Formalizing a Strong Normalization Proof for Moggi's Computational Metalanguage

A case study in Isabelle/HOL-Nominal

Christian Doczkal [*]   Jan Schwinghammer

Saarland University, Saarbrücken, Germany

Abstract

Lindley and Stark have given an elegant proof of strong normalization for various lambda calculi whose type systems preclude a direct inductive definition of Girard-Tait style logical relations, such as the simply typed lambda calculus with sum types or Moggi's calculus with monadic computation types. The key construction in their proof is a notion of relational $\top\top$-lifting, which is expressed with the help of stacks of evaluation contexts. We describe a formalization of Lindley and Stark's strong normalization proof for Moggi's computational metalanguage in Isabelle/HOL, using the nominal package.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory; F.4.1 [*Mathematical Logic and Formal Languages*]: Mathematical Logic – *mechanical theorem proving*

General Terms Theory, verification

Keywords HOL-Nominal, logical relations and $\top\top$-lifting

1. Introduction

Proofs of (strong) normalization for lambda calculi have long been used as case studies for the formalization of programming language meta-theory. An early example is the strong normalization proof for System F by Altenkirch (1993), other examples include those by Abel (2004), Berger et al. (2006), Donnelly and Xi (2007), and Schürmann and Sarnat (2008).

Normalization proofs provide interesting case studies for formalization because they combine syntactic as well as more semantic arguments about terms and reduction: just like proofs of type safety by 'progress and preservation,' in a formalization one must deal with variable binding, renaming and substitution, but usually one also uses various induction and strong reasoning principles (for instance, to define logical relations). Moreover, these proofs are of moderate size, and their structure is generally well-understood. Thus, they provide a good point of reference for comparing a formalization to the a paper-and-pencil version of the proof, as well as for comparing formalizations of the proof in different proof assistants, and using different paradigms, to each other.

In the classic Girard-Tait reducibility method for proving normalization, a family of sets of 'reducible' terms is defined by induction on types (Girard et al. 1989). For instance, a term of function type $\sigma \rightarrow \tau$ is reducible if and only if its application to reducible arguments (at type σ) yields a reducible term (at type τ). Adapting these definitions to lambda calculi with a richer type structure can be challenging. The computational metalanguage of Moggi (1991) provides an example of such a calculus: its types contain 'computation types' $T\sigma$, of computations that return values of type σ. The only way to deconstruct terms s of this type is by sequencing, $s \mathsf{~to~} x \mathsf{~in~} t$, which binds x in t to the result of s. (We use this notation, instead of $\mathsf{let}\, x{\Leftarrow}s \,\mathsf{in}\, t$, as it corresponds to the scoping of x.) Since the type $T\tau$ of t is in general not smaller than the type $T\sigma$ of s, the definition of reducibility at type $T\sigma$ cannot refer to reducibility of terms at type $T\tau$. A similar problem arises in the lambda calculus with sum types, where the *case* construct for sum types prevents a straightforward inductive definition of the reducibility predicates (*cf.* Prawitz 1971).

Lindley and Stark (2005) present an elegant strong normalization proof for Moggi's computational metalanguage. The difficulty of defining a logical relation for computation types described above is addressed by using a notion of $\top\top$-lifting of predicates. The basic idea is as follows: Given a stack of elimination constructs for the computation type, $k = (\ldots([\cdot] \mathsf{~to~} x_1 \mathsf{~in~} t_1)\ldots) \mathsf{~to~} x_k \mathsf{~in~} t_k$, one can plug in a term t to observe if the resulting term $t \star k$ is strongly normalizing. Asserting that $[t] \star k$ is strongly normalizing for all reducible terms t of type τ (where $[t]$ is the trivial computation immediately returning t) defines a notion of reducibility for $T\tau$-expecting stacks. Then, a term t is defined to be reducible at type $T\tau$ whenever $t \star k$ is strongly normalizing for all reducible $T\tau$-expecting stacks k. Using this indirection via stacks, reducibility at type $T\tau$ is obtained from reducibility at type τ, giving rise to an inductive definition of this type-indexed family of predicates with properties sufficient to establish strong normalization. Lindley and Stark's technique is interesting because it also handles 'commuting conversions,' like associativity for nested to-bindings in Moggi's calculus, and it is sufficiently robust to adapt to, e.g., sum types.

In this paper, we describe a formalization of the published normalization proof (Lindley and Stark 2005), as a case study similar in spirit to the formalized logical relations proofs by Narboux and Urban (2008). Apart from the obvious question whether the paper-and-pencil proof is correct (of which there was little doubt), we are interested in how close we can stay to the original published proof of Lindley and Stark (2005). Answering this question

[*] Support has been provided by a scholarship from the Saarbrücken Graduate School of Computer Science.

will give some further indication of how well the Nominal datatype package achieves the goal of permitting *faithful* formalizations of informal reasoning about languages with binding constructs, and it may point to (current) short-comings and possible areas for improvements. We believe that a closer look at Lindley and Stark's proof is interesting for the following reasons. First, the calculus of Moggi (1991) is slightly more complex than simply typed lambda calculus: it has a second binding construct, and its commuting conversions provide an example that requires non-trivial variable freshness conditions, as these conversions rearrange the scoping. Second, the formalization of stacks introduces a second datatype with binding, and the interaction between stacks and terms requires more complex patterns of recursion and induction compared to the simply typed case.

The next section gives a brief reminder of the work on nominal logic, which provides the foundation for the nominal package in Isabelle/HOL. Section 3 gives an overview of our formalization, focussing on those parts of the strong normalization proof that are specific to the metalanguage. (The complete proof document can be found at `http://www.ps.uni-sb.de/Papers`) In Section 4, we discuss some aspects of this formalization more generally.

