdot t\right)\right]\)
        by (auto simp add: fresh-prod)
    obtain \(x^{\prime}:\) :name
        where \(x: x^{\prime} \sharp s \quad x^{\prime} \sharp y^{\prime} \quad x^{\prime} \sharp y \quad x^{\prime} \sharp t \quad x^{\prime} \sharp u\)
            \(x^{\prime} \sharp\left(\left[\left(y, y^{\prime}\right)\right] \cdot u\right)\)
        using ex-fresh \(\left[\right.\) of \(\left.\left(s, y^{\prime}, y, t, u,\left(\left[\left(y, y^{\prime}\right)\right] \cdot u\right)\right)\right]\)
        by (auto simp add: fresh-prod)
    from \(x y\) have yaux: \(y^{\prime} \sharp\left[\left(x, x^{\prime}\right)\right] \cdot t\)
        by (simp add: fresh-left perm-fresh-fresh fresh-atm)
    have (s to \(x\) in \(t\) ) to \(y\) in \(u=(s\) to \(x\) in \(t)\) to \(y^{\prime}\) in \(\left(\left[\left(y, y^{\prime}\right)\right] \cdot u\right)\)
        using \(\left\langle y^{\prime} \sharp u\right\rangle\) by (simp add: alpha \({ }^{\prime \prime}\) )
    also have \(\ldots=\left(\right.\) s to \(x^{\prime}\) in \(\left.\left(\left[\left(x, x^{\prime}\right)\right] \cdot t\right)\right)\) to \(y^{\prime}\) in \(\left(\left[\left(y, y^{\prime}\right)\right] \cdot u\right)\)
        using 〈 \(\left.x^{\prime} \sharp t\right\rangle\) by (simp add: alpha' \({ }^{\prime \prime}\) )
    also have \(\ldots \mapsto\) s to \(x^{\prime}\) in \(\left(\left(\left[\left(x, x^{\prime}\right)\right] \cdot t\right)\right.\) to \(y^{\prime}\) in \(\left.\left(\left[\left(y, y^{\prime}\right)\right] \cdot u\right)\right)\)
        using \(x\) y yaux by (auto simp add: fresh-prod)
    also have \(\ldots=s\) to \(x^{\prime}\) in \(\left(\left(\left[\left(x, x^{\prime}\right)\right] \cdot t\right)\right.\) to \(y\) in \(\left.u\right)\)
        using \(\left\langle y^{\prime} \sharp u\right\rangle\) by (simp add: abs-fun-eq1 alpha \({ }^{\prime \prime}\) )
    also have \(\ldots=s\) to \(x\) in ( \(t\) to \(y\) in \(u\) )
    proof (subst trm.inject)
        from \(x f\) have swap: \(\left[\left(x, x^{\prime}\right)\right] \cdot y=y \quad\left[\left(x, x^{\prime}\right)\right] \cdot u=u\)
        by (auto simp add: fresh-atm perm-fresh-fresh )
        with \(x\) show \(s=s \wedge\left[x^{\prime}\right] .\left(\left[\left(x, x^{\prime}\right)\right] \cdot t\right)\) to \(y\) in \(u=[x]\).t to \(y\) in \(u\)
            by (auto simp add: alpha \({ }^{\prime \prime}\left[\right.\) of \(\left.x^{\prime}-x\right]\) abs-fresh abs-fun-eq1 swap)
    qed
    finally show ?thesis.
qed
declare r10[rule del]
```

Since now all the introduction rules of the vc-compatible reduction relation exactly match their standard counterparts, both directions of the adequacy proof are trivial inductions.
theorem adequacy: $s \mapsto t=s \rightsquigarrow t$
by (auto elim:reduction.induct std-reduction.induct)
Next we show that the reduction relation preserves freshness and is in turn preserved under substitution.

```
lemma reduction-fresh:
    fixes x::name
    assumes r:t\mapsto t'
    shows }x\sharpt\Longrightarrowx\sharp\mp@subsup{t}{}{\prime
using }
```

by (nominal-induct $t t^{\prime}$ avoiding: x rule: reduction.strong-induct)
(auto simp add: abs-fresh fresh-fact fresh-atm)

```
lemma reduction-subst:
    assumes \(a\) : \(t \mapsto t^{\prime}\)
    shows \(t[x::=v] \mapsto t^{\prime}[x::=v]\)
using \(a\)
by (nominal-induct \(t t^{\prime}\) avoiding: \(x\) rule: reduction.strong-induct)
    (auto simp add: fresh-atm fresh-fact subst-lemma fresh-prod abs-fresh)
```

Following [Nom], we use an inductive variant of strong normalization, as it allows for inductive proofs on terms being strongly normalizing, without establishing that the reduction relation is finitely branching.

```
inductive
    SN :: trm }=>\mathrm{ bool
where
    SN-intro:(\bigwedge t' .t\mapsto t'\LongrightarrowSN 㰪)\LongrightarrowSN t
```

```
lemma \(S N\)-preserved[intro]:
    assumes \(a\) : SN \(t \quad t \mapsto t^{\prime}\)
    shows \(S N t^{\prime}\)
using \(a\) by (cases) (auto)
definition NORMAL :: trm \(\Rightarrow\) bool
where
    NORMAL \(t \equiv \neg\left(\exists t^{\prime} . t \mapsto t^{\prime}\right)\)
```

lemma normal-var: NORMAL (Var $x$ )
unfolding NORMAL-def by (auto elim: reduction.cases)
lemma normal-implies-sn: NORMAL $s \Longrightarrow S N s$
unfolding NORMAL-def by (auto intro: SN-intro)

## 4 Stacks

As explained in [LS05], the monadic type structure of the $\lambda_{m l}$-calculus does not lend itself to an easy definition of a logical relation along the type structure of the calculus. Therefore, we need to introduce stacks as an auxiliary notion to handle the monadic type constructor $T$. Stacks can be thought of as lists of term abstractions $[x] . t$. The notation for stacks is chosen with this resemblance in mind.

```
nominal-datatype stack \(=I d \mid\) St «name»trm stack \(([-]-\gg-)\)
lemma stack-exhaust:
    fixes \(c::\) ' \(a:: f s\)-name
    shows \(k=I d \vee(\exists y n l . y \sharp l \wedge y \sharp c \wedge k=[y] n \gg l)\)
by (nominal-induct \(k\) avoiding: c rule: stack.strong-induct) (auto)
nominal-primrec
    length :: stack \(\Rightarrow\) nat ( |-|)
where
```

$$
\begin{aligned}
& |I d|=0 \\
& \mid y \sharp L \Longrightarrow \text { length }([y] n \gg L)=1+|L| \\
& \text { by }(\text { finite-guess }+ \text {,auto simp add: fresh-nat,fresh-guess })
\end{aligned}
$$

Together with the stack datatype, we introduce the notion of dismantling a term onto a stack. Unfortunately, the dismantling operation has no easy primitive recursive formulation. The Nominal package, however, only provides a recursion combinator for primitive recursion. This means that for dismantling one has to prove pattern completeness, right uniqueness, and termination explicitly.

```
function
    dismantle \(::\) trm \(\Rightarrow\) stack \(\Rightarrow \operatorname{trm}(-\star-[160,160] 160)\)
where
    \(t \star I d=t \mid\)
    \(x \sharp(K, t) \Longrightarrow t \star([x] s \gg K)=(t\) to \(x\) in \(s) \star K\)
proof - - pattern completeness
    fix \(P\) :: bool and arg::trm \(\times\) stack
    assume \(i d: \wedge t\). arg \(=(t\), stack.Id \() \Longrightarrow P\)
        and st: \(\wedge x K t s . \llbracket x \sharp(K, t) ; \arg =(t,[x] s \gg K) \rrbracket \Longrightarrow P\)
    \{ assume snd arg \(=I d\)
        hence \(P\) by (metis id[where \(t=f\) st arg] surjective-pairing) \(\}\)
    moreover
    \{ fix \(y n L\) assume snd arg \(=[y] n \gg L \quad y \sharp(L\), fst arg \()\)
        hence \(P\) by (metis st[where \(t=f\) st arg] surjective-pairing) \(\}\)
```



```
next
```

    - right uniqueness
    - only the case of the second equation matching both args needs to be shown.
    fix \(t t^{\prime}::\) trm and \(x x^{\prime}::\) name and \(s s^{\prime}::\) trm and \(K K^{\prime}::\) stack
    let \(? g=\) dismantle-sum \(C-\) graph of dismantle
    assume \(x \sharp(K, t) \quad x^{\prime} \sharp\left(K^{\prime}, t^{\prime}\right)\)
        and \((t,[x] s \gg K)=\left(t^{\prime},[x] s^{\prime} \gg K^{\prime}\right)\)
    thus ? \(g(t\) to \(x\) in \(s, K)=? g\left(t^{\prime}\right.\) to \(x^{\prime}\) in \(\left.s^{\prime}, K^{\prime}\right)\)
        by (auto intro!: arg-cong[where \(f=\) ?g] simp add: stack.inject)
    qed (simp-all add: stack.inject) - all other cases are trivial
termination dismantle
by $($ relation measure $(\lambda(t, K) .|K|))$ (auto)

Like all our constructions, dismantling is equivariant. Also, freshness can be pushed over dismantling, and the freshness requirement in the second defining equation is not needed
lemma dismantle-eqvt[eqvt]:
fixes $p i::($ name $\times$ name) list
shows $p i \cdot(t \star K)=(p i \cdot t) \star(p i \cdot K)$
by (nominal-induct $K$ avoiding: pi t rule:stack.strong-induct)
(auto simp add: equts fresh-bij)
lemma dismantle-fresh[iff]:
fixes $x::$ name
shows $(x \sharp(t \star k))=(x \sharp t \wedge x \sharp k)$
by (nominal-induct $k$ avoiding: $t x$ rule: stack.strong-induct)
(simp-all)

