Barendregt's Variable Convention in Rule Inductions

Christian Urban[1], Stefan Berghofer[1], and Michael Norrish[2]

[1] TU Munich, Germany
[2] NICTA, Australia

Abstract. Inductive definitions and rule inductions are two fundamental reasoning tools in logic and computer science. When inductive definitions involve binders, then Barendregt's variable convention is nearly always employed (explicitly or implicitly) in order to obtain simple proofs. Using this convention, one does not consider truly arbitrary bound names, as required by the rule induction principle, but rather bound names about which various freshness assumptions are made. Unfortunately, neither Barendregt nor others give a formal justification for the variable convention, which makes it hard to formalise such proofs. In this paper we identify conditions an inductive definition has to satisfy so that a form of the variable convention can be built into the rule induction principle. In practice this means we come quite close to the informal reasoning of "pencil-and-paper" proofs, while remaining completely formal. Our conditions also reveal circumstances in which Barendregt's variable convention is not applicable, and can even lead to faulty reasoning.

1 Introduction

In informal proofs about languages that feature bound variables, one often assumes (explicitly or implicitly) a rather convenient convention about those bound variables. Barendregt's statement of the convention is:

> **Variable Convention**: *If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables.* [2, Page 26]

The reason for this convention is that it leads to very slick informal proofs—one can avoid having to rename bound variables.

One example of such a slick informal proof is given in [2, Page 60], proving the substitutivity property of the $\underset{1}{\longrightarrow\!\!\!\!\!\rightarrow}$ (or "parallel reduction") relation, which is defined by the rules:

$$\frac{}{M \underset{1}{\twoheadrightarrow} M}\;\mathrm{One}_1 \qquad \frac{M \underset{1}{\twoheadrightarrow} M'}{lam(y.M) \underset{1}{\twoheadrightarrow} lam(y.M')}\;\mathrm{One}_2$$
$$\frac{M \underset{1}{\twoheadrightarrow} M' \quad N \underset{1}{\twoheadrightarrow} N'}{app(M,N) \underset{1}{\twoheadrightarrow} app(M',N')}\;\mathrm{One}_3 \qquad \frac{M \underset{1}{\twoheadrightarrow} M' \quad N \underset{1}{\twoheadrightarrow} N'}{app(lam(y.M),N) \underset{1}{\twoheadrightarrow} M'[y := N']}\;\mathrm{One}_4 \qquad (1)$$

The substitutivity property states:

Lemma. If $M \underset{1}{\twoheadrightarrow} M'$ and $N \underset{1}{\twoheadrightarrow} N'$, then $M[x := N] \underset{1}{\twoheadrightarrow} M'[x := N']$.

In [2], the proof of this lemma proceeds by an induction over the definition of $M \underset{1}{\twoheadrightarrow} M'$. Though Barendregt does not acknowledge the fact explicitly, there are two places in his proof where the variable convention is used. In case of rule One_2, for example, Barendregt writes (slightly changed to conform with the syntax we shall employ for λ-terms):

Case One_2. $M \underset{1}{\twoheadrightarrow} M'$ is $lam(y.P) \underset{1}{\twoheadrightarrow} lam(y.P')$ and is a direct consequence of $P \underset{1}{\twoheadrightarrow} P'$. By induction hypothesis one has $P[x := N] \underset{1}{\twoheadrightarrow} P[x := N']$. But then $lam(y.P[x := N]) \underset{1}{\twoheadrightarrow} lam(y.P'[x := N'])$, i.e. $M[x := N] \underset{1}{\twoheadrightarrow} M'[x := N']$. □

However, the last step in this case only works if one knows that

$$\begin{array}{c} lam(y.P[x := N]) = lam(y.P)[x := N] \quad \text{and} \\ lam(y.P'[x := N']) = lam(y.P')[x := N'] \end{array}$$

which only holds when the bound variable y is not equal to x, and not free in N and N'. These assumptions might be inferred from the variable convention, provided one has a formal justification for this convention. Since one usually assumes that λ-terms are α-equated, one might think a simple justification for the variable convention is along the lines that one can always rename binders with fresh names. This is however *not* sufficient in the context of inductive definitions, because there rules can have the same variable occurring both in binding and non-binding positions. In rule One_4, for example, y occurs in binding position in the subterm $lam(y.M)$, and in the subterm $M'[y := N']$ it is in a *non*-binding position. Both occurrences must refer to the same variable as the rule

$$\frac{M \underset{1}{\twoheadrightarrow} M' \quad N \underset{1}{\twoheadrightarrow} N'}{app(lam(z.M), N) \underset{1}{\twoheadrightarrow} M'[y := N']}\ \mathrm{One}_4'$$

leads to a nonsensical reduction relation.

In the absence, however, of a formal justification for the variable convention, Barendregt's argument considering only a well-chosen y seems dubious, because the induction principle that comes with the inductive definition of $\underset{1}{\twoheadrightarrow}$ is:

$$\frac{\begin{array}{l} \forall M.\ P\,M\,M \\ \forall y\,M\,M'.\ P\,M\,M' \Rightarrow P\,(lam(y.M))\,(lam(y.M')) \\ \forall M\,M'\,N\,N'.\ P\,M\,M' \wedge P\,N\,N' \Rightarrow P\,(app(M,N))\,(app(M',N')) \\ \forall y\,M\,M'\,N\,N'.\ P\,M\,M' \wedge P\,N\,N' \Rightarrow P\,(app(lam(y.M),N))\,(M'[y := N']) \end{array}}{M \underset{1}{\twoheadrightarrow} N \Rightarrow P\,M\,N}$$

where both cases One_2 and One_4 require that the corresponding implication holds for **all** y, not just the ones with $y \neq x$ and $y \notin FV(N, N')$. Nevertheless, we will show that Barendregt's apparently dubious step can be given a faithful, and sound, mechanisation. Being able to restrict the argument in general to a suitably chosen bound variable will, however, depend on the form of the rules in an inductive definition. In this paper we will make precise what this form is and will show how the variable convention can be built into the induction principle.