2. Nominal Logic

We recall the basic notions from work on nominal logic, on which the Isabelle nominal datatype package is based (Pitts 2003, 2006; Urban 2008), and which provides a formal justification for the Barendregt variable convention used in informal proofs: one can always find variables that are 'fresh' for the current context, and the reasoning is independent of the particular choice of representatives of α-equivalence classes.

Permutations, support and freshness We fix a countably infinite set *name* of atomic names, which are used to represent object level variables. Finite permutations of names can be represented as (finite) lists π of transpositions $[(a_1\, b_1), (a_2\, b_2), \ldots, (a_k\, b_k)]$, and the application $\pi \cdot a$ of π to a name a is defined by induction on this list in the evident way. The finite permutations on *name* form a group under composition, with a representation of the unit *id* given by the empty list, composition represented by list concatenation, and inverses by list reversal. The operation $\pi \cdot a$ is a special case of a group action on a set X that is compatible with the list representation of permutations: $id \cdot x = x$ and $(\pi_1 \circ \pi_2) \cdot x = \pi_1 \cdot (\pi_2 \cdot x)$, and if $\pi_1 \cdot a = \pi_2 \cdot a$ for all $a \in$ *name* then also $\pi_1 \cdot x = \pi_2 \cdot x$, for all π_1, π_2 and all $x \in X$. Such a permutation action determines the *support* of the elements of X:

$$a \in \mathit{supp}(x) \overset{def}{\Leftrightarrow} \{b \in \mathit{name} \mid (a\, b) \cdot x \neq x\} \text{ is infinite}$$

A *nominal set* is given by an action on X such that *supp*(x) is finite for all $x \in X$. A name a is *fresh for* x, written $a\sharp x$, if $a \notin \mathit{supp}(x)$. In particular, for every element x of a nominal set there exists a fresh name $a\sharp x$. The support generalizes the usual notion of free variable of a term: an important example of a nominal set is given by terms modulo α-equivalence, which can be equipped with a permutation action such that *supp* coincides with the free variables.

Nominal datatypes in Isabelle/HOL There are two approaches to achieve finite supportedness in an implementation. First, it is possible to work in a logic that only permits the description of finitely supported objects by construction. This gives the existence of fresh names for free but is inconsistent with several standard libraries, notably those using choice (Pitts 2006, Example 3.4). In contrast, the approach of Urban (2008) is to work in the standard higher-order logic of Isabelle/HOL, at the expense of additional proof obligations whenever the finite support property is needed. In the implementation this is alleviated by making good use of Isabelle's axiomatic type class mechanism (Haftmann 2009): in most practically relevant cases, the finite support property is inferred automatically from the type of an object. This includes in particular nominal datatypes and tuples thereof.

A function f between nominal sets is *equivariant* if $\pi \cdot (f\, x) = f(\pi \cdot x)$ for all π and x, and a predicate P is equivariant if its characteristic function is, i.e., if $P(x) \Leftrightarrow P(\pi \cdot x)$. The equivariance of relations is one of the prerequisites for the automatic derivation of (strong) induction and inversion principles by the nominal package (Urban et al. 2007; Berghofer and Urban 2008), and is usually assumed in informal proofs.

3. Formal Development

3.1 Terms and Types

The λ_{ml} calculus is an extension of the simply typed λ-calculus with the additional type constructor T and the following terms:

$$\frac{\Gamma \vdash t : \tau}{\Gamma \vdash [t] : T\tau} \qquad \frac{\Gamma \vdash s : T\sigma \qquad \Gamma, x{:}\sigma \vdash t : T\tau}{\Gamma \vdash s \,\mathsf{to}\, x \,\mathsf{in}\, t : T\tau}$$

We deviate from Lindley and Stark (2005) by using typing contexts instead of explicitly typed variables. This allows us to carry out a large part of the normalization proof on untyped terms. Terms and types are represented using nominal datatypes:

nominal-datatype *trm* =
  Var name
  | *App trm trm*
  | *Lam* ≪*name*≫*trm*
  | *To trm* ≪*name*≫*trm* (- *to* - *in* -)
  | *Ret trm* ([-])

nominal-datatype *ty* =
  TBase
  | *TFun ty ty* (**infix** → *200*)
  | *T ty*

The notation ≪*name*≫*trm* means that, e.g. in the lambda case *Lam* $x\, t$, the variable x is bound. Equality on *trm* is α-equivalence of the abstract syntax trees, and the permutation action that is derived from the **nominal-datatype** declaration respects this equivalence. It satisfies $\pi \cdot (\mathit{Lam}\, x\, t) = \mathit{Lam}\, (\pi \cdot x)\, (\pi \cdot t)$, and the support of a term is just its set of free variables. In particular, we have $\mathit{Lam}\, x\, t = (x\, y) \cdot (\mathit{Lam}\, x\, t) = \mathit{Lam}\, y\, ((x\, y) \cdot t)$ for any y that is not free in *Lam* $x\, t$.

Substitution on terms can be defined just as one would do on paper, using the primitive recursion operator for nominal data types. For the binder cases one may assume that the binder is suitably fresh, and the nominal package provides very good support for showing that primitive recursive functions are well defined even in the presence of freshness requirements. This is all quite standard (Urban 2008). In our case (ordinary) substitution $t[x ::= s]$ is defined as a special case of parallel substitution, written $\theta{<}t{>}$, which we need to state the Fundamental Theorem below.