```
lemma dismantle-simp \([\operatorname{simp}]: s \star[y] n \gg L=(s\) to \(y\) in \(n) \star L\)
proof -
    obtain \(x\) :: name where \(f: x \sharp s \quad x \sharp L \quad x \sharp n\)
        using ex-fresh[of ( \(s, L, n\) )] by (auto simp add:fresh-prod)
    hence \(t\) : \(s\) to \(y\) in \(n=s\) to \(x\) in \(([(y, x)] \cdot n)\)
        by (auto simp add: alpha'/)
    from \(f\) have \([y] n \gg L=[x]([(y, x)] \cdot n) \gg L\)
        by (auto simp add: stack.inject alpha'")
    hence \(s \star[y] n \gg L=s \star[x]([(y, x)] \cdot n) \gg L\) by \(\operatorname{simp}\)
    also have \(\ldots=(\) s to \(y\) in \(n) \star L\) using \(f t \operatorname{by}(\operatorname{simp}\) del:trm.inject)
    finally show ?thesis .
qed
```

We also need a notion of reduction on stacks. This reduction relation allows us to define strong normalization not only for terms but also for stacks and is needed to prove the properties of the logical relation later on.

```
definition stack-reduction :: stack \(\Rightarrow\) stack \(\Rightarrow\) bool ( - \(\mapsto-)\)
where
    \(k \mapsto k^{\prime} \equiv \forall(t:: t r m) \cdot(t \star k) \mapsto\left(t \star k^{\prime}\right)\)
lemma stack-reduction-fresh:
    fixes \(k::\) stack and \(x\) :: name
    assumes \(r: k \mapsto k^{\prime}\) and \(f: x \sharp k\)
    shows \(x \sharp k^{\prime}\)
proof -
    from ex-fresh[of \(x]\) obtain \(z:: n a m e\) where \(f^{\prime}: z \sharp x\)..
    from \(r\) have Var \(z \star k \mapsto \operatorname{Var} z \star k^{\prime}\) unfolding stack-reduction-def ..
    moreover from \(f f^{\prime}\) have \(x \sharp \operatorname{Var} z \star k\) by (auto simp add: fresh-atm)
    ultimately have \(x \sharp\) Var \(z \star k^{\prime}\) by (rule reduction-fresh)
    thus \(x \sharp k^{\prime}\) by simp
qed
lemma dismantle-red[intro]:
    fixes \(m\) :: trm
    assumes \(r: m \mapsto m^{\prime}\)
    shows \(m \star k \mapsto m^{\prime} \star k\)
using \(r\)
by (nominal-induct \(k\) avoiding: \(m m^{\prime}\) rule:stack.strong-induct) auto
```

Next we define a substitution operation for stacks. The main purpose of this is to distribute substitution over dismantling.

```
nominal-primrec
    ssubst :: name \(\Rightarrow\) trm \(\Rightarrow\) stack \(\Rightarrow\) stack
where
    ssubst x v \(I d=I d\)
    \(\mid y \sharp(k, x, v) \Longrightarrow\) ssubst \(x v([y] n \gg k)=[y](n[x::=v]) \gg(\) ssubst \(x\) v \(k)\)
by(finite-guess+, (simp add: abs-fresh \()+\), fresh-guess + )
lemma ssubst-fresh:
    fixes \(y\) :: name
    assumes \(y \sharp(x, v, k)\)
    shows \(y \sharp\) ssubst \(x v k\)
using assms
by (nominal-induct \(k\) avoiding: \(y x\) v rule: stack.strong-induct)
```

(auto simp add: fresh-prod fresh-atm abs-fresh fresh-fact)

```
lemma ssubst-forget:
    fixes \(x\) :: name
    assumes \(x \sharp k\)
    shows ssubst \(x \vee k=k\)
using assms
by (nominal-induct \(k\) avoiding: \(x v\) rule: stack.strong-induct)
    (auto simp add: abs-fresh fresh-atm forget)
```

lemma subst-dismantle $[$ simp $]:(t \star k)[x::=v]=(t[x::=v]) \star$ ssubst $x v k$
by(nominal-induct $k$ avoiding: $t x v$ rule: stack.strong-induct)
(auto simp add: ssubst-fresh fresh-prod fresh-fact)

## 5 Reducibility for Terms and Stacks

Following [Nom], we formalize the logical relation as a function $R E D$ of type $t y \Rightarrow t r m$ set for the term part and accordingly $S R E D$ of type $t y \Rightarrow$ stack set for the stack part of the logical relation.
lemma ty-exhaust: ty $=$ TBase $\vee(\exists \sigma \tau . t y=\sigma \rightarrow \tau) \vee(\exists \sigma . t y=T \sigma)$
by (induct ty rule:ty.induct) (auto)
function $R E D::$ ty $\Rightarrow$ trm set
and $\quad S R E D::$ ty $\Rightarrow$ stack set
where
$R E D($ TBase $)=\{t . S N(t)\}$
$\mid R E D(\tau \rightarrow \sigma)=\{t . \forall u \in R E D \tau .(\operatorname{App} t u) \in R E D \sigma\}$
|RED $(T \sigma)=\{t . \forall k \in S R E D \sigma \cdot S N(t \star k)\}$
| SRED $\tau=\{k . \forall t \in R E D \tau . S N([t] \star k)\}$
by(auto simp add: ty.inject, case-tac x rule: sum.exhaust,insert ty-exhaust) (blast)+

This is the second non-primitive function in the formalization. Since types do not involve binders, pattern completeness and right uniqueness are mostly trivial. The termination argument is not as simple as for the dismantling function, because the definiton of $S R E D \tau$ involves a recursive call to $R E D \tau$ without reducing the size of $\tau$.

## nominal-primrec

tsize :: $t y \Rightarrow$ nat
where
tsize TBase $=1$
|tsize $(\sigma \rightarrow \tau)=1+$ tsize $\sigma+$ tsize $\tau$
|tsize $(T \tau)=1+$ tsize $\tau$
by (rule TrueI) +
In the termination argument below, $\operatorname{Inl} \tau$ corresponds to the call $R E D \tau$, whereas Inr $\tau$ corresponds to $S R E D \tau$
termination $R E D$
by (relation measure
( $\lambda$ x. case $x$ of Inl $\tau \Rightarrow 2 *$ tsize $\tau$
| Inr $\tau \Rightarrow 2 *$ tsize $\tau+1$ )) (auto)

## 6 Properties of the Reducibility Relation

After defining the logical relations we need to prove that the relation implies strong normalization, is preserved under reduction, and satisfies the head expansion property.

```
definition NEUT :: trm \(\Rightarrow\) bool
where
    NEUT \(t \equiv(\exists a . t=\operatorname{Var} a) \vee(\exists t 1 t 2 . t=A p p t 1 t 2)\)
definition \(C R 1\) :: ty \(\Rightarrow\) bool
where
    \(C R 1 \tau \equiv \forall t .(t \in R E D \tau \longrightarrow S N t)\)
definition CR2 :: ty \(\Rightarrow\) bool
where
    \(C R 2 \tau \equiv \forall t t^{\prime} .\left(t \in R E D \tau \wedge t \mapsto t^{\prime}\right) \longrightarrow t^{\prime} \in R E D \tau\)
definition CR3-RED :: trm \(\Rightarrow t y \Rightarrow\) bool
where
    CR3-RED \(t \tau \equiv \forall t^{\prime} . t \mapsto t^{\prime} \longrightarrow t^{\prime} \in R E D \tau\)
definition \(C R 3\) :: ty \(\Rightarrow\) bool
where
    \(C R 3 \tau \equiv \forall t .(\) NEUT \(t \wedge C R 3-R E D t \tau) \longrightarrow t \in R E D \tau\)
definition \(C R_{4}\) :: ty \(\Rightarrow\) bool
where
    \(C R 4 \tau \equiv \forall t .(N E U T t \wedge N O R M A L t) \longrightarrow t \in R E D \tau\)
```

lemma CR3-implies-CR4[intro]:CR3 $\tau \Longrightarrow$ CR4 $\tau$
by (auto simp add: CR3-def CR3-RED-def CR4-def NORMAL-def)
inductive
FST :: trm $\Rightarrow$ trm $\Rightarrow$ bool ( - »-[80,80] 80)
where
fst[intro!]: (App ts)»t
lemma $S N$-of-FST-of-App:
assumes a: SN (Appts)
shows $S N t$
proof -
from $a$ have $\forall z$. (App $t s » z) \longrightarrow S N z$
by (induct rule: SN.induct)
(blast elim: FST.cases intro: SN-intro)
then show $S N t$ by blast
qed

The lemma above is a simplified version of the one used in [Nom]. Since we have generalized our notion of reduction from terms to stacks, we can also generalize the notion of strong normalization. The new induction principle will be used to prove the $T$ case of the properties of the reducibility relation.
inductive

$$
S S N \text { :: stack } \Rightarrow \text { bool }
$$

where
SSN-intro: $\left(\bigwedge k^{\prime} . k \mapsto k^{\prime} \Longrightarrow S S N k^{\prime}\right) \Longrightarrow S S N k$
Furthermore, the approach for deriving strong normalization of subterms from above can be generalized to terms of the form $t \star k$. In contrast to the case of applications, $t \star k$ does not uniquely determine $t$ and $k$. Thus, the extraction is a proper relation in this case.