The interactions between bound and free occurrences of variables, and their consequences for obtaining a formal argument, seem to often be overlooked in the literature when claiming that proofs by rule inductions are straightforward. One example of this comes with a weakening result for contexts in the simply-typed λ-calculus.

We assume types are of the form $T ::= X \mid T \to T$, and that typing contexts (finite lists of variable-type pairs) are *valid* if no variable occurs twice. The typing relation can then be defined by the rules

$$\frac{\mathit{valid}(\Gamma) \quad (x\!:\!T) \in \Gamma}{\Gamma \vdash \mathit{var}(x) : T}\,\mathrm{Type}_1 \qquad \frac{\Gamma \vdash M : T_1 \to T_2 \quad \Gamma \vdash N : T_1}{\Gamma \vdash \mathit{app}(M,N) : T_2}\,\mathrm{Type}_2$$
$$\frac{x \# \Gamma \quad (x\!:\!T_1)::\Gamma \vdash M : T_2}{\Gamma \vdash \mathit{lam}(x.M) : T_1 \to T_2}\,\mathrm{Type}_3 \tag{2}$$

where $(x : T) \in \Gamma$ stands for list-membership, and $x \# \Gamma$ for x being fresh for Γ, or equivalently x not occuring in Γ. Define a context Γ' to be weaker than Γ (written $\Gamma \subseteq \Gamma'$), if every name-type pair in Γ also appears in Γ'. Then we have

Lemma (Weakening). *If $\Gamma \vdash M : T$ is derivable, and $\Gamma \subseteq \Gamma'$ with Γ' valid, then $\Gamma' \vdash M : T$ is also derivable.*

The informal proof of this lemma is straightforward, provided(!) one uses the variable convention.

Informal Proof. *By rule induction over $\Gamma \vdash M : T$ showing that $\Gamma' \vdash M : T$ holds for all Γ' with $\Gamma \subseteq \Gamma'$ and Γ' being valid.*

Case Type_1: *$\Gamma \vdash M : T$ is $\Gamma \vdash \mathit{var}(x) : T$. By assumption we know $\mathit{valid}(\Gamma')$, $(x\!:\!T) \in \Gamma$ and $\Gamma \subseteq \Gamma'$. Therefore we can use* Type_1 *to derive $\Gamma' \vdash \mathit{var}(x) : T$.*

Case Type_2: *$\Gamma \vdash M : T$ is $\Gamma \vdash \mathit{app}(M_1, M_2) : T$. Case follows from the induction hypotheses and rule* Type_2.

Case Type_3: *$\Gamma \vdash M : T$ is $\Gamma \vdash \mathit{lam}(x.M_1) : T_1 \to T_2$. Using the variable convention we assume that $x \# \Gamma'$. Then we know that $((x\!:\!T_1)::\Gamma')$ is valid and hence that $((x\!:\!T_1)::\Gamma') \vdash M_1 : T_2$ holds. By appealing to the variable convention again, we have that $\Gamma' \vdash \mathit{lam}(x.M_1) : T_1 \to T_2$ holds using rule* Type_3 □

However, in order to make this informal proof work with the induction principle that comes with the rules in (2), namely

$$\frac{\begin{array}{l}\forall \Gamma\, x\, T.\ \mathit{valid}(\Gamma) \wedge (x\!:\!T) \in \Gamma \Rightarrow P\ \Gamma\ (\mathit{var}(x))\ T \\ \forall \Gamma\, M\, N\, T_1\, T_2.\ P\ \Gamma\ M\ (T_1 \to T_2)\ \wedge\ P\ \Gamma\ N\ T_1\ \Rightarrow P\ \Gamma\ (\mathit{app}(M,N))\ T_2 \\ \forall x\, \Gamma\, M\, T_1\, T_2.\ x \# \Gamma \wedge P\ ((x\!:\!T_1)::\Gamma)\ M\ T_2\ \Rightarrow P\ \Gamma\ (\mathit{lam}(x.M))\ (T_1 \to T_2)\end{array}}{\Gamma \vdash M : T\ \Rightarrow\ P\ \Gamma\ M\ T} \tag{3}$$

we need in case of rule Type_3 to be able to rename the bound variable to be suitably fresh for Γ'; by the induction we only know that x is fresh for the smaller context Γ. To be able to do this renaming depends on two conditions: first, there must exist a fresh variable which we can choose. In our example this means that the context Γ' must not contain all possible free variables. Second, the relation $\Gamma \vdash M : T$ must be invariant under suitable renamings. This is because when we change the goal from $\Gamma' \vdash lam(x.M_1) : T_1 \to T_2$ to $\Gamma' \vdash lam(z.M_1[x := z]) : T_1 \to T_2$, we must be able to infer from $((x : T_1) :: \Gamma') \vdash M_1 : T_2$ that $((z : T_1) :: \Gamma') \vdash M_1[x := z] : T_2$ holds. This invariance under renamings does, however, *not* hold in general, not even under renamings with fresh variables. For example if we assume that variables are linearly ordered, then the relation

$$\frac{v = min\{v_0, \ldots, v_n\}}{(\{v_0, \ldots, v_n\}, v)}$$

that associates finite subsets of these variables to the smallest variable occurring in it, is *not* invariant (apply the renaming $[v := v']$ where v' is a variable that is bigger than every variable in $\{v_0, \ldots, v_n\}$). Other examples are rules that involve a substitution for concrete variables or a substitution with concrete terms. In order to avoid such pathological cases, we require that the relation for which one wants to employ the variable convention must be invariant under renamings; from the induction we require that the variable convention can only be applied in contexts where there are only finitely many free names.