The typing relation is a straightforward translation of the usual typing rules in the style of Narboux and Urban (2008). We use lists to implement contexts; in contrast to functions, finite lists over a type of finitely supported objects have finite support. Here are the cases corresponding to the two typing rules shown above:

inductive
  typing :: (*name*×*ty*) *list*⇒*trm*⇒*ty*⇒*bool* (- ⊢ - : -)
where [...]
| *t4*[*intro*]: ⟦ Γ ⊢ *s* : *σ* ⟧ ⟹ Γ ⊢ [*s*] : *T σ*
| *t5*[*intro*]: ⟦*x* ♯ (Γ,*s*); Γ ⊢ *s* : *T σ* ; ((*x*,*σ*)#Γ) ⊢ *t* : *T τ* ⟧
    ⟹ Γ ⊢ *s to x in t* : *T τ*

3.2 Reduction and Strong Normalization

We use the standard $\beta\eta$-rules of the simply typed lambda calculus, and define reductions in an arbitrary context by rules of the form

$$\frac{t \mapsto t' \qquad x \sharp s}{s \,\mathsf{to}\, x \,\mathsf{in}\, t \mapsto s \,\mathsf{to}\, x \,\mathsf{in}\, t'} \qquad (1)$$

Furthermore, we have the following new reduction rules:

$$
\begin{array}{lr}
[s]\,\mathsf{to}\,x\,\mathsf{in}\,t \mapsto t[x ::= s] & x \sharp s \\
s\,\mathsf{to}\,x\,\mathsf{in}[x] \mapsto s & x \sharp s \\
(s\,\mathsf{to}\,x\,\mathsf{in}\,t)\,\mathsf{to}\,y\,\mathsf{in}\,u \mapsto s\,\mathsf{to}\,x\,\mathsf{in}(t\,\mathsf{to}\,y\,\mathsf{in}\,u) & x\sharp(s,y,u), y\sharp(s,t)
\end{array}
$$

In these rules, the freshness conditions on s and t are only needed to make the reduction relation *vc-compatible*, a condition which permits the automatic derivation of strong induction and inversion principles (Urban et al. 2007; Berghofer and Urban 2008). Using some explicit α-renaming, one can however show adequacy of the vc-compatible formulation of the reduction relation with respect to the standard presentation, which omits all these freshness requirements (except the ones on y and u in the final rule).

For the reduction relation, we define an inductive variant of strong normalization (*cf.* Altenkirch 1993), given by the single inference rule[1]:

$$\frac{\bigwedge t'.\ t \mapsto t' \Longrightarrow SN\,t'}{SN\,t} \tag{2}$$

This definition of *SN* is convenient because it gives an induction principle for strongly normalizing terms: to prove that P holds for all strongly normalizing terms it suffices to show $P(t)$ from the assumption that $P(t')$ for all $t \mapsto t'$. The alternative, used in the proofs by Lindley and Stark, are inductions on the length $\max(t)$ of a longest reduction sequence starting from a strongly normalizing term t, for which we would need to establish first that the reduction relation is finitely branching.

We can relate the definition of *SN* in (2) to one in phrased in terms of reduction sequences. There is an easy inductive argument showing that $SN(t)$ implies the absence of infinite reduction sequences from t. There is also a fairly simple proof of the converse, using the axiom of choice – without choice, one needs to work harder and use more specific properties of the reduction relation.

3.3 Stacks and Dismantling

We declare stacks of sequencing constructs as lists of term abstractions, exploiting the fact that term abstractions are primitive to HOL-Nominal.

nominal-datatype *stack* = *Id* | *St* ≪*name*≫*trm stack* ([-]-≫-)

We also need the associated function of *stack dismantling* $t \star k$ mentioned in the Introduction, which plugs the term t into the context of nested sequencing constructs represented by k. It satisfies:

$$t \star Id = t$$
$$t \star ([x]n \gg k) = (t\,\mathsf{to}\,x\,\mathsf{in}\,n) \star k$$

Note that $\star$ is not in the form of a primitive recursive function definition. Since the nominal package has no support for general recursive functions involving binders, this means that one has to manually prove pattern completeness, right-uniqueness and termination.

As in (Lindley and Stark 2005), the reduction relation on terms induces a notion of reduction on stacks, also denoted by $\mapsto$.

constdefs
stack-reduction :: *stack* ⇒ *stack* ⇒ *bool* (- ↦ -)
$k \mapsto k' \equiv \forall\ (t{::}trm)\,.\,(t \star k) \mapsto (t \star k')$

We can then define strong normalization also for stacks, using the same single rule inductive definition that we used to define *SN* on terms in (2) above. We call this predicate *SSN*.

[1] $\bigwedge$ and $\Longrightarrow$ denote the universal quantifier and implication at the meta level. The latter also corresponds to the line in rule notation.

3.4 Reducibility of Terms and Stacks

We formalize the logical relation as mutually recursive functions from types to sets of terms and stacks, respectively:

function *RED* :: *ty* ⇒ *trm set* **and** *SRED* :: *ty* ⇒ *stack set*
where
$RED\ (TBase) = \{t.\ SN(t)\}$
$|\ RED\ (\tau \to \sigma) = \{t.\ \forall\ u \in RED\ \tau\ .\ (App\ t\ u) \in RED\ \sigma\ \}$
$|\ RED\ (T\ \sigma) = \{t.\ \forall\ k \in SRED\ \sigma\ .\ SN(t \star k)\ \}$
$|\ SRED\ \tau = \{k.\ \forall\ t \in RED\ \tau\ .\ SN\ ([t] \star k)\ \}$

As sketched in the introduction, $RED_{T\sigma}$ implements the $\top\top$-lifting with help of the auxiliary relation $SRED_\sigma$. Informally, the latter picks out a subset of stacks, which then provide 'tests' with respect to which the normalization of terms at computation type $T\sigma$ is observed.