```
inductive
    SND-DIS :: trm }=>\mathrm{ stack }=>\mathrm{ bool (- - -)
where
    snd-dis[intro!]: t \star k\triangleright 
lemma SN-SSN:
    assumes a:SN (t\stark)
    shows SSN k
proof -
    from a have }\forallz.(t\stark\trianglerightz)\longrightarrowSSNz by (induct rule: SN.induct
            (metis SND-DIS.cases SSN-intro snd-dis stack-reduction-def)
    thus SSN k by blast
qed
```

To prove CR1-3, the authors of [LS05] use a case distinction on the reducts of $t \star k$, where $t$ is a neutral term and therefore no interaction occurs between $t$ and $k$.


We strive for a proof of this rule by structural induction on $k$. The general idea of the case where $k=[y] n \gg l$ is to move the first stack frame into the term $t$ and then apply the induction hypothesis as a case rule. Unfortunately, this term is no longer neutral, so, for the induction to go through, we need to generalize the claim to also include the possible interactions of non-neutral terms and stacks.

```
lemma dismantle-cases:
    fixes \(t::\) trm
    assumes \(r: t \star k \mapsto r\)
    and \(T: \wedge t^{\prime} . \llbracket t \mapsto t^{\prime} ; r=t^{\prime} \star k \rrbracket \Longrightarrow P\)
    and \(K: \bigwedge k^{\prime} . \llbracket k \mapsto k^{\prime} ; r=t \star k^{\prime} \rrbracket \Longrightarrow P\)
    and \(B: \bigwedge s\) y n \(l . \llbracket t=[s] ; k=[y] n \gg l ; r=(n[y::=s]) \star l \rrbracket \Longrightarrow P\)
    and \(A: \bigwedge u x v y n l . \llbracket x \sharp y ; x \sharp n ; t=u\) to \(x\) in \(v ;\)
        \(k=[y] n \gg l ; r=(u\) to \(x\) in \((v\) to \(y\) in \(n)) \star l \rrbracket \Longrightarrow P\)
    shows \(P\)
using assms
proof (nominal-induct \(k\) avoiding: \(t r\) rule:stack.strong-induct)
    case \((S t y n L)\) note \(y\) fresh \(=\langle y \sharp t\rangle\langle y \sharp r\rangle\langle y \sharp L\rangle\)
    note \(I H=S t(4)\)
        and \(T=S t(6)\) and \(K=S t(7)\) and \(B=S t(8)\) and \(A=S t(9)\)
    thus \(P\) proof (cases rule: \(I H[\) where \(b=t\) to \(y\) in \(n\) and \(b a=r]\) )
        case (2 \(r^{\prime}\) ) have red: to to in \(n \mapsto r^{\prime}\) and \(r: r=r^{\prime} \star L\) by fact +
```

If $m$ to $y$ in $n$ makes a step we reason by case distinction on the successors of $m$ to $y$ in $n$. We want to use the strong inversion principle for the reduction relation. For this we need that $y$ is fresh for to $y$ in $n$ and $r^{\prime}$.

```
from yfresh \(r\) have \(y: y \sharp t\) to \(y\) in \(n \quad y \sharp r^{\prime}\)
    by (auto simp add: abs-fresh)
obtain \(z\) where \(z: z \neq y \quad z \sharp r^{\prime} \quad z \sharp t\) to \(y\) in \(n\)
    using ex-fresh[of ( \(y, r^{\prime}\), t to \(y\) in \(n\) )]
    by (auto simp add:fresh-prod fresh-atm)
from red \(r\) show \(P\)
proof (cases rule:reduction.strong-cases
        [ where \(x=y\) and \(x a=y\) and \(x b=y\) and \(x c=y\) and \(x d=y\)
        and \(x e=y\) and \(x f=y\) and \(x g=z\) and \(y=y]\) )
    case \(\left(r 6 s t^{\prime} u\right)\) - if \(t\) makes a step we use assumption T
    with \(y\) have \(m: t \mapsto t^{\prime} \quad r^{\prime}=t^{\prime}\) to \(y\) in \(n\) by auto
    thus \(P\) using \(T\) [of \(t\rceil r\) by auto
next
    case \(\left(r^{\prime} 7-n^{\prime}\right)\) with \(y\) have \(n: n \mapsto n^{\prime}\) and \(r^{\prime}: r^{\prime}=t\) to \(y\) in \(n^{\prime}\)
        by (auto simp add: alpha)
```

Since $k=[y] n \gg L$, the reduction $n \mapsto n^{\prime}$ occurs within the stack $k$. Hence, we need to establish this stack reduction.

```
    have \([y] n \gg L \mapsto[y] n^{\prime} \gg L\) unfolding stack-reduction-def
    proof
        fix \(u\) have \(u\) to \(y\) in \(n \mapsto u\) to \(y\) in \(n^{\prime}\) using \(n\)..
        hence \((u\) to \(y\) in \(n) \star L \mapsto\left(u\right.\) to \(y\) in \(\left.n^{\prime}\right) \star L\)..
        thus \(u \star[y] n \gg L \mapsto u \star[y] n^{\prime} \gg L\)
        by \(\operatorname{simp}\)
    qed
    moreover have \(r=t \star[y] n^{\prime} \gg L\) using \(r r^{\prime}\) by \(\operatorname{simp}\)
    ultimately show \(P\) by (rule \(K\) )
next
    case (r8s-) - the case of a \(\beta\)-reduction is exactly B
    with \(y\) have \(t=[s] \quad r^{\prime}=n[y::=s]\) by (auto simp add: alpha)
    thus \(P\) using \(B[o f s\) s \(n L] r\) by auto
next
    case (r9 -) - The case of an \(\eta\)-reduction is a stack reduction as well.
    with \(y\) have \(n: n=[\) Var \(y]\) and \(r^{\prime}: r^{\prime}=t\)
        by (auto simp add: alpha)
    \{ fix \(u\) have \(u\) to \(y\) in \(n \mapsto u\) unfolding \(n\)..
        hence ( \(u\) to \(y\) in \(n\) ) \(\star L \mapsto u \star L\)..
        hence \(u \star[y] n \gg L \mapsto u \star L\) by simp
    \} hence \([y] n \gg L \mapsto L\) unfolding stack-reduction-def ..
    moreover have \(r=t \star L\) using \(r r^{\prime}\) by simp
    ultimately show \(P\) by (rule \(K\) )
    next
        case \((r 10 u-v)\) - The assoc case holds by A.
        with \(y z\) have
        \(t=(u\) to \(z\) in \(v)\)
        \(r^{\prime}=u\) to \(z\) in ( \(v\) to \(y\) in \(n\) )
        \(z \sharp(y, n)\) by (auto simp add: fresh-prod alpha)
    thus \(P\) using \(A[o f z y n] r\) by auto
    qed (insert \(y\), auto) - No other reductions are possible.
next
```

Next we have to solve the case where a reduction occurs deep within $L$. We get a reduction of the stack $k$ by moving the first stack frame "[y]n" back to the right hand
side of the dismantling operator.