However, these two requirements are *not* yet sufficient, and we need to impose a second condition that inductive definitions have to satisfy. Consider the function that takes a list of variables and binds them in λ-abstractions, that is

$$bind\ t\ [\,] \stackrel{\text{def}}{=} t \qquad bind\ t\ (x :: xs) \stackrel{\text{def}}{=} lam(x.(bind\ t\ xs))$$

Further consider the relation $\hookrightarrow$, which "unbinds" the outermost abstractions of a λ-term and is defined by:

$$\frac{}{var(x) \hookrightarrow [\,], var(x)}\ \text{Unbind}_1 \qquad \frac{}{app(t_1, t_2) \hookrightarrow [\,], app(t_1, t_2)}\ \text{Unbind}_2$$
$$\frac{t \hookrightarrow xs, t'}{lam(x.t) \hookrightarrow x :: xs, t'}\ \text{Unbind}_3 \tag{4}$$

Of course, this relation cannot be expressed as a function because the bound variables do not have "particular" names. Nonetheless it is well-defined, and not trivial. For example, we have

$$\begin{array}{l} lam(x.lam(y.app(var(x), app(var(y), var(z))))) \\ \qquad\qquad\qquad \hookrightarrow [x, y], app(var(x), app(var(y), var(z)))\ \ \text{and} \\ lam(x.lam(y.app(var(x), app(var(y), var(z))))) \\ \qquad\qquad\qquad \hookrightarrow [y, x], app(var(y), app(var(x), var(z))) \end{array}$$

but we also have $\forall t'.\ lam(x.lam(y.app(x, app(var(y), var(z))))) \not\hookrightarrow [x, z], t'$.

Further, one can also easily establish (by induction on the term t) that for every t there exists a t' and a list xs of distinct variables such that $t \hookrightarrow xs, t'$ holds, demonstrating that the relation is "total" if the last two parameters are viewed as results.

If one wished to do rule inductions over the definition of this relation, one might imagine that the variable convention allowed us to assume that the bound name x was distinct from the free variables of the conclusion of the rule, and in particular that x could not appear in the list xs. However, this use of the variable convention quickly leads to the *faulty* lemma:

Lemma (Faulty). *If* $t \hookrightarrow (x :: xs), t'$ *and* $x \in FV(t')$ *then* $x \in FV(bind\ t'\ xs)$.

The "proof" is by an induction over the rules given in (4) and assumes that the binder x in the third rule is fresh with respect to xs. This lemma is of course false as witnessed by the term $lam(x.lam(x.var(x)))$. Therefore, including the variable convention in the induction principle that comes with the rules in (4), would produce an inconsistency. To prevent this problem we introduce a second condition for rules, which requires that all variables occurring as a binder in a rule must be fresh (a notion which we shall make precise later on) for the conclusion of this rule, and if a rule has several such variables, they must be mutually distinct.

Our Contribution: We introduce two conditions inductive definitions must satisfy in order to make sure they are compatible with the variable convention. We will build a version of this convention into the induction principles that come with the inductive definitions. Moreover, it will be shown how these new ("vc-compatible") induction principles can be automatically derived in the nominal datatype package [11, 9]. The presented results have already been extensively used in formalisations: for example in our formalisations of the CR and SN properties in the λ-calculus, in a formalisation by Bengtson and Parrow for several proofs in the pi-calculus [3], in a formalisation of Crary's chapter on logical relation [4], and in various formalised proofs on structural operational semantics.

2 Nominal Logic

Before proceeding, we briefly introduce some important notions from nominal logic [8, 11]. In particular, we will build on the three central notions of *permutations*, *support* and *equivariance*. Permutations are finite bijective mappings from atoms to atoms, where atoms are drawn from a countably infinite set denoted by $\mathbb{A}$. We represent permutations as finite lists whose elements are swappings (i.e., pairs of atoms). We write such permutations as $(a_1\, b_1)(a_2\, b_2) \cdots (a_n\, b_n)$; the empty list $[\,]$ stands for the identity permutation. A permutation π *acting* on an atom a is defined as:

$$[\,]\bullet a \stackrel{\text{def}}{=} a \qquad ((a_1\, a_2) :: \pi)\bullet a \stackrel{\text{def}}{=} \begin{cases} a_2 & \text{if } \pi\bullet a = a_1 \\ a_1 & \text{if } \pi\bullet a = a_2 \\ \pi\bullet a & \text{otherwise} \end{cases}$$

where $(a\, b) :: \pi$ is the composition of a permutation followed by the swapping $(a\, b)$. The composition of π followed by another permutation π' is given by list-concatenation, written as $\pi' @ \pi$, and the inverse of a permutation is given by list reversal, written as π^{-1}. Our representation of permutations as lists does not give unique representatives: for example, the permutation $(a\, a)$ is "equal" to the identity permutation. We equate permutations with a relation $\sim$:

Definition 1 (Permutation Equality). *Two permutations are* equal*, written* $\pi_1 \sim \pi_2$*, provided* $\pi_1 \bullet a = \pi_2 \bullet a$*, for all* $a \in \mathbb{A}$*.*

The permutation action on atoms can be lifted to other types.