Showing that the logical relation is well defined amounts to proving that *RED* and *SRED* terminate. This is established through the termination measure $2\cdot|\tau|$ for arguments to *RED*, and $2\cdot|\tau|+1$ for arguments to *SRED*. The properties of the reducibility relation we are interested in are the usual ones (Girard et al. 1989).

constdefs
$NORMAL\ t \equiv \neg(\exists\, t'.\ t \mapsto t')$
$NEUT\ t \equiv (\exists\, x.\ t = Var\ x) \lor (\exists\, u\ v.\ t = App\ u\ v)$
$CR1\ \tau \equiv \forall\, t.\ (t{\in}RED\ \tau \longrightarrow SN\ t)$
$CR2\ \tau \equiv \forall\, t\ t'.\ (t{\in}RED\ \tau \land t \mapsto t') \longrightarrow t'{\in}RED\ \tau$
$CR3\text{-}RED\ t\ \tau \equiv \forall\, t'.\ t \mapsto t' \longrightarrow\ t'{\in}RED\ \tau$
$CR3\ \tau \equiv \forall\, t.\ (NEUT\ t \land CR3\text{-}RED\ t\ \tau) \longrightarrow t{\in}RED\ \tau$
$CR4\ \tau \equiv \forall\, t.\ (NEUT\ t \land NORMAL\ t) \longrightarrow t{\in}RED\ \tau$

These properties state that reducibility entails strong normalization (*CR1*), and that reducibility is closed under reduction (*CR2*) and expansion (*CR3*).

Property *CR4* is an easy consequence of *CR3*. The proof that *CR1-3* hold for all types is done by mutual induction on the type structure. In fact, due to the modular nature of logical relations proofs we could reuse the corresponding cases from the normalization proof for the simply typed lambda calculus, which is included as an example in the nominal datatype package (Nominal Methods group 2009). Thus, we only need to cover the case of the monadic type constructor T. We consider this case in detail for *CR3*. Under the assumption that *CR1-4* hold for RED_σ, we show:

If $NEUT(t)$ and $t \mapsto t'$ implies $t' \in RED_{T\sigma}$, then $t \in RED_{T\sigma}$.

We first show the published paper-and-pencil proof (Lindley and Stark 2005, Theorem 5) of this case, and then discuss the challenges in formalizing it.

The proof of Lindley and Stark *Let t be neutral such that $t' \in RED_{T\sigma}$ whenever $t \mapsto t'$. We have to show that $(t \star k)$ is SN for each $k \in SRED_\sigma$. First, we have that $[x] \star k$ is SN, as $x \in RED_\sigma$ by the induction hypothesis. Hence k itself is SN, and we can work by induction on $\max(k)$. Application $t \star k$ may reduce as follows:*

- *$t' \star k$, where $t \mapsto t'$, which is SN as $k \in SRED_\sigma$ and $t' \in RED_{T\sigma}$.*
- *$t \star k'$, where $k \mapsto k'$. For any $s \in RED_\sigma$, $[s] \star k$ is SN as $k \in SRED_\sigma$; and $[s] \star k \mapsto [s] \star k'$, so $[s] \star k'$ is also SN. From this we have $k' \in SRED_\sigma$ with $\max(k') < \max(k)$, so by induction hypothesis $t \star k'$ is SN.*

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

In this proof, $\max(k)$ denotes the length of a longest sequence of stack reductions beginning at k.

Formalized proof First, we note that the general reasoning, i.e., showing strong normalization of $t \star k$ by showing that all successors are SN, directly corresponds to our introduction rule for *SN*. Next, the sub-induction on $\max(k)$ can be handled more directly using the induction principle for *SSN*. To apply the latter, we need to establish that $SN(t \star k)$ implies $SSN(k)$. The main difficulty is the extraction of k, since $t \star k$ does not uniquely determine k. To achieve this, we define a single rule inductive relation $t \star k \rhd k$, and then show the more general claim that $SN(t \star k)$ implies $\forall z. t \star k \rhd z \longrightarrow SN(z)$. This is (a simplification of) a trick used by the Nominal Methods group (2009) to address a similar problem in the application case of the normalization proof for simply typed lambda calculus, where one needs to extract s from the compound term *App* s t. In the case of *App*, extraction is a (partial) function on terms, but the method equally applies in the case of $\star$ where extraction becomes a proper relation.

The major challenge in the formalization of the informal proof above is the sentence concluding the case analysis, "*There are no other possibilities*." Its formalization requires an exhaustive case analysis on the successors of $t \star k$. To enhance modularity and readability, this case analysis is established separately as a rule:

$$\frac{\begin{array}{ll} t \star k \mapsto r & \bigwedge t'.\, [\![\, t \mapsto t';\, r = t' \star k]\!] \Longrightarrow P \\ NEUT\, t & \bigwedge k'.\, [\![\, k \mapsto k';\, r = t \star k']\!] \Longrightarrow P \end{array}}{P} \tag{3}$$

We want to prove soundness of this rule by induction on the structure of k, and the case $k = Id$ is trivial. In the case where $k = [y]n \gg l$ the idea is to unfold the operation $t \star k$ once and move the first stack frame of k onto t, and we would like to apply the induction hypothesis with $t^* = t\,\mathsf{to}\,y\,\mathsf{in}\,n$ and l. However, t^* clearly is *not* neutral, so we have to generalize the claim. More precisely, we obtain a rule *dismantle-cases* by replacing $NEUT(t)$ in (3) with the following two hypothesis about the possible interactions of t and k:

```
⋀ s y n l .⟦ t = [s] ; k = [y]n≫l ; r = (n[y::=s]) ⋆ l ⟧ ⟹ P
⋀ u x v y n l.⟦ x ♯ (y, n) ; t = u to x in v ;
        k = [y]n≫l ; r = (u to x in (v to y in n)) ⋆ l ⟧ ⟹ P
```

Now we can apply the induction hypothesis as a case rule. In the case analysis on the successors of $t^* \star l$ (which is the same as $t \star k$) most cases are tedious but straightforward. Interesting cases are some of the reductions of $t^* = t\,\mathsf{to}\,y\,\mathsf{in}\,n$. For example if $n \mapsto n'$ we fix some arbitrary u and show that $u\,\mathsf{to}\,y\,\mathsf{in}\,n \mapsto u\,\mathsf{to}\,y\,\mathsf{in}\,n'$ and hence $u \star ([y]n \gg l) \mapsto u \star ([y]n' \gg l)$. Thus by the definition of stack reduction and the form of k we have some k' such that $k \mapsto k'$, and from this can conclude P by hypothesis. The proof makes extensive use of the strong inversion principles described by Berghofer and Urban (2008). Having established the general claim, we can easily instantiate it to the case where t is neutral, giving the rule *dismantle-cases'* shown in (3). This rule facilitates a formalization of Lindley and Stark's Theorem 5 very close to the original proof (Figure 1).