```
    case (3 L')
    hence L:L\mapsto L' and r:r = (t to y in n)\star L' by auto
    { fix s from L have (s to y in n)\starL\mapsto(s to y in n)\star L'
        unfolding stack-reduction-def ..
    hence s\star [y]n>>L\mapstos\star [y]n>> L' by simp
    } hence [y]n>>L\mapsto[y]n>>L'
    moreover from r have r=t\star [y]n>> L' by simp
    ultimately show P by (rule K)
next
    case (5xz n' s v K) - The "assoc" case is again a stack reduction
    have }xf:x\sharpz\quadx\sharp\mp@subsup{n}{}{\prime
        - We get the following equalities
    and red: t to y in n=s to x in v
                L=[z]n}\mp@subsup{n}{}{\prime}>>
                r=(s to x in v to z in n) \star K by fact+
    { fix }u\mathrm{ from red have }u\star[y]n>>L=((u to x in v) to z in n')\star
        by(auto intro:arg-cong[where f=\lambdax.x\starK])
        moreover
        { from xf have (u to x in v) to z in n'\mapsto | to x in (v to z in n').
            hence ((u to x in v) to z in n') \star K\mapsto(u to x in (v to z in n')) \star K
                by rule
        } ultimately have }u\star[y]n>>L\mapsto(u\mathrm{ to }x\mathrm{ in (v to z in n')) *K
        by (simp (no-asm-simp) del:dismantle-simp)
        hence}u\star[y]n>>L\mapstou\star[x](v to z in n')>> b by sim
    } hence [y]n>>L\mapsto [x](v to z in n')>}>>
        unfolding stack-reduction-def by simp
    moreover have r=t\star ([x](v to z in n')>>K) using red
        by (auto)
    ultimately show P by (rule K)
    qed (insert St, auto )
qed auto
```

Now that we have established the general claim, we can restrict $t$ to neutral terms only and drop the cases dealing with possible interactions.

```
lemma dismantle-cases'[consumes 2, case-names T K]:
    fixes \(m\) :: trm
    assumes \(r: t \star k \mapsto r\)
    and NEUT \(t\)
    and \(\bigwedge t^{\prime} . \llbracket t \mapsto t^{\prime} ; r=t^{\prime} \star k \rrbracket \Longrightarrow P\)
    and \(\bigwedge k^{\prime} . \llbracket k \mapsto k^{\prime} ; r=t \star k^{\prime} \rrbracket \Longrightarrow P\)
    shows \(P\)
using assms unfolding NEUT-def
by (cases rule: dismantle-cases \([o f t k r])\) (auto)
lemma red-Ret:
    fixes \(t::\) trm
    assumes \([s] \mapsto t\)
    shows \(\exists s^{\prime} \cdot t=\left[s^{\prime}\right] \wedge s \mapsto s^{\prime}\)
using assms by cases (auto)
lemma \(S N\)-Ret: \(S N u \Longrightarrow S N[u]\)
by(induct rule:SN.induct) (metis SN.intros red-Ret)
```

All the properties of reducibility are shown simultaneously by induction on
the type．Lindley and Stark［LS05］only spell out the cases dealing with the monadic type constructor $T$ ．We do the same by reusing the proofs from［Nom］ for the other cases．To shorten the presentation，these proofs are omitted

```
lemma RED-props:
    shows CR1 \tau and CR2 }\tau\mathrm{ and CR3 }
proof (nominal-induct \tau rule: ty.strong-induct)
    case TBasenext
    case (TFun \tau1 \tau2)next
    case (T \sigma)
    { case 1- follows from the fact that stack.Id \inSRED \sigma
        have ih-CR1-\sigma:CR1 \sigma by fact
        { fix t assume t-red: t\inRED (T \sigma)
            { fix s assume s\inRED\sigma
                hence SN s using ih-CR1-\sigma by (auto simp add: CR1-def)
                hence SN ([s]) by (rule SN-Ret)
                hence SN ([s]\star Id) by simp
            } hence Id }\inSRED \sigma by sim
            with t-red have SN (t) by (auto simp del: SRED.simps)
        } thus CR1 (T \sigma) unfolding CR1-def by blast
    next
        case 2 - follows since SN is preserved under redcution
        { fix t t'::trm assume t-red:t\inRED (T \sigma) and t-t':t\mapsto 多
            { fix k assume k: k\inSRED \sigma
                with t-red have }SN(t\stark)\mathrm{ by simp
                moreover from t-t' have t \star k\mapsto 㰪\star k ..
                ultimately have}SN(\mp@subsup{t}{}{\prime}\stark)\mathrm{ by (rule SN-preserved)
            } hence t'\inRED (T \sigma) by (simp del: SRED.simps)
        } thus CR2 (T \sigma)unfolding CR2-def by blast
    next
        case 3 from \CR3 \sigma\rangle have ih-CR4-\sigma:CR4 \sigma ..
```



```
            and neut-t: NEUT t
            { fix k assume k-red: k G SRED \sigma
                fix }x\mathrm{ have NEUT (Var x) unfolding NEUT-def by simp
                hence Var x ERED \sigma using normal-var ih-CR4-\sigma
                    by (simp add:CR4-def)
                hence SN ([Var x]* k) using k-red by simp
                hence SSN k by (rule SN-SSN)
                then have SN (t\stark) using k-red
                proof (induct k rule:SSN.induct)
                    case (SSN-intro k)
                    have ih:\k'.\llbracketk\mapsto k
                    and k-red: }k\inSRED\sigma\mathrm{ by fact+
                    { fix r assume r: t\stark\mapstor
                    hence SN r using neut-t
                    proof (cases rule:dismantle-cases')
                            case (T t') hence t-t':t\mapsto t' and r-def:r= t'^k.
                    from }t-\mp@subsup{t}{}{\prime}\mathrm{ have }\mp@subsup{t}{}{\prime}\inRED(T\sigma)\mathrm{ by (rule t'-red)
                    thus SNr using k-red r-def by simp
                    next
                    case (K k') hence k-k':k\mapsto 隌 and r-def:r=t\star 䖮.
                    {fix s assume s\inRED\sigma
                        hence SN}([s]\stark) using k-red
                        by simp
                        moreover have [s]\stark\mapsto[s]\star k'
```

Let $t$ be neutral such that $t^{\prime} \in R E D_{T \sigma}$ whenever $t \mapsto t^{\prime}$. We have to show that $(t \star k)$ is $S N$ for each $k \in S R E D_{\sigma}$. First, we have that $[x] \star k$ is $S N$, as $x \in R E D_{\sigma}$ by the induction hypothesis. Hence $k$ itself is $S N$, and we can work by induction on $\max (k)$. Application $t \star k$ may reduce as follows:

- $t^{\prime} \star k$, where $t \mapsto t^{\prime}$, which is $S N$ as $k \in S R E D_{\sigma}$ and $t^{\prime} \in R E D_{T \sigma}$.
- $t \star k^{\prime}$, where $k \mapsto k^{\prime}$. For any $s \in R E D_{\sigma},[s] \star k$ is $S N$ as $k \in S R E D_{\sigma}$; and $[s] \star k \mapsto$ $[s] \star k^{\prime}$, so $[s] \star k^{\prime}$ is also $S N$. From this we have $k^{\prime} \in S R E D_{\sigma}$ with $\max \left(k^{\prime}\right)<\max (k)$, so by induction hypothesis $t \star k^{\prime}$ is $S N$.

There are no other possibilities as $t$ is neutral. Hence $t \star k$ is strongly normalizing for every $k \in S R E D_{\sigma}$, and so $t \in R E D_{T \sigma}$ as required.

Figure 1: Proof of the case $T \sigma$ subcase CR3 as in [LS05]

```
                                    using k-k' unfolding stack-reduction-def ..
                    ultimately have SN ([s]\star k') ..
                } hence }\mp@subsup{k}{}{\prime}\inSRED \sigma by sim
                with }k\mathrm{ -k' show SN r unfolding r-def by (rule ih)
                qed } thus SN (t\stark)..
            qed } hence }t\inRED(T\sigma)\mathrm{ by simp
    } thus CR3 (T \sigma) unfolding CR3-def CR3-RED-def by blast
}
qed
```

The last case above shows that, once all the reasoning principles have been established, some proofs have a formalization which is amazingly close to the informal version. For a direct comparison, the informal proof is presented in Figure 1.

Now that we have established the properties of the reducibility relation, we need to show that reducibility is preserved by the various term constructors. The only nontrivial cases are abstraction and sequencing.

## 7 Abstraction Preserves Reducibility

Once again we could reuse the proofs from [Nom]. The proof uses the double$S N$ rule and the lemma red-Lam below. Unfortunately, this time the proofs are not fully identical to the proofs in [Nom] because we consider $\beta \eta$-reduction rather than $\beta$-reduction only. However, the differences are only minor.
lemma double-SN[consumes 2]:
assumes $a$ : $S N a$
and $\quad b: S N b$
and $\quad c: \bigwedge(x:: t r m)(z:: t r m)$.
$\llbracket \bigwedge y . x \mapsto y \Longrightarrow P y z ; \bigwedge u . z \mapsto u \Longrightarrow P x u \rrbracket \Longrightarrow P x z$
shows $P a b$
using $a b c$