Definition 2 (The Action of a Permutation). *A permutation action* $\pi \bullet (-)$ *lifts to a type* T *provided it the following three properties hold on all values* $x \in T$

$$\begin{array}{rl} (i) & [\,]\bullet x = x \\ (ii) & (\pi_1 @ \pi_2)\bullet x = \pi_1 \bullet (\pi_2 \bullet x) \\ (iii) & \textit{if } \pi_1 \sim \pi_2 \textit{ then } \pi_1 \bullet x = \pi_2 \bullet x \end{array}$$

For example, lists and tuples can be given the following permutation action:

$$\begin{array}{lrcl} \text{lists:} & \pi\bullet[\,] & \stackrel{\text{def}}{=} & [\,] \\ & \pi\bullet(h :: t) & \stackrel{\text{def}}{=} & (\pi\bullet h) :: (\pi\bullet t) \\ \text{tuples:} & \pi\bullet(x_1,\ldots,x_n) & \stackrel{\text{def}}{=} & (\pi\bullet x_1,\ldots,\pi\bullet x_n) \end{array} \qquad (5)$$

Further, on α-equated λ-terms we can define the permutation action:

$$\begin{array}{rcl} \pi\bullet var(x) & \stackrel{\text{def}}{=} & var(\pi\bullet x) \\ \pi\bullet app(M_1, M_2) & \stackrel{\text{def}}{=} & app(\pi\bullet M_1, \pi\bullet M_2) \\ \pi\bullet lam(x.M) & \stackrel{\text{def}}{=} & lam(\pi\bullet x.\pi\bullet M) \end{array} \qquad (6)$$

The second notion that we use is that of *support* (roughly speaking, the support of an element is its set of free atoms). The set supporting an element is defined in terms of permutation actions on that element, so that as soon as one has defined a permutation action for a type, one automatically derives its accompanying notion of support, which in turn determines the notion of freshness (see [11]):

Definition 3 (Support and Freshness). *The* support *of* x *is defined as:* $\texttt{supp}(x) \stackrel{def}{=} \{a \mid \texttt{infinite}\{b \mid (a\,b)\bullet x \neq x\}\}$*. An atom* a *is said to be* fresh *for an* x*, written* $a \# x$*, if* $a \notin \texttt{supp}(x)$*.*

We will also use the auxiliary notation $a \# xs$, in which xs stands for a collection of objects $x_1 \ldots x_n$, to mean $a \# x_1 \ldots a \# x_n$. We further generalise this notation to a collection of atoms, namely $as \# xs$, which means $a_1 \# xs \ldots a_m \# xs$.

Later on we will often make use of the following two properties of freshness, which can be derived from the definition of support, the permutation action on $\mathbb{A}$ and the requirements of permutation actions on other types (see [11]).

Lemma 1.
- (a) $a \# x$ *implies* $\pi\bullet a \# \pi\bullet x$*; and*
- (b) *if* $a \# x$ *and* $b \# x$*, then* $(a\,b)\bullet x = x$*.*

Henceforth we will only be interested in those objects which have finite support, because for them there exists always a fresh atom (recall that the set of atoms $\mathbb{A}$ is infinite).

Lemma 2. *If x is finitely supported, then there exists an atom a such that $a \# x$.*

Unwinding the definitions of permutation actions and support one can often easily calculate the support of an object:

atoms:	$\mathtt{supp}(a) = \{a\}$
tuples:	$\mathtt{supp}(x_1, \ldots, x_n) = \mathtt{supp}(x_1) \cup \ldots \cup \mathtt{supp}(x_n)$
lists:	$\mathtt{supp}([\,]) = \varnothing$, $\mathtt{supp}(h :: t) = \mathtt{supp}(h) \cup \mathtt{supp}(t)$
α-equated λ-terms:	$\mathtt{supp}(\mathit{var}(x)) = \{x\}$ $\mathtt{supp}(\mathit{app}(M, N)) = \mathtt{supp}(M) \cup \mathtt{supp}(N)$ $\mathtt{supp}(\mathit{lam}(x.M)) = \mathtt{supp}(M) - \{x\}$

We therefore note the following: all elements in $\mathbb{A}$ and all α-equated λ-terms are finitely supported. Lists (similarly tuples) containing finitely supported elements are finitely supported. The last three equations show that the support of α-equated λ-terms coincides with the usual notion of free variables. Hence, $a \# M$ with M being an α-equated λ-term coincides with a not being free in M. If b is an atom, then $a \# b$ coincides with $a \neq b$.

The last notion of nominal logic we use here is that of *equivariance*.

Definition 4 (Equivariance).
- *A relation R is* equivariant *if $R\,(\pi \bullet xs)$ is implied by $R\,xs$ for all π.*
- *A function f is* equivariant *provided $\pi \bullet (f\ xs) = f\,(\pi \bullet xs)$ for all π.*

Remark 1. Note that if we regard the term-constructors *var*, *app* and *lam* as functions, then they are equivariant on account of the definition given in (6). Because of the definition in (5), the cons-constructors of lists are equivariant. By a simple structural induction on the list argument of *valid*, we can establish that the relation *valid* is equivariant. By Lem. 1(a) freshness is equivariant. Also list-membership, $(-) \in (-)$, is equivariant, which can be shown by an induction on the length of lists.