3.5 Fundamental Theorem

The fundamental theorem of logical relations states that every typeable term of type τ is contained in RED_τ. As usual, it is proved by induction on typing derivations, and a case analysis on the last rule of the derivation. It needs to be generalized to open terms for the induction to go through, by substituting terms reducible at the corresponding type.

```
abbreviation
  θ closes Γ ≡ ∀ x τ. ((x,τ) ∈ set Γ
                  ⟶ (∃ t. θ maps x to t ∧ t ∈ RED τ))

theorem Fundamental-Theorem:
  assumes a: Γ ⊢ t : τ   and   b: θ closes Γ
  shows θ<t> ∈ RED τ
```

```
case 3 from ‹CR3 σ› have ih-CR4-σ : CR4 σ ..
{ fix t assume t′-red: ⋀ t′ . t ↦ t′ ⟹ t′ ∈ RED (T σ)
      and neut-t: NEUT t
 { fix k assume k-red: k ∈ SRED σ
  fix x have NEUT (Var x) unfolding NEUT-def by simp
  hence Var x ∈ RED σ using normal-var ih-CR4-σ
   by (simp add: CR4-def)
  hence SN ([Var x] ⋆ k) using k-red by simp
  hence SSN k by (rule SN-SSN)
  hence SN (t ⋆ k) using k-red
  proof (induct k rule:SSN.induct)
   case (SSN-intro k)
   have ih : ⋀k′. ⟦ k ↦ k′ ; k′ ∈ SRED σ ⟧ ⟹ SN (t ⋆ k′)
    and k-red: k ∈ SRED σ by fact+
   { fix r assume r: t ⋆ k ↦ r
    hence SN r using neut-t
    proof (cases rule:dismantle-cases′)
     case (M t′) hence t-t′: t ↦ t′ and r-def: r = t′ ⋆ k .
     from t-t′ have t′ ∈ RED (T σ) by (rule t′-red)
     thus SN r using k-red r-def by simp
    next
     case (K k′) hence k-k′: k ↦ k′ and r-def: r = t ⋆ k′ .
     { fix s assume s ∈ RED σ
      hence SN ([s] ⋆ k) using k-red
       by simp
      moreover have [s] ⋆ k ↦ [s] ⋆ k′
       using k-k′ unfolding stack-reduction-def ..
      ultimately have SN ([s] ⋆ k′) ..
     } hence k′ ∈ SRED σ by simp
     with k-k′ show SN r unfolding r-def by (rule ih)
    qed } thus SN (t ⋆ k) ..
  qed } hence t ∈ RED (T σ) by simp
} thus CR3 (T σ) unfolding CR3-def CR3-RED-def by blast
```

Figure 1. Formal version of *CR3* for $T\sigma$

In the proof of this theorem, one needs a 'semantic' variant of each typing rule which states how to derive reducibility of a term from reducibility of its subterms. For instance, in the case of lambda abstraction this is asserted by the following lemma:

```
lemma abs-RED:
  assumes ∀ s∈RED τ. t[x::=s]∈RED σ
  shows Lam x t ∈RED (τ→σ)
```

Lindley and Stark (2005) cover only those cases that deal with the new terms and typing rules. As for the proof of CR1-3, this approach is reflected in our formalization, where we can reuse the proof scripts from the simply typed lambda calculus. In fact, only the cases for lambda abstraction and sequencing require an explicit statement, all other cases in the proof of the fundamental theorem are proved automatically by a single application of *auto*.

Thus, the remaining proof obligation is the lifting of reducibility to sequencing, i.e., inferring $s\,\mathsf{to}\,x\,\mathsf{in}\,t \in RED_{T\tau}$ from the corresponding assumptions for terms $\Gamma \vdash s : T\sigma$ and $\Gamma, x{:}\sigma \vdash t : T\tau$. Because of the definition of $RED_{T\tau}$ in terms of stacks and *SN*, the key lemma for this is that if $SN(p)$ and $SN(n[x ::= p] \star k)$, then also $SN(([p]\,\mathsf{to}\,x\,\mathsf{in}\,n) \star k)$. In Isabelle, we formalize this as:

```
lemma to-RED-aux:
  assumes SN p  and  SN (n[x::=p] ⋆ k)  and  x ♯ p  x ♯ k
  shows SN (([p] to x in n) ⋆ k)
```

Lindley and Stark prove this lemma by (natural) induction on $|k| + \max(n \star k) + \max(p)$. Here we deviate from their proof, and instead use our inductive characterization of strong normalization to first establish a more general induction principle:

lemma *triple-induct*:
assumes *a*: $SN\ (p)$
and *b*: $SN\ (q)$
and *hyp*: $\bigwedge\ (p{::}trm)\ (q{::}trm)\ (k{::}stack)\ .$
$[\![\ \bigwedge p'\ .\ p \mapsto p' \Longrightarrow P\ p'\ q\ k\ ;$
$\bigwedge q'\ k\ .\ q \mapsto q' \Longrightarrow P\ p\ q'\ k;$
$\bigwedge k'\ .\ \ |k'| < |k| \Longrightarrow P\ p\ q\ k'\ ]\!] \Longrightarrow P\ p\ q\ k$
shows $P\ p\ q\ k$

Essentially, *triple-induct* is based on the lexicographic ordering on $(SN, \mapsto) \times (SN, \mapsto) \times (\mathbb{N}, >)$, exploiting that $\mapsto$ is a well-founded relation on the set of strongly normalizing terms. However, this induction principle is more specific than the lexicographic induction, for which the first part of '*hyp*' would be replaced by

$$\bigwedge p'\ q\ k.\ [\![\ SN\ (q);\ p \mapsto p'\ ]\!] \Longrightarrow P\ p'\ q\ k\ .$$

Since we do not need this generality (which would in fact lead to an induction hypothesis that is more difficult to apply), we derive *triple-induct* directly in the above form.