```
lemma red-Lam:
    assumes \(a: \Lambda x . t \mapsto r\)
    shows \(\left(\exists t^{\prime} . r=\Lambda x . t^{\prime} \wedge t \mapsto t^{\prime}\right) \vee(t=\operatorname{App} r(\operatorname{Var} x) \wedge x \sharp r)\)
proof -
    obtain \(z::\) :name where \(z: z \sharp x \quad z \sharp t \quad z \sharp r\)
        using ex-fresh[of ( \(x, t, r\) )] by (auto simp add: fresh-prod)
    have \(x \sharp \Lambda x . t\) by (simp add: abs-fresh)
    with \(a\) have \(x \sharp r\) by (simp add: reduction-fresh)
    with \(a\) show ?thesis using \(z\)
        by (cases rule: reduction.strong-cases
            [where \(x=x\) and \(x a=x\) and \(x b=x\) and \(x c=x\) and
                    \(x d=x\) and \(x e=x\) and \(x f=x\) and \(x g=x\) and \(y=z])\)
            (auto simp add: abs-fresh alpha fresh-atm)
qed
lemma abs-RED:
    assumes asm: \(\forall s \in R E D \tau . t[x::=s] \in R E D \sigma\)
    shows \(\Lambda x . t \in R E D(\tau \rightarrow \sigma)\)
```


## 8 Sequencing Preserves Reducibility

This section corresponds to the main part of the paper being formalized and as such deserves special attention. In the lambda case one has to formalize doing induction on $\max (s)+\max (t)$ for two strongly normalizing terms $s$ and $t$ (cf. [GTL89, Section 6.3]). Above, this was done through a double-SN rule. The central Lemma 7 of Lindley and Stark's paper uses an even more complicated induction scheme. They assume terms $p$ and $n$ as well as a stack $K$ such that $S N p$ and $S N(n[x::=p] \star K)$. The induction is then done on $|K|+\max (n \star K)+\max (p)$. See Figure 2 in for details.

Since we have settled for a different characterization of strong normalization, we have to derive an induction principle similar in spirit to the double-SN rule. Furthermore, it turns out that it is not necessary to formalize the fact that stack reductions do not increase the length of the stack. ${ }^{1}$ Doing induction on the sum above, this is necessary to handle the case of a reduction occurring in $K$. We differ from [LS05] and establish an induction principle which to some extent resembles the lexicographic order on

$$
(S N, \mapsto) \times(S N, \mapsto) \times(\mathbb{N},>)
$$

```
lemma triple-induct[consumes 2]:
    assumes \(a\) : \(S N(p)\)
    and \(b: S N(q)\)
    and hyp: \(\bigwedge(p:: t r m)(q:: t r m)(k:: s t a c k)\).
    \(\llbracket \bigwedge p^{\prime} \cdot p \mapsto p^{\prime} \Longrightarrow P p^{\prime} q k ;\)
        \(\bigwedge q^{\prime} k . q \mapsto q^{\prime} \Longrightarrow P p q^{\prime} k ;\)
        \(\bigwedge k^{\prime} .\left|k^{\prime}\right|<|k| \Longrightarrow P p q k^{\prime} \rrbracket \Longrightarrow P p q k\)
    shows \(P p q k\)
proof -
```

[^0]Lemma 8.1. (Lemma 7) Let $p, n$ be terms and $K$ a stack such that $S N(p)$ and $S N(n[x::=$ $p] \star K)$. Then $S N(([p]$ to $x$ in $n) \star K)$

Proof. We show by induction on $|K|+\max (n \star K)+\max (p)$ that the reducts of ( $[p]$ to $x$ in $n$ ) $\star K$ are all strongly normalizing. The interesting reductions are as follows:

- T. $\beta$ giving $n[x::=p] \star K$ which is strongly normalizing by hypothesis.
- T. $\eta$ when $n=[x]$ giving $[p] \star K$. But $[p] \star K=n[x::=p] \star K$ which is again strongly normalizing by hypothesis
- T.assoc in the case where $K=[y] m \gg K^{\prime}$ with $x \notin f v(m)$; giving the reduct ([p] to $x$ in $(n$ to $y$ in $m)) \star K$. We aim to apply the induction hypothesis with $K^{\prime}$ and ( $n$ to $y$ in $m$ ) for $K$ and $n$ respectively. Now

$$
\begin{aligned}
(n \text { to } y \text { in } m)[x::=p] \star K^{\prime} & =(n[x::=p] \text { to } y \text { in } m) \star K^{\prime} \\
& =n[x::=p] \star K
\end{aligned}
$$

which is strongly normalizing by induction hypothesis. Also

$$
\left|K^{\prime}\right|+\max \left((n \text { to } y \text { in } m) \star K^{\prime}\right)+\max (p)<|K|+\max (n \star K)+\max (p)
$$

as $\left|K^{\prime}\right|<|K|$ and $(n$ to $y$ in $m) \star K^{\prime}=n \star K$. This last equation explains the use of $\max (n \star K)$; it remains fixed under T.assoc unlike $\max (K)$ and $\max (n)$. Applying the induction hypothesis gives $S N(([p]$ to $x$ in $(n$ to $y$ in $m)) \star K)$ as required.

Other reductions are confined to $K, n$ or $p$ and can be treated by the induction hypothesis, decreasing either $\max (n \star K)$ or $\max (p)$.

Figure 2: Proof of Lemma 7 as in [LS05]

```
from \(a\) have \(\bigwedge q K . S N q \Longrightarrow P p q K\)
proof (induct p)
    case ( \(S N\)-intro \(p\) )
    have sn1: \(\bigwedge p^{\prime} q K . \llbracket p \mapsto p^{\prime} ; S N q \rrbracket \Longrightarrow P p^{\prime} q K\) by fact
    have \(s n-q\) : \(S N q \quad S N q\) by fact+
    thus \(P p q K\)
    proof (induct q arbitrary: K)
        case (SN-intro q K)
        have sn2: \(\wedge q^{\prime} K . \llbracket q \mapsto q^{\prime} ; S N q^{\dagger} \rrbracket \Longrightarrow P p q^{\prime} K\) by fact
        show \(P\) p \(q K\)
        proof (induct \(K\) rule: measure-induct-rule \([\) where \(f=\) length \(]\) )
            case (less \(k\) )
            have le: \(\wedge k^{\prime} .\left|k^{\prime}\right|<|k| \Longrightarrow P p q k^{\prime}\) by fact
            \{ fix \(p^{\prime}\) assume \(p \mapsto p^{\prime}\)
                moreover have \(S N q\) by fact
                ultimately have \(P p^{\prime} q k\) using sn1 by auto \}
            moreover
            \{ fix \(q^{\prime} K\) assume \(r: q \mapsto q^{\prime}\)
                have \(S N q\) by fact
                hence \(S N q^{\prime}\) using \(r\) by (rule \(S N\)-preserved)
                with \(r\) have \(P\) p \(q^{\prime} K\) using sn2 by auto \}
            ultimately show ?case using le
                by (auto intro:hyp)
        qed
    qed
    qed
    with \(b\) show ?thesis by blast
qed
```

Here we strengthen the case rule for terms of the form $t \star k \mapsto r$. The freshness requirements on $x, y$, and $z$ correspond to those for the rule reduction.strongcases, the strong inversion principle for the reduction relation.

```
lemma dismantle-strong-cases:
    fixes \(t::\) trm
    assumes \(r: t \star k \mapsto r\)
    and \(f: y \sharp(t, k, r) \quad x \sharp(z, t, k, r) \quad z \sharp(t, k, r)\)
    and \(T: \bigwedge t^{\prime} . \llbracket t \mapsto t^{\prime} ; r=t^{\prime} \star k \rrbracket \Longrightarrow P\)
    and \(K: \bigwedge k^{\prime} . \llbracket k \mapsto k^{\prime} ; r=t \star k^{\prime} \rrbracket \Longrightarrow P\)
    and \(B: \bigwedge s n l . \llbracket t=[s]\);
        \(k=[y] n \gg l ; r=(n[y::=s]) \star l \rrbracket \Longrightarrow P\)
    and \(A: \bigwedge u v n l\).
                \(\llbracket x \sharp(z, n) ; t=u\) to \(x\) in \(v ; k=[z] n \gg l ;\)
            \(r=(u\) to \(x\) in \((v\) to \(z\) in \(n)) \star l \rrbracket \Longrightarrow P\)
    shows \(P\)
proof (cases rule:dismantle-cases \([\) of \(t k r P]\) )
    case (4s \(y^{\prime} n L\) ) have \(c h\) :
        \(t=[s]\)
        \(k=[y] n \gg L\)
    \(r=n\left[y^{\prime}::=s\right] \star L\) by fact +
```

The equations we get look almost like those we need to instantiate the hypothesis $B$. The only difference is that $B$ only applies to $y$, and since we want $y$ to become an instantiation variable of the strengthened rule, we only know that $y$ satisfies $f$ and nothing else. But the condition $f$ is just strong enough to rename $y^{\prime}$ to $y$ and apply $B$.