3 Schematic Terms and Schematic Rules

Inductive relations are defined as the smallest relation closed under some schematic rules. In this section we will formally specify the form of such rules. Diagrammatically they have the form

$$\frac{\mathit{premises} \quad \mathit{side\text{-}conditions}}{\mathit{conclusion}}\ \varrho \tag{7}$$

where the premises, side-conditions and conclusions are predicates of the form $R\ ts$ where we use the letters R, S, P and Q to stand for predicates; ts stands for a collection of schematic terms (the arguments of R). They are either variables, abstractions or functions, namely $t ::= x \mid a.t \mid f\ ts$ where a is a variable standing for an atom and f stands for a function. We call the variable a in $a.t$ as being in *binding position.* Note that a schematic rule may contain the same variable in binding and non-binding positions (One_4 and Type_3 are examples).

Assuming an inductive definition of the predicate R, the schematic rule in (7) must be of the form

$$\frac{R\,ts_1 \;\ldots\; R\,ts_n \quad S_1\,ss_1 \;\ldots\; S_m\,ss_m}{R\,ts}\;\varrho \tag{8}$$

where the predicates $S_i\,ss_i$ (the ones different from R) stand for the side-conditions in the schematic rule.

For proving our main result in the next section it is convenient to introduce several auxiliary notions for schematic terms and rules. The following functions calculate for a schematic term the set of variables in non-binding position and the set of variables in binding position, respectively:

$$\begin{array}{rclcrcl} \mathit{vars}(x) &=& \{x\} & \qquad & \mathit{varsbp}(x) &=& \varnothing \\ \mathit{vars}(a.t) &=& \mathit{vars}(t) - \{a\} & & \mathit{varsbp}(a.t) &=& \mathit{varsbp}(t) \cup \{a\} \\ \mathit{vars}(f\;ts) &=& \mathit{vars}(ts) & & \mathit{varsbp}(f\;ts) &=& \mathit{varsbp}(ts) \end{array} \tag{9}$$

The notation $t[as;xs]$ will be used for schematic terms to indicate that the variables in binding position of t are included in as and the other variables of t are either in as or xs. That means we have for $t[as;xs]$ that $\mathit{varsbp}(t) \subseteq as$ and $\mathit{vars}(t) \subseteq as \cup xs$ hold.
We extend this notation also to schematic rules: by writing $\varrho[as;xs]$ for (8) we mean

$$\frac{R\,ts_1[as;xs] \;\ldots\; R\,ts_n[as;xs] \quad S_1\;ss_1[as;xs] \;\ldots\; S_m\;ss_m[as;xs]}{R\;ts[as;xs]}\;\varrho \tag{10}$$

However, unlike in the notation for schematic terms, we mean in $\varrho[as;xs]$ that the as stand *exactly* for the variables occurring somewhere in ϱ in binding position and the xs stand for the rest of variables. That means we have for $\varrho[as;xs]$ that $\mathit{varsbp}(\varrho) = as$ and $\mathit{vars}(\varrho) = xs$ hold, assuming suitable generalisations of the functions vars and varsbp to schematic rules. To see how the schematic notation works out in examples, reconsider the definitions for the relations One, given in (1), and Type, given in (2). Using our schematic notation for the rules, we have

$$\begin{array}{ll} \mathrm{One}_1[-;M] & \mathrm{Type}_1[-;\Gamma,x,T] \\ \mathrm{One}_2[y;M,M'] & \mathrm{Type}_2[-;\Gamma,M,N,T_1,T_2] \\ \mathrm{One}_3[-;M,N,M',N'] & \mathrm{Type}_3[x;\Gamma,M,T_1,T_2] \\ \mathrm{One}_4[y;M,N,M',N'] & \end{array}$$

where '$-$' stands for no variable in binding position.

The main property of an inductive definition, say for the inductive predicate R, is that it comes with an induction principle, which establishes a property $P\,ts$ under the assumption that $R\,ts$ holds. This means we have an induction principle diagrammatically looking as follows

$$\frac{\begin{array}{l} \ldots \\ \forall as\;xs.\; P\,ts_1[as;xs] \;\wedge\; \ldots \;\wedge\; P\,ts_n[as;xs] \;\wedge \\ \qquad\qquad S\,ss_1[as;xs] \;\wedge\; \ldots \;\wedge\; S\,ss_m[as;xs] \;\Rightarrow\; P\;ts[as;xs] \\ \ldots \end{array}}{R\;ts \Rightarrow P\;ts} \tag{11}$$

where for every schematic rule ϱ in the inductive definition we have to establish an implication. These implications state that we can assume the property for all premises and also can assume that the side-conditions hold; we have to show that the property holds for the conclusion of the schematic rule.

As explained in the introduction, we need to impose some conditions on schematic rules in order to avoid faulty reasoning and to permit an argument employing the variable convention. A rule $\varrho[as; xs]$, as given in (10), is *variable convention compatible*, short *vc-compatible*, provided the following two conditions are satisfied.

Definition 5 (Variable Convention Compatibility). *A rule $\varrho[as; xs]$ with conclusion $R\,ts$ is* vc-compatible *provided that:*

- *all functions and side-conditions occurring in ϱ are equivariant, and*
- *the side-conditions $S_1 ss_1 \wedge \ldots \wedge S_m ss_m$ imply that $as \# ts$ holds and that the as are distinct.*

If every schematic rule in an inductive definition satisfies these conditions, then the induction principle can be strengthened such that it includes a version of the variable convention.