One interesting consequence of using *triple-induct*, rather than natural induction on $|k| + \max(n \star k) + \max(p)$, is that we do not have to establish that $k \mapsto k' \Rightarrow |k'| \leq |k|$. This is otherwise required in the proof of *to-RED-aux* to handle the case of a reduction occurring somewhere inside k. Also, note that the *triple-induct* rule abstracts from the fact that one of the strongly normalizing terms on which we induct is given in the form of dismantling the stack k. For the instantiation we therefore prove that for any term q with $SN(q)$ we have:

$$\begin{aligned}&\bigwedge m\ .\ [\![\ q = m \star k\ ;\ SN(m[x{::=}p] \star k)\ ]\!]\\ &\quad\Longrightarrow SN\ (([p]\ to\ x\ in\ m) \star k)\end{aligned} \qquad (4)$$

After instantiation with p, q, and k, *triple-induct* provides us with the following induction hypotheses:

have *ih-p*:
$\bigwedge p'\ m\ .\ [\![p \mapsto p';\ q = m \star k;\ SN\ (m[x{::=}p'] \star k);\ x \sharp p';\ x \sharp k]\!]$
$\Longrightarrow SN\ (([p']\ to\ x\ in\ m) \star k)$ **by** *fact*
have *ih-q*:
$\bigwedge q'\ m\ k.[\![q \mapsto q';\ q' = m \star k;\ SN\ (m[x{::=}p] \star k);\ x \sharp p;\ x \sharp k]\!]$
$\Longrightarrow SN\ (([p]\ to\ x\ in\ m) \star k)$ **by** *fact*
have *ih-k*:
$\bigwedge k'\ m\ .\ [\![\ |k'| < |k|;\ q = m \star k';\ SN\ (m[x{::=}p] \star k');$
$x \sharp p;\ x \sharp k']\!] \Longrightarrow SN\ (([p]\ to\ x\ in\ m) \star k')$ **by** *fact*

Once again we follow the reasoning of Lindley and Stark, this time by showing that all successors of $([p]\,\mathsf{to}\,x\,\mathsf{in}\,m) \star k$ are strongly normalizing. Hence, we fix some r, assume

$$([p]\,\mathsf{to}\,x\,\mathsf{in}\,m) \star k \mapsto r\ ,$$

and reason by inversion. Unfortunately, the case rule *dismantle-cases'* defined in Section 3.4 does not allow us to choose names for the binders in the cases where there is an interaction between $[p]\,\mathsf{to}\,x\,\mathsf{in}\,m$ and the first stack frame of k. As a consequence, it gives rise to equations of the form

$$[p]\,\mathsf{to}\,x\,\mathsf{in}\,n = u\,\mathsf{to}\,x'\,\mathsf{in}\,v$$

for arbitrarily chosen, fresh x'. Since we have ensured that x itself is suitably fresh, we would prefer to obtain equations of the form

$$[p]\,\mathsf{to}\,x\,\mathsf{in}\,m = u\,\mathsf{to}\,x\,\mathsf{in}\,v$$

when doing inversion on $[p]\,\mathsf{to}\,x\,\mathsf{in}\,m \star k \mapsto r$, as this immediately yields $s = [p]$ and $v = m$ without further alpha-renaming. This is achieved by strengthening *dismantle-cases* to the following form:

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: $\bigwedge t'\ .\ [\![\ t \mapsto t'\ ;\ r = t' \star k\ ]\!] \Longrightarrow P$
and K: $\bigwedge k'\ .\ [\![\ k \mapsto k'\ ;\ r = t \star k'\ ]\!] \Longrightarrow P$
and B: $\bigwedge s\ n\ l\ .\ [\![\ t = [s]\ ;\ k = [y]n{\gg}l\ ;\ r = (n[y{::=}s]) \star l\ ]\!]$
$\Longrightarrow P$
and A: $\bigwedge u\ v\ n\ l\ .\ [\![\ x \sharp (z,n)\ ;\ t = u\ to\ x\ in\ v\ ;\ k = [z]n{\gg}l\ ;$
$r = (u\ to\ x\ in\ (v\ to\ z\ in\ n)) \star l\ ]\!] \Longrightarrow P$
shows P

The change from the weak *dismantle-cases* rule to *dismantle-strong-cases* is that the variables y, x and z (cases B and A) are no longer bound, but become free variables of the theorem. When deriving the strong rule from the weak one, we need to show each case for universally quantified names but the respective hypothesis of the strong case rule only provides the claim for specific (externally chosen) names. In the case of a β-reduction, we get $t = [s]$, $k = [y']n{\gg}l$, and $r = (n[y'{::=}s]) \star l$ for some fixed name y'. Either $y = y'$ and we can apply B directly or $y \sharp n$ (using f) and we can α-rename and apply B as well. The case for A follows the same pattern but is more tedious due to the nested binding structure. The rule *dismantle-strong-cases* can also be seen as a strong inversion principle for the three place relation $t \star k \mapsto r$, where the freshness conditions imposed by f correspond to those described by Berghofer and Urban (2008) for inductively defined relations.