```
with \(f\) have \(y=y^{\prime} \vee y \sharp n\)
    by (auto simp add: fresh-prod abs-fresh)
    hence \(n\left[y^{\prime}::=s\right]=\left(\left[\left(y, y^{\prime}\right)\right] \cdot n\right)[y::=s]\)
    and \(\left[y^{\prime}\right] n \gg L=[y]\left(\left[\left(y, y^{\prime}\right)\right] \cdot n\right) \gg L\)
    by(auto simp add: name-swap-bij subst-rename' stack.inject alpha')
    with \(c h\) have \(t=[s]\)
    \(k=[y]\left(\left[\left(y, y^{\prime}\right)\right] \cdot n\right) \gg L\)
    \(r=\left(\left[\left(y, y^{\prime}\right)\right] \cdot n\right)[y::=s] \star L\)
    by (auto)
    thus \(P\) by (rule \(B\) )
next
    case (5u \(\left.x^{\prime} v z^{\prime} n L\right)\) have \(c h\) :
        \(x^{\prime} \sharp z^{\prime} \quad x^{\prime} \sharp n\)
        \(t=u\) to \(x^{\prime}\) in \(v\)
    \(k=[z] n \gg L\)
    \(r=\left(u\right.\) to \(x^{\prime}\) in \(v\) to \(z^{\prime}\) in \(\left.n\right) \star L\) by fact +
```

We want to do the same trick as above but at this point we have to take care of the possibility that $x$ might coincide with $x^{\prime}$ or $z^{\prime}$. Similarly, $z$ might coincide with $z^{\prime}$.

```
with \(f\) have \(x: x=x^{\prime} \vee x \sharp v\) to \(z^{\prime}\) in \(n\)
    and \(z: z=z^{\prime} \vee z \sharp n\)
    by (auto simp add: fresh-prod abs-fresh)
from \(f\) ch have \(x^{\prime}: x^{\prime} \sharp n \quad x^{\prime} \sharp z^{\prime}\)
    and \(x z^{\prime}: x=z^{\prime} \vee x \sharp n\)
    by (auto simp add:name-swap-bij alpha fresh-prod fresh-atm abs-fresh)
from \(f\) ch have \(x \sharp z x \sharp[z\rceil\). \(n\) by (auto simp add: fresh-prod)
with \(x z^{\prime} z\) have \(x \sharp\left(z,\left(\left[\left(z, z^{\prime}\right)\right] \cdot n\right)\right)\)
        by (auto simp add: fresh-atm fresh-bij name-swap-bij
            fresh-prod abs-fresh calc-atm fresh-aux fresh-left)
moreover from \(x\) ch have \(t=u\) to \(x\) in \(\left(\left[\left(x, x^{\prime}\right)\right] \cdot v\right)\)
    by (auto simp add:name-swap-bij alpha')
moreover from \(z\) ch have \(k=[z]\left(\left[\left(z, z^{\prime}\right)\right] \cdot n\right) \gg L\)
    by (auto simp add:name-swap-bij stack.inject alpha')
```

The first two $\alpha$-renamings are simple, but here we have to handle the nested binding structure of the assoc rule. Since $x$ scopes over the whole term $v$ to $z^{\prime}$ in $n$, we have to push the swapping over $z^{\prime}$

```
moreover \{ from \(x\) have
    \(u\) to \(x^{\prime}\) in \(\left(v\right.\) to \(z^{\prime}\) in \(\left.n\right)=u\) to \(x\) in \(\left(\left[\left(x, x^{\prime}\right)\right] \cdot\left(v\right.\right.\) to \(z^{\prime}\) in \(\left.\left.n\right)\right)\)
        by (auto simp add:name-swap-bij alpha' simp del: trm.perm)
    also from \(x z^{\prime} x^{\prime}\) have \(\ldots=u\) to \(x\) in \(\left(([(x, x)] \cdot v)\right.\) to \(z^{\prime}\) in \(\left.n\right)\)
        by (auto simp add: abs-fun-eq1 swap-simps alpha'")
            (metis alpha" fresh-atm perm-fresh-fresh swap-simps(1) \(x^{\prime}\) )
    also from \(z\) have \(\ldots=u\) to \(x\) in \(\left(\left(\left[\left(x, x^{\prime}\right)\right] \cdot v\right)\right.\) to \(z\) in \(\left.\left(\left[\left(z, z^{\prime}\right)\right] \cdot n\right)\right)\)
        by (auto simp add: abs-fun-eq1 alpha' name-swap-bij )
    finally
        have \(r=\left(u\right.\) to \(x\) in \(\left(\left(\left[\left(x, x^{\prime}\right)\right] \cdot v\right)\right.\) to \(z\) in \(\left.\left.\left(\left[\left(z, z^{\prime}\right)\right] \cdot n\right)\right)\right) \star L\)
            using ch by (simp del: trm.inject) \}
ultimately show \(P\)
    by (rule \(A\left[\right.\) where \(n=\left[\left(z, z^{\prime}\right)\right] \cdot n\) and \(\left.\left.v=\left(\left[\left(x, x^{\prime}\right)\right] \cdot v\right)\right]\right)\)
qed (insert \(r T K\), auto)
```

The lemma in Figure 2 assumes $S N(n[x::=p] \star K)$ but the actual induction in done on $S N(n \star K)$. The stronger assumption $S N(n[x::=p] \star K)$ is needed to handle the $\beta$ and $\eta$ cases.

```
lemma sn-forget:
    assumes a:SN(t[x::=v])
    shows SN t
proof -
    define q}\mathrm{ where q=t[x::=v]
    from a have SN q unfolding q-def .
    thus SN t using q-def
    proof (induct q arbitrary: t)
        case (SN-intro t)
        hence ih: \bigwedge t'. \llbrackett[x::=v]\mapsto t'[x::=v]\rrbracket\LongrightarrowSN 㰪 by auto
        { fix t' assume t\mapsto t'
            hence }t[x::=v]\mapsto\mp@subsup{t}{}{\prime}[x::=v] by (rule reduction-subst
            hence SN t' by (rule ih) }
        thus SNt ..
    qed
qed
lemma sn-forget':
    assumes sn:SN (t[x::=p]\star k)
    and x: x\sharpk
    shows }SN(t\stark
proof -
    from x have }t[x::=p]\stark=(t\stark)[x::=p] by (simp add: ssubst-forget
    with sn have SN( (t\stark)[x::=p]) by simp
    thus ?thesis by (rule sn-forget)
qed
```


## abbreviation

redrtrans $::$ trm $\Rightarrow$ trm $\Rightarrow$ bool $\left(-\mapsto^{*}-\right)$
where redrtrans $\equiv$ reduction ${ }^{* *}$
To be able to handle the case where $p$ makes a step, we need to establish $p \mapsto$ $p^{\prime} \Longrightarrow m[x::=p] \mapsto^{*} m[x::=p]$ as well as the fact that strong normalization is preserved for an arbitrary number of reduction steps. The first claim involves a number of simple transitivity lemmas. Here we can benefit from having removed the freshness conditions from the reduction relation as this allows all the cases to be proven automatically. Similarly, in the red-subst lemma, only those cases where substitution is pushed to two subterms needs to be proven explicitly.

```
lemma red-trans:
    shows r1-trans: \(s \mapsto^{*} s^{\prime} \Longrightarrow A p p s t \mapsto^{*} A p p s^{\prime} t\)
    and r2-trans: \(t \mapsto^{*} t^{\prime} \Longrightarrow A p p s t \mapsto^{*} A p p s t^{\prime}\)
    and r4-trans: \(t \mapsto^{*} t^{\prime} \Longrightarrow \Lambda x \cdot t \mapsto^{*} \Lambda x \cdot t^{\prime}\)
    and r6-trans: \(s \mapsto^{*} s^{\prime} \Longrightarrow s\) to \(x\) in \(t \mapsto^{*} s^{\prime}\) to \(x\) in \(t\)
    and \(r 7\)-trans: \(\llbracket t \mapsto^{*} t^{\prime} \rrbracket \Longrightarrow s\) to \(x\) in \(t \mapsto^{*} s\) to \(x\) in \(t^{\prime}\)
    and r11-trans: \(s \mapsto^{*} s^{\prime} \Longrightarrow[s] \mapsto^{*}\left(\left[s^{\prime}\right]\right)\)
by - (induct rule: rtranclp-induct, (auto intro:
transitive-closurep-trans')[2])+
lemma red-subst: \(p \mapsto p^{\prime} \Longrightarrow(m[x::=p]) \mapsto^{*}(m[x::=p])\)
proof(nominal-induct \(m\) avoiding: \(x \quad p p^{\prime}\) rule:trm.strong-induct)
    case \((A p p s t)\)
    hence \(\operatorname{App}(s[x::=p])(t[x::=p]) \mapsto^{*} \operatorname{App}(s[x::=p\rceil)(t[x::=p])\)
        by (auto intro: r1-trans)
```

```
    also from \(A p p\) have \(\ldots \mapsto^{*} \operatorname{App}(s[x::=p\rceil)(t[x::=p\rceil)\)
    by (auto intro: r2-trans)
    finally show ?case by auto
next
    case (To s y \(n\) ) hence
        \((s[x::=p])\) to \(y\) in \((n[x::=p]) \mapsto^{*}(s[x::=p])\) to \(y\) in \((n[x::=p])\)
        by (auto intro: r6-trans)
    also from \(T o\) have \(\ldots \mapsto^{*}(s[x::=p\rceil)\) to \(y\) in \((n[x::=p])\)
    by (auto intro: r7-trans)
    finally show ?case using To by auto
qed (auto intro:red-trans)
lemma \(S N\)-trans : \(\llbracket p \mapsto^{*} p^{\prime} ; S N p \rrbracket \Longrightarrow S N p^{\prime}\)
by (induct rule: rtranclp-induct) (auto intro: \(S N\)-preserved)
```


### 8.1 Central lemma

Now we have everything in place we need to tackle the central "Lemma 7 " of [LS05] (cf. Figure 2). The proof is quite long, but for the most part, the reasoning is that of [LS05].