4 Strengthening of the Induction Principle

In this section we will show how to obtain a stronger induction principle than the one given in (11). By stronger we mean that it has the variable convention already built in (this will then enable us to give slick proofs by rule induction which do not need any renaming). Formally we show that induction principles of the form

$$\frac{\begin{array}{l} \ldots \\ \forall as\; xs\; C.\; (\forall C. P\; C\; ts_1[as; xs]) \wedge\; \ldots\; \wedge (\forall C. P\; C\; ts_n[as; xs]) \wedge \\ \qquad\qquad S ss_1[as; xs]\; \wedge\; \ldots\; \wedge\; S ss_n[as; xs]\; \wedge\; \boxed{as \# C}\; \Rightarrow P\; C\; ts[as; xs] \\ \ldots \end{array}}{R\, ts \Rightarrow P\; C\; ts} \qquad (12)$$

can be used, where C stands for an *induction context*. This induction context can be instantiated appropriately (we will explain this in the next section). The only requirement we have about C is that it needs to be finitely supported. The main difference between the stronger induction principle in (12) and the weaker one in (11) is that in a proof using the stronger we can assume that the as, i.e. the variables in binding-position, are fresh with respect to the context C (see highlighted freshness-condition). This additional assumption allows us to reason as in informal "paper-and-pencil" proofs where one assumes the variable convention (we will also show this in the next section).

The first condition of vc-compatibility implies that the inductively defined predicate R is equivariant and that every schematic subterm occurring in a rule is equivariant.

Lemma 3. *(a) If all functions in a schematic term $t[as; xs]$ are equivariant, then (viewed as a function) t is equivariant, that is $\pi \bullet t[as; xs] = t[\pi \bullet as; \pi \bullet xs]$. (b) If all functions and side-conditions in the rules of an inductive definition for the predicate R are equivariant, then R is equivariant, that is if $R\,ts$ holds than also $R\,(\pi \bullet ts)$ holds.*

Proof. The first part is by a routine induction on the structure of the schematic term t. The second part is by a simple rule induction using the weak induction principle given in (11).

We now prove our main theorem: if the rules of an inductive definition are vc-compatible, then the strong induction principle in (12) holds.

Theorem 1. *Given an inductive definition for the predicate R involving vc-compatible schematic rules only, then a strong induction principle is available for this definition establishing the implication $R\,ts \Rightarrow P\,C\,ts$ with the induction context C being finitely supported.*

Proof. We need to establish $R\,ts \Rightarrow P\,C\,ts$ using the implications indicated in (12). To do so we will use the weak induction from (11) and establish that the proposition $R\,ts \Rightarrow \forall \pi\, C.P\,C\,(\pi \bullet ts)$ holds. For each schematic rule $\varrho[as; xs]$

$$\frac{R\,ts_1[as; xs]\ \ldots\ R\,ts_n[as; xs] \quad S_1\,ss_1[as; xs]\ \ldots\ S_m\,ss_m[as; xs]}{R\,ts[as; xs]}\ \varrho$$

in the inductive definition we have to analyse one case. The reasoning proceeds in each of them as follows: By induction hypothesis and side-conditions we have

$$(\forall \pi\, C.P\,C\,(\pi \bullet ts_1[as; xs]))\ \ldots\ (\forall \pi\, C.P\,C\,(\pi \bullet ts_n[as; xs])) \tag{13}$$

$$S_1\,ss_1[as; xs]\ \ldots\ S_m\,ss_m[as; xs] \tag{14}$$

hold. Since ϱ is assumed to be vc-compatible, we have by Lem. 3 that (*) $\pi \bullet ts_i[as; xs]$ is equal to $ts_i[\pi \bullet as; \pi \bullet xs]$ in (13). For (14) we can further infer from the vc-compatibility of ϱ that

$$(a)\ as \# ts[as; xs] \qquad \text{and} \qquad (b)\ distinct(as) \tag{15}$$

hold. We have to show that $P\,C\,(\pi \bullet ts[as; xs])$ holds, which because of Lem. 3 is equivalent to $P\,C\,ts[\pi \bullet as; \pi \bullet xs]$.

The proof proceeds by using Lem. 2 and choosing for every atom a in as a fresh atom c such that for all the cs the following holds:

$$(a)\ cs \# ts[\pi \bullet as; \pi \bullet xs] \quad (b)\ cs \# \pi \bullet as \quad (c)\ cs \# C \quad (d)\ distinct(cs) \tag{16}$$

Such cs always exists: the first and the second property can be obtained since the schematic terms $ts[\pi \bullet as; \pi \bullet xs]$ and $\pi \bullet as$ stand for finitely supported objects; the third can also be obtained since we assumed that the induction context C is finitely supported; the last can be obtained by choosing the cs one after another avoiding the ones that have already been chosen.