Using this strong case rule we obtain a a proof of *to-RED-aux* staying relatively close to the informal reasoning of Lindley and Stark. After applying the *dismantle-strong-cases* rule, we reason as follows: The case of toplevel reductions occurring in $[p]\,\mathsf{to}\,x\,\mathsf{in}\,m$ can be handled by hypothesis. The case of the commuting conversion interacting with the top frame reduces the length of k but does not change q or p and so this is handled by *ih-k*. These are the cases spelled out by Lindley and Stark (2005).

The second interaction from our case rule cannot occur, which leaves us with reductions occurring within k, m, or p. The first two of these are easily handled by the induction hypothesis *ih-q*. But the case of a reduction in p requires a special treatment. To apply the induction hypothesis, we have to derive $SN(m[x ::= p'] \star k)$ from $SN(m[x ::= p] \star k)$. For this, we actually need to reason about the reflexive, transitive closure of the reduction relation, obtained from Isabelle's built-in closure operator, and written $\mapsto^*$ as usual. We can easily establish that strong normalization does not only extend to immediate successors, but also to the transitive case.

lemma *SN-trans* : $[\![\ p \mapsto^* p'\ ;\ SN\,p\ ]\!] \Longrightarrow SN\,p'$

So the last remaining proof obligation is to show:

lemma *red-subst*: $p \mapsto p' \Longrightarrow (m[x{::=}p]) \mapsto^* (m[x{::=}p'])$

The proof is a straightforward induction on the term p but requires transitive versions of *all* context rules of the reduction relation. To complete the proof of *to-RED-aux* we instantiate q in (4) with $n \star k$ and therefore m with n.

This establishes the *Fundamental-Theorem* from which we conclude that all well-typed terms are strongly normalizing using the identity substitution and *CR1*.

4. Discussion

Our whole development is performed using the Isar structured proof language (Wenzel 2002). This allows us to naturally follow the forward reasoning style of Lindley and Stark (2005), even though the natural deduction reasoning in Isabelle/HOL is centered around backward proofs. The use of Isar has further benefits: for instance, we could port the complete development from Isabelle2008 to Isabelle2009, without touching any of the proofs.

We believe that we achieve the goal of 'faithfully' formalizing Lindley and Stark's paper-and-pencil proof. Clearly we need to reason in much smaller steps, and formulate additional lemmas, most notably those for case analyses (like *dismantle-cases* and *dismantle-strong-cases*) and tailor-made induction principles (like *triple-induct*). Once this has been done, however, the statement of lemmas and proofs like the one shown in Figure 1 become pretty direct translations of their informal counterparts. As mentioned earlier, we were also able to directly reuse large portions of a previously formalized normalization proof for the simply typed lambda calculus (Nominal Methods group 2009). This provided a pleasant analogy in our development to the approach of Lindley and Stark, who show only the cases specific to Moggi's calculus.

Since our formalization stays reasonably close to the informal reasoning, we can confirm that Lindley and Stark's proofs are correct, apart from one small inaccuracy: With regard to stack reductions (*cf.* definition in Section 3.3), Lindley and Stark claim that $k \mapsto k'$ if and only if $x \star k \mapsto x \star k'$ for any variable x, without any further restrictions on x. However, the proposed proof of this claim – using the preservation of reduction under substitution – only works if $x \sharp (k, k')$. The proof becomes significantly more tedious if one drops the freshness requirement $x \sharp k'$, and we have no formal proof for an arbitrary variable x. (But we also do not have a counterexample.) On the other hand, this claim about stack reduction is not essential to the overall proof: Lindley and Stark use it only to establish the fact that stack reduction does not increase the stack length, i.e. that $k \mapsto k'$ implies $|k'| \leq |k|$, for which the trivial direction from left to right suffices. Our formalization does not even use this property of stack reduction, since we replaced Lindley and Stark's induction involving $|k|$ as summand by the stronger induction principle *triple-induct*, shown in Section 3.5. Of course, this would also apply to the informal proof.

The strong induction and inversion principles are very convenient in proofs: a finitely supported 'freshness context' can be chosen, and the variables that appear in induction hypotheses will be fresh for this context. Since these principles are unsound for arbitrary inductively defined relations (Berghofer and Urban 2008, Section 2), their automatic derivation requires the relation to satisfy the *vc-condition*. This condition demands that all predicates appearing in the definition must be equivariant, and that the introduction rules of the relation must conform to a second, syntactically defined restriction: basically, this restriction amounts to the fact that a variable occurring in a binding position somewhere in a rule cannot also appear free in the conclusion of the rule.

While equivariance of all our relations is easy to show, the syntactic restriction forces us to add freshness conditions, for instance to the introduction rules of the reduction relation in Section 3.2, that are not needed in informal proofs. Proofs involving these additional freshness conditions quickly become unwieldy; one example are the context rules for $\mapsto^*$ used in *red-subst*, which can be proved automatically for the standard formulation of $\mapsto$ but become tedious with the additional freshness conditions. Therefore we follow the approach suggested by Berghofer and Urban (2008) and show the equivalence of the introduction rules adhering to the vc-condition and the standard ones, which can then be used instead. In Berghofer and Urban's example of the reduction relation of simply typed lambda calculus, β-reduction is the only problematic rule. In contrast, in our case each rule with a sequencing construct $s \,\mathsf{to}\, x \,\mathsf{in}\, t$ in its conclusion requires the condition $x \sharp s$. All the adequacy proofs, showing that these freshness conditions can be removed from each of our reduction rules, follow a 'standard' pattern (with the exception of associativity which involves *two* binders, and rearranges them), so some automatic support for these proofs could be feasible.