```
lemma to-RED-aux:
    assumes \(p: S N p\)
    and \(x: x \sharp p \quad x \sharp k\)
    and \(n p k: S N(n[x::=p] \star k)\)
    shows \(S N(([p]\) to \(x\) in \(n) \star k)\)
proof -
    \{ fix \(q\) assume \(S N q\) with \(p\)
        have \(\bigwedge m . \llbracket q=m \star k ; S N(m[x::=p] \star k) \rrbracket\)
                        \(\Longrightarrow S N(([p]\) to \(x\) in \(m) \star k)\)
```

        using \(x\)
    proof (induct p q rule:triple-induct[where \(k=k]\) )
        case \((1 p q k)\) - We obtain an induction hypothesis for \(p, q\), and \(k\).
        have \(i h-p\) :
            \(\wedge p^{\prime} m . \llbracket p \mapsto p^{\prime} ; q=m \star k ; S N(m[x::=p] \star k) ; x \sharp p^{\prime} ; x \sharp k \rrbracket\)
                \(\Longrightarrow S N(([p]\) to \(x\) in \(m) \star k)\) by fact
            have \(i h-q\) :
            \(\bigwedge q^{\prime} m k . \llbracket q \mapsto q^{\prime} ; q^{\prime}=m \star k ; S N(m[x::=p] \star k) ; x \sharp p ; x \sharp k \rrbracket\)
                \(\Longrightarrow S N(([p]\) to \(x\) in \(m) \star k)\) by fact
            have \(i h-k\) :
    ```
                    \(\bigwedge k^{\prime} m . \llbracket\left|k^{\prime}\right|<|k| ; q=m \star k^{\prime} ; S N\left(m[x::=p] \star k^{\prime}\right) ; x \sharp p ; x \sharp k^{\prime} \rrbracket\)
                        \(\Longrightarrow S N\left(([p]\right.\) to \(x\) in \(\left.m) \star k^{\prime}\right)\) by fact
```

            have \(q: q=m \star k\) and \(s n: S N(m[x::=p] \star k)\) by fact +
            have \(x p\) : \(x \sharp p\) and \(x k: x \sharp k\) by fact+
    Once again we want to reason via case distinction on the successors of a term including a dismantling operator. Since this time we also need to handle the cases where interactions occur, we want to use the strengthened case rule. We already require $x$ to be suitably fresh. To instantiate the rule, we need another fresh name.

```
\{ fix \(r\) assume red: \(([p]\) to \(x\) in \(m) \star k \mapsto r\)
    from \(x p x k\) have \(x 1: x \sharp([p]\) to \(x\) in \(m) \star k\)
        by (simp add: abs-fresh)
    with red have \(x 2: x \sharp r\) by (rule reduction-fresh)
    obtain \(z:: n a m e\) where \(z: z \sharp(x, p, m, k, r)\)
    using ex-fresh[of ( \(x, p, m, k, r\) )] by (auto simp add: fresh-prod)
    have \(S N r\)
```

```
proof (cases rule:dismantle-strong-cases
    [of [p] to x in m krexx z ])
case (5 r') have r:r= r'\star k and r': [p] to x in m\mapsto 盾古y fact+
```

To handle the case of a reduction occurring somewhere in $[p]$ to $x$ in $m$, we need to contract the freshness conditions to this subterm. This allows the use of the strong inversion rule for the reduction relation.

```
from \(x 1 x 2 r\)
have \(x l:(x \sharp[p]\) to \(x\) in \(m)\) and \(x r: x \sharp r^{\prime}\) by auto
from \(z\) have \(z l: z \sharp([p]\) to \(x\) in \(m) \quad x \neq z\)
    by (auto simp add: abs-fresh fresh-prod fresh-atm)
with \(r^{\prime}\) have \(z r: z \sharp r^{\prime}\) by (blast intro:reduction-fresh)
- handle all reductions of \([p]\) to \(x\) in \(m\)
from \(r^{\prime}\) show \(S N r\) proof (cases rule:reduction.strong-cases
    [where \(x=x\) and \(x a=x\) and \(x b=x\) and \(x c=x\) and \(x d=x\)
    and \(x e=x\) and \(x f=x\) and \(x g=x\) and \(y=z])\)
```

The case where $p \mapsto p^{\prime}$ is interesting, because it requires reasioning about the reflexive transitive closure of the reduction relation.

```
case ( \(r 6 s s^{\prime} t\) ) hence \(c h:[p] \mapsto s^{\prime} \quad r^{\prime}=s^{\prime}\) to \(x\) in \(m\)
    using \(x l x r\) by (auto)
from this obtain \(p^{\prime}\) where \(s: s^{\prime}=\left[p^{\prime}\right]\) and \(p: p \mapsto p^{\prime}\)
    by (blast dest:red-Ret)
from \(p\) have \(((m \star k)[x::=p]) \mapsto^{*}((m \star k)[x::=p])\)
    by (rule red-subst)
    with \(x k\) have \(((m[x::=p]) \star k) \mapsto^{*}((m[x::=p\rceil) \star k)\)
    by (simp add: ssubst-forget)
    hence \(s n\) : \(S N((m[x::=p]) \star k)\) using \(s n\) by (rule \(S N\)-trans)
    from \(p x p\) have \(x p^{\prime}: x \sharp p^{\prime}\) by (rule reduction-fresh)
    from \(c h s\) have \(r r: r^{\prime}=\left[p^{\prime}\right]\) to \(x\) in \(m\) by simp
    from \(p q\) sn \(x p^{\prime} x k\)
    show \(S N r\) unfolding \(r r r\) by (rule \(i h-p\) )
next
```

```
    case \(\left(r 7\right.\) s \(\left.t m^{\prime}\right)\) hence \(r^{\prime}=[p]\) to \(x\) in \(m^{\prime}\) and \(m \mapsto m^{\prime}\)
    using \(x l\) xr by (auto simp add: alpha)
    hence \(r r: r^{\prime}=[p]\) to \(x\) in \(m^{\prime}\) by simp
    from \(q\left\langle m \mapsto m^{\prime}\right\rangle\) have \(q \mapsto m^{\prime} \star k \mathbf{b y}\) (simp add: dismantle-red)
    moreover have \(m^{\prime} \star k=m^{\prime} \star k\).. - a triviality
    moreover \{ from \(\left\langle m \mapsto m^{\prime}\right\rangle\) have \((m[x::=p]) \star k \mapsto\left(m^{\prime}[x::=p]\right) \star k\)
        by (simp add: dismantle-red reduction-subst)
    with \(s n\) have \(\left.S N\left(m^{\prime}[x::=p] \star k\right) ..\right\}\)
    ultimately show \(S N r\) using \(x p x k\) unfolding \(r r r\) by (rule ih-q)
next
```

    case \((r 8 s t)\) - the \(\beta\)-case is handled by assumption
    hence \(r^{\prime}=m[x::=p]\) using xl \(x r\) by (auto simp add: alpha)
    thus \(S N r\) unfolding \(r\) using \(s n\) by simp
    next
case (r9 s) - the $\eta$-case is handled by assumption as well
hence $m=[\operatorname{Var} x]$ and $r^{\prime}=[p]$ using $x l x r$
by (auto simp add: alpha)
hence $r^{\prime}=m[x::=p]$ by simp
thus $S N r$ unfolding $r$ using sn by simp
qed (simp-all only: xr xl zl zr abs-fresh, auto)
— There are no other possible reductions of $[p]$ to $x$ in $m$. next

```
case (6 k}
have k:k\mapsto 乍' and r:r=([p] to x in m)\star k' by fact+
from qk have q}\mapstom\star k' unfolding stack-reduction-def by blas
moreover have m\star k'=m\star 乍'.
moreover { have SN (m[x::=p]\star k) by fact
    moreover have (m[x::=p])\stark\mapsto(m[x::=p])\star k'
            using k unfolding stack-reduction-def ..
    ultimately have SN (m[x::=p]\star k') .. }
    moreover note xp
    moreover from kxk have }x\sharp\mp@subsup{k}{}{\prime
    by (rule stack-reduction-fresh)
    ultimately show SN r unfolding r by (rule ih-q)
next
```