Now we form the permutation $\pi' \stackrel{\text{def}}{=} (\pi\bullet as\ \ cs)$ where $(\pi\bullet as\ \ cs)$ stands for the sequence of swappings $(\pi\bullet a_1\ \ c_1)\ldots(\pi\bullet a_j\ \ c_j)$. Since permutations are bijective renamings, we can infer from (15.b) that $distinct(\pi\bullet as)$ holds. This and the fact in (16.d) implies that

$$\pi'@\pi\bullet as = \pi'\bullet(\pi\bullet as) = cs \tag{17}$$

We then instantiate the π in the induction hypotheses given in (13) with $\pi'@\pi$ and obtain using (17) and (*) so that

$$(\forall C.P\,C\,ts_1[cs;\pi'@\pi\bullet xs]))\ \ldots\ (\forall C.P\,C\,ts_n[cs;\pi'@\pi\bullet xs])) \tag{18}$$

hold. Since the rule ϱ is vc-compatible, we can infer from (14) and the equivariance of the side-conditions that

$$S_1\ ss_1[cs;\pi'@\pi\bullet xs]\ \ldots\ S_m\ ss_m[cs;\pi'@\pi\bullet xs] \tag{19}$$

hold (we use here the fact that $\pi'@\pi\bullet(ss_i[as;xs])$ is equal to $ss_i[cs;\pi'@\pi\bullet xs]$). From (16.$c$), (18), (19) and the implication from the strong induction principle we can infer $P\ C\ ts[cs;\pi'@\pi\bullet xs]$ which by Lem. 3 is equivalent to

$$P\ C\ \pi'\bullet ts[\pi\bullet as;\pi\bullet xs] \tag{20}$$

From (15.a) we can by Lem. 1(a) infer that $\pi\bullet as\ \#\ ts[\pi\bullet as;\pi\bullet xs]$ holds. This however implies by (16.a) and by repeated application of Lem. 1(b) that

$$\pi'\bullet ts[\pi\bullet as;\pi\bullet xs] = ts[\pi\bullet as;\pi\bullet xs] \tag{21}$$

Substituting this equation into (20) establishes the proof obligation for the rule ϱ. Provided we analysed all such cases, we have shown $R\,ts \Rightarrow \forall\pi\,C.P\,C\,(\pi\bullet ts)$. We obtain our original goal by instantiating π with the identity permutation. □

5 Examples

We can now apply our technique to the examples from the Introduction.

5.1 Simple Typing

Given the typing relation defined in (2), we must first check the conditions spelt out in Definition 5. The first condition is that all of the definition's functions (namely *var*, *app*, *lam* and ::) and side-conditions (namely *valid*, $\in$ and $\#$) must be equivariant. This is easily confirmed (see Remark 1). The second condition requires that all variables in binding positions be distinct (there is just one, the x in Type_3); and that it be fresh for all the terms appearing in the conclusion of that rule, namely $\Gamma \vdash lam(x.M) : T_1 \to T_2$, under the assumption that the side-condition, $x\ \#\ \Gamma$, of this rule holds.

In this case, therefore, we must check that $x\ \#\ \Gamma$, $x\ \#\ lam(x.M)$ and $x\ \#\ T_1 \to T_2$ hold. The first is immediate given our assumption; the second follows from the definition of support for lambda-terms ($x\ \#\ lam(x.M)$ for all x and M); and the third follows from the definition of support for types (we define permutation on types T as $\pi\bullet T \stackrel{\text{def}}{=} T$ and thus obtain that $\mathtt{supp}(T) = \varnothing$).

With these conditions established, Theorem 1 tells us that the strong, or vc-compatible principle exists, and that it is

$$\frac{\begin{array}{l}\forall \Gamma\, x\, T\, C.\ valid(\Gamma) \wedge (x:T) \in \Gamma \Rightarrow P\ C\ \Gamma\ (var(x))\ T \\ \forall \Gamma\, M\, N\, T_1\, T_2\, C.\ (\forall C.\ \ P\ C\ \Gamma\ M\ (T_1 \to T_2))\ \wedge\ (\forall C.\ \ P\ C\ \Gamma\ N\ T_1)\ \Rightarrow \\ \qquad\qquad\qquad P\ C\ \Gamma\ (app(M,N))\ T_2 \\ \forall \Gamma\, x\, M\, T_1\, T_2\, C.\ x \# \Gamma\ \wedge\ (\forall C.\, P\ C\ ((x:T_1)::\Gamma)\ M\ T_2)\ \wedge\ x \# C\ \Rightarrow \\ \qquad\qquad\qquad P\ C\ \Gamma\ (lam(x.M))\ (T_1 \to T_2)\end{array}}{\Gamma \vdash M : T\ \Rightarrow\ P\ C\ \Gamma\ M\ T}$$

This principle can now be used to establish the weakening result. The statement is

$$\Gamma \vdash M : T\ \Rightarrow\ \Gamma \subseteq \Gamma'\ \Rightarrow\ valid(\Gamma')\ \Rightarrow\ \Gamma' \vdash M : T \tag{22}$$

With the strong induction principle, the formal proof of this statement proceeds like the informal one given in the Introduction. There, in the Type_3 case, we used the variable convention to assume that the bound x was fresh for Γ'. Given this information, we instantiate the induction context C in the strong induction principle with Γ' (which is finitely supported). The complete instantiation of the vc-compatible induction principle is

$$\begin{array}{c}P = \lambda \Gamma\, M\, T\, \Gamma'.\ \ \Gamma \subseteq \Gamma' \Rightarrow valid(\Gamma') \Rightarrow \Gamma' \vdash M : T \\ C = \Gamma' \qquad\qquad \Gamma = \Gamma \qquad\qquad M = M \qquad\qquad T = T\end{array}$$

which after some beta-contractions gives us the statement in (22). The induction cases are then as follows (stripping off the outermost quantifiers):

(1) $valid(\Gamma)\ \wedge\ (x:T) \in \Gamma\ \Rightarrow\ \Gamma \subseteq \Gamma'\ \Rightarrow\ valid(\Gamma') \Rightarrow \Gamma' \vdash var(x) : T$