Related to the vc-conditions, note that there is a subtlety in the proof sketch for the case analysis on $t \star k \mapsto r$ from Section 3.4. There, we fixed some arbitrary term u and showed that $s \mapsto s'$ implies $u \,\mathsf{to}\, y \,\mathsf{in}\, s \mapsto u \,\mathsf{to}\, y \,\mathsf{in}\, s'$. Due to the missing freshness condition $y \sharp u$ (since u must be chosen *arbitrarily*, and only *after y* has been chosen), this reduction step does not directly follow with the introduction rules of the vc-compatible version (1) of the reduction relation. This is similar to the limitation of Nominal Isabelle's strong induction principles noted by Aydemir et al. (2008, Section 4.6), where the freshness context must be instantiated when induction is invoked, and therefore cannot ensure additional freshness requirements possibly needed when the hypotheses are subsequently applied. However, since our development includes a proof of adequacy of the vc-compatible reduction relation with respect to the standard one, the freshness condition $y \sharp u$ is not needed. Alternatively, if one does not want to prove adequacy (which involves a fair amount of α-renaming), one could replace the term u with some suitably fresh variable x and use the fact that $x \sharp (k, k')$ and $x \star k \mapsto x \star k'$ also implies $k \mapsto k'$. In this particular case, the freshness requirements for k and k' can be easily satisfied.

In summary, our formalization benefits from the automatic derivation of induction principles for nominal datatypes and the good support provided by the nominal package for the definition of primitive recursive functions over nominal datatypes, such as the substitution operation. Unfortunately, there is no similar machinery available for non-primitive recursive definitions, like the stack dismantling operation $t \star k$: we have to prove its basic properties manually. At the moment we do not know of other such examples, however, so it is unclear whether it is worth to try and extend the theory in this direction. The strong induction and case analysis principles (Berghofer and Urban 2008) were extremely helpful to avoid several of the proofs being obscured with inessential reasoning about alpha-equivalence. Other case studies (Bengtson and Parrow 2007) make similar observations about the indispensability of strong induction and case rules to keep proofs manageable. To take full advantage of the automatic derivation of strong induction and case rules we have to add several additional freshness requirements, in particular to the reduction relation. These are needed to satisfy the (technical) vc-condition of Berghofer and Urban (2008), which is too restrictive for calculi with let-like binding structures. Essentially, one has to add an extra freshness condition whenever a term appears outside of the scope of a binder.

Schürmann and Sarnat (2008) present an approach to logical relation proofs in Twelf, which have proved challenging before. Their key idea is to additionally represent an 'assertion logic' in Twelf, and then formalize reasoning about the logical relation in this assertion logic rather than the meta-logic of Twelf. Schürmann and Sarnat illustrate their technique with several examples, including a normalization proof along the lines of Lindley and Stark (2005). A technical difference is that they prove weak normalization (more precisely, the existence of β-short η-long normal forms), and their formalization has a rather different feel due to the use of the assertion logic and a representation with higher-order abstract syntax. A more detailed comparison between the formalizations could be interesting.

Previously, the first author has used Lindley and Stark's proof technique to show strong normalization for the call-by-push-value calculus of Levy (2004). This calculus has a richer set of types, for instance including product and sum types of arbitrary (even infinite) arity. At present, it is unclear to us how to best represent such syntactic features, and thereby also adapt the formalized proof to Levy's calculus.

Acknowledgments

We thank the Nominal Methods group and the members of the Isabelle mailing list for their helpful answers to our questions.

References

Andreas Abel. Normalization for the Simply Typed λ-calculus in Twelf. In *Informal Proceedings LFM'04*, 2004.

Thorsten Altenkirch. A formalization of the strong normalization proof for System F in LEGO. In *TLCA*, volume 664 of *LNCS*, pages 13–28. Springer, 1993.

Brian E. Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. Engineering formal metatheory. In *POPL*, pages 3–15. ACM Press, 2008.

Jesper Bengtson and Joachim Parrow. Formalising the π-Calculus Using Nominal Logic. In *FOSSACS*, volume 4423 of *LNCS*, pages 63–77. Springer, 2007.

Ulrich Berger, Stefan Berghofer, Pierre Letouzey, and Helmut Schwichtenberg. Program extraction from normalization proofs. *Studia Logica*, 82 (1):25–49, 2006.

Stefan Berghofer and Christian Urban. Nominal inversion principles. In *TPHOLs*, volume 5170 of *LNCS*, pages 71–85. Springer, 2008.

Kevin Donnelly and Hongwei Xi. A formalization of strong normalization for simply-typed lambda-calculus and System F. *Electr. Notes Theor. Comput. Sci.*, 174(5):109–125, 2007.

Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

Florian Haftmann. *Haskell-style type classes with Isabelle/Isar*, 2009. URL `http://isabelle.in.tum.de/doc/classes.pdf`.

Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*. Springer, 2004.

Sam Lindley and Ian Stark. Reducibility and $\top\top$-lifting for computation types. In *TLCA*, volume 3461 of *LNCS*, pages 262–277. Springer, 2005.

Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1): 55–92, 1991.

Julien Narboux and Christian Urban. Formalising in Nominal Isabelle Crary's completeness proof for equivalence checking. *Electr. Notes Theor. Comput. Sci.*, 196:3–18, 2008.

Nominal Methods group. Strong normalization for the simply typed lambda calculus, 2009. URL `http://isabelle.in.tum.de/dist/library/HOL/HOL-Nominal/Examples/SN.html`.

Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Inf. Comput.*, 186(2):165–193, 2003.

Andrew M. Pitts. Alpha-structural recursion and induction. *J. ACM*, 53(3): 459–506, 2006.

Dag Prawitz. Ideas and results in proof theory. In *Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 235–307. North-Holland, 1971.

Carsten Schürmann and Jeffrey Sarnat. Structural logical relations. In *LICS*, pages 69–80. IEEE Computer Society, 2008.

Christian Urban. Nominal techniques in Isabelle/HOL. *J. Autom. Reason.*, 40(4):327–356, 2008.

Christian Urban, Stefan Berghofer, and Michael Norrish. Barendregt's variable convention in rule inductions. In *CADE-21*, volume 4603 of *LNCS*, pages 35–50. Springer, 2007.

Markus M. Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents*. PhD thesis, Technische Universität München, 2002.