The case of an assoc interaction between $[p]$ to $x$ in $m$ and $k$ is easily handled by the induction hypothesis, since $m[x::=p] \star k$ remains fixed under assoc.

```
    case (8 s t uL)
    hence k: k= [z]u>>L
    and}r:r=([p] to x in (m to z in u))\star
    and u:x\sharpu
    by(auto simp add: alpha fresh-prod)
let ? }k=L\mathrm{ and ?m=m to z in u
from kz have |?k|<|k| by (simp add: fresh-prod)
moreover have q=?m\star ? k using k q by simp
moreover { from kuzxp have (?m[x::=p]\star ?k) = (m[x::=p])\stark
    by(simp add: fresh-prod forget)
hence SN (?m[x::=p]\star ?k) using sn by simp }
moreover from xp xk k have }x\sharpp\mathrm{ and }x\sharp??k\mathrm{ by auto
ultimately show SN r unfolding r by (rule ih-k)
qed (insert red z x1 x2 xp xk,
    auto simp add: fresh-prod fresh-atm abs-fresh)
    } thus SN (([p] to x in m)\star k)..
    qed }
    moreover have SN ((n[x::=p])\star k) by fact
    moreover hence SN (n\stark) using \langlex \sharpk\rangle by (rule sn-forget')
    ultimately show ?thesis by blast
qed
```

Having established the claim above, we use it show that to-bindings preserve reducibility.

```
lemma to-RED:
    assumes \(s: s \in R E D(T \sigma)\)
    and \(t: \forall p \in R E D \sigma . t[x::=p] \in R E D(T \tau)\)
    shows \(s\) to \(x\) in \(t \in R E D(T \tau)\)
proof -
    \{ fix \(K\) assume \(k: K \in S R E D \tau\)
            \(\{\) fix \(p\) assume \(p: p \in R E D \sigma\)
        hence snp: SN p using RED-props by(simp add: CR1-def)
        obtain \(x^{\prime}::\) name where \(x: x^{\prime} \sharp(t, p, K)\)
            using ex-fresh[of ( \(t, p, K)]\) by (auto)
        from \(p t k\) have \(S N((t[x::=p]) \star K)\) by auto
        with \(x\) have \(S N\left(\left(\left(\left[\left(x^{\prime}, x\right)\right] \cdot t\right)\left[x^{\prime}::=p\right]\right) \star K\right)\)
```

```
            by (simp add: fresh-prod subst-rename)
            with snp x have snx': SN (([p] to x' in ([(\mp@subsup{x}{}{\prime},x)] \cdot t ))\star *)
                by (auto intro: to-RED-aux)
            from x have [p] to x' in ([(x',x)] • t) = [p] to x in t
                by simp (metis alpha' fresh-prod name-swap-bij x)
            moreover have ([p] to x in t)\star K=[p]\star \x]t>>K by simp
            ultimately have snx:SN([p]\star [x]t>>K) using snx'
                by (simp del: trm.inject)
    } hence [x]t>>K S SRED \sigma by simp
    with s have SN((s to x in t)\star K) by(auto simp del: SRED.simps)
    } thus s to }x\mathrm{ in t f RED (T T) by simp
qed
```


## 9 Fundamental Theorem

The remainder of this section follows [Nom] very closely. We first establish that all well typed terms are reducible if we substitute reducible terms for the free variables.

## abbreviation

mapsto $::($ name $\times$ trm $)$ list $\Rightarrow$ name $\Rightarrow$ trm $\Rightarrow$ bool (- maps - to $-[55,55,55] 55)$
where
$\vartheta$ maps $x$ to $e \equiv($ lookup $\vartheta x)=e$

## abbreviation

```
    closes :: \((\) name \(\times\) trm \()\) list \(\Rightarrow(\) name \(\times\) ty \()\) list \(\Rightarrow\) bool (- closes \(-[55,55] 55)\)
```

where
$\vartheta$ closes $\Gamma \equiv \forall x \tau .((x, \tau) \in$ set $\Gamma \longrightarrow(\exists t . \vartheta$ maps $x$ to $t \wedge t \in R E D \tau))$
theorem fundamental-theorem:
assumes $a: \Gamma \vdash t: \tau$ and $b: \vartheta$ closes $\Gamma$
shows $\vartheta<t>\in R E D \tau$
using $a b$
proof(nominal-induct avoiding: $\vartheta$ rule: typing.strong-induct)
case (t3 a $\Gamma \sigma t \tau \vartheta)$ — lambda case
next
case ( $t 5 x \Gamma s \sigma t \tau \vartheta$ ) - to case
have ihs : $\bigwedge \vartheta . \vartheta$ closes $\Gamma \Longrightarrow \vartheta<s>\in R E D(T \sigma)$ by fact
have iht $: \bigwedge \vartheta . \vartheta$ closes $((x, \sigma) \# \Gamma) \Longrightarrow \vartheta<t>\in R E D(T \tau)$ by fact
have $\vartheta$-cond: $\vartheta$ closes $\Gamma$ by fact
have fresh: $x \sharp \vartheta x \sharp \Gamma \quad x \sharp s$ by fact +
from ihs have $\vartheta<s>\in R E D(T \sigma)$ using $\vartheta$-cond by simp
moreover
\{ from iht have $\forall s \in R E D \sigma .((x, s) \# \vartheta)<t>\in R E D(T \tau)$
using fresh $\vartheta$-cond fresh-context by simp
hence $\forall s \in R E D \sigma . \vartheta<t>[x::=s] \in R E D(T \tau)$
using fresh by (simp add: psubst-subst) \}
ultimately have $(\vartheta<s>)$ to $x$ in $(\vartheta<t>) \in R E D(T \tau)$ by (simp only: to-RED)
thus $\vartheta<s$ to $x$ in $t>\in R E D(T \tau)$ using fresh by simp
qed auto - all other cases are trivial
The final result then follows using the identity substitution, which is $\Gamma$-closing since all variables are reducible at any type.
fun
id $::($ name $\times$ ty $)$ list $\Rightarrow($ name $\times$ trm $)$ list
where
id []$=[]$
$\mid i d((x, \tau) \# \Gamma)=(x, \operatorname{Var} x) \#(i d \Gamma)$
lemma id-maps:
shows (id $\Gamma$ ) maps a to (Var a)
by (induct $\Gamma$ ) (auto)
lemma $i d$-fresh:
fixes $x$ ::name
assumes $x: x \sharp \Gamma$
shows $x \sharp(i d \Gamma)$
using $x$
by (induct $\Gamma$ ) (auto simp add: fresh-list-nil fresh-list-cons)
lemma id-apply:
shows $(i d \Gamma)<t>=t$
by (nominal-induct t avoiding: $\Gamma$ rule: trm.strong-induct)
(auto simp add: id-maps id-fresh)
lemma id-closes:
shows (id $\Gamma$ ) closes $\Gamma$
proof -
\{ fix $x \tau$ assume $(x, \tau) \in$ set $\Gamma$
have CR4 $\tau$ by (simp add: RED-props CR3-implies-CR4)
hence Var $x \in R E D \tau$
by (auto simp add: NEUT-def normal-var CR4-def)
hence (id $\Gamma$ ) maps $x$ to Var $x \wedge \operatorname{Var} x \in R E D \tau$
by (simp add: id-maps)
\} thus ?thesis by blast
qed

### 9.1 Strong normalization theorem

lemma typing-implies-RED:
assumes $a$ : $\Gamma \vdash t: \tau$
shows $t \in R E D \tau$
proof -
have $(i d \Gamma)<t>\in R E D \tau$
proof -
have (id $\Gamma$ ) closes $\Gamma$ by (rule id-closes)
with $a$ show ?thesis by (rule fundamental-theorem)
qed
thust $\in R E D \tau$ by (simp add: id-apply)
qed
theorem strong-normalization:
assumes $a: \Gamma \vdash t: \tau$
shows $S N(t)$
proof -
from $a$ have $t \in R E D \tau$ by (rule typing-implies-RED)
moreover have CR1 $\tau$ by (rule RED-props)
ultimately show $S N(t)$ by (simp add: CR1-def)
qed
This finishes our formalization effort. This article is generated from the Is-
abelle theory file, which consists of roughly 1500 lines of proof code. The reader is invited to replay some of the more technical proofs using the theory file provided.

## Acknowledgments

I thank Christian Urban, the Nominal Methods group, and the members of the Isabelle mailing list for their helpful answers to my questions.

## References

[DS09] Christian Doczkal and Jan Schwinghammer. Formalizing a Strong Normalization proof for Moggi's Computational Metalanguage: A Case Study in Isabelle/HOL-Nominal. In LFMTP '09: Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages, pages 57-63, New York, NY, USA, 2009. ACM.
[GTL89] Jean-Yves Girard, Paul Taylor, and Yves Lafont. Proofs and Types. Cambridge University Press, New York, NY, USA, 1989.
[LS05] Samuel Lindley and Ian Stark. Reducibility and TT-lifting for Computation Types. In Proceedings of Typed Lambda Calculi and Applications (TLCA '05), volume 3461 of Lecture Notes in Computer Science, pages 262-277. Springer, Apr 2005.
[Nom] Nominal Methods Group. Strong normalisation proof from the proofs and types book. http://isabelle.in.tum.de/dist/library/ HOL/HOLNominal/Examples/SN.html.
[Urb08] Christian Urban. Nominal Techniques in Isabelle/HOL. J. Autom. Reason., 40(4):327-356, 2008.


[^0]:    ${ }^{1}$ This possibility was only discovered after having formalized $K \mapsto K^{\prime} \Rightarrow|K| \geq\left|K^{\prime}\right|$. The proof of this seemingly simple fact was about 90 lines of Isar code.