(2) $(\forall \Gamma''.\ \Gamma \subseteq \Gamma'' \Rightarrow valid(\Gamma'') \Rightarrow \Gamma'' \vdash M_1 : T_1 \to T_2) \wedge$
$(\forall \Gamma''.\ \Gamma \subseteq \Gamma'' \Rightarrow valid(\Gamma'') \Rightarrow \Gamma'' \vdash M_2 : T_1)\ \Rightarrow$
$\Gamma \subseteq \Gamma'\ \Rightarrow\ valid(\Gamma')\ \Rightarrow\ \Gamma' \vdash app(M_1, M_2) : T_2$

(3) $(\forall \Gamma''.\ (x:T_1)::\Gamma \subseteq \Gamma''\ \Rightarrow valid(\Gamma'')\ \Rightarrow\ \Gamma'' \vdash M : T_2)\ \wedge\ x \# \Gamma'\ \Rightarrow$
$\Gamma \subseteq \Gamma'\ \Rightarrow\ valid(\Gamma')\ \Rightarrow\ \Gamma' \vdash lam(x.M) : T_1 \to T_2$

The first two cases are trivial. For (3), we instantiate Γ'' in the induction hypothesis to be $(x : T_1) :: \Gamma'$. From the assumption $\Gamma \subseteq \Gamma'$ we have $(x : T_1) :: \Gamma \subseteq (x : T_1) :: \Gamma'$. Moreover from the assumption $valid(\Gamma')$ we also have $valid((x : T_1)::\Gamma')$ using the variable convention's $x \# \Gamma'$. Hence we can derive $(x : T_1)::\Gamma' \vdash M : T_2$ using the induction hypothesis. Now applying rule Type_3 we can obtain $\Gamma' \vdash lam(x.M) : T_1 \to T_2$, again using the variable convention's $x \# \Gamma'$. This completes the proof. Its *readable* version expressed in Isabelle's Isar-language [12] and using the nominal datatype package [9] is shown in Fig. 1.

By way of contrast, recall that a proof without the stronger induction principle would not be able to assume anything about the relationship between x and Γ', forcing the prover to α-convert $lam(x.M)$ to a form with a new and suitably fresh bound variable, $lam(z.((z\, x)\bullet M))$, say. At this point, the simplicity of the proof using the variable convention disappears: the inductive hypothesis is much harder to show applicable because it mentions M, but the desired goal is in terms of $(z\, x)\bullet M$.

lemma *weakening*:
  assumes a_1: $\Gamma \vdash M : T$ **and** a_2: $\Gamma \subseteq \Gamma'$ **and** a_3: *valid* Γ'
  shows $\Gamma' \vdash M : T$
using a_1 a_2 a_3
proof (*nominal-induct* Γ M T *avoiding*: Γ' *rule*: *strong-typing-induct*)
  case (*Type*$_3$ x Γ T_1 T_2 M)
  have *vc*: $x \# \Gamma'$ **by** *fact* — variable convention
  have *ih*: $(x{:}T_1)::\Gamma \subseteq (x{:}T_1)::\Gamma' \Longrightarrow valid\ ((x{:}T_1)::\Gamma') \Longrightarrow (x{:}T_1)::\Gamma' \vdash M : T_2$ **by** *fact*
  have $\Gamma \subseteq \Gamma'$ **by** *fact*
  then have $(x{:}T_1)::\Gamma \subseteq (x{:}T_1)::\Gamma'$ **by** *simp*
  moreover
  have *valid* Γ' **by** *fact*
  then have *valid* $((x{:}T_1)::\Gamma')$ **using** *vc* **by** (*simp add*: *valid-cons*)
  ultimately have $(x{:}T_1)::\Gamma' \vdash M : T_2$ **using** *ih* **by** *simp*
  with *vc* **show** $\Gamma' \vdash lam(x.M) : T_1 \to T_2$ **by** *auto*
qed (*auto*) — cases Type$_1$ and Type$_2$

Fig. 1. A readable Isabelle-Isar proof for the weakening lemma using the strong induction principle of the typing relation. The stronger induction principle allows us to assume a variable convention, in this proof $x \ \# \ \Gamma'$, which makes the proof to go through without difficulties.

5.2 Parallel Reduction

In [2], the central lemma of the proof for the Church-Rosser property of beta-reduction is the substitutivity property of the $\underset{1}{\longrightarrow\!\!\!\!\!\rightarrow}$-reduction. To formalise this proof while preserving the informal version's simplicity, we will need the strong induction principle for $\underset{1}{\longrightarrow\!\!\!\!\!\rightarrow}$.

Before proceeding, we need two important properties of the substitution function, which occurs in the redex rule One$_4$. We characterise the action of a permutation over a substitution (showing that substitution is equivariant), and the support of a substitution. Both proofs are by straightforward vc-compatible *structural* induction over M:

$$\pi \bullet (M[x := N]) = (\pi \bullet M)[(\pi \bullet x) := (\pi \bullet N)] \tag{23}$$

$$\mathtt{supp}(M[x := N]) \subseteq (\mathtt{supp}(M) - \{x\}) \cup \mathtt{supp}(N) \tag{24}$$

With this we can start to check the vc-compatibility conditions: the condition about equivariance of functions and side-conditions is again easily confirmed. The second condition is that bound variables are free in the relation's rules' conclusions. In rule One$_2$, this is trivial because $y \ \# \ lam(y.M)$ and $y \ \# \ lam(y.M')$ hold. A problem arises, however, with rule One$_4$. Here we have to show that $y \ \# \ app(lam(y.M), N)$ and $y \ \# \ M'[y := N']$, and we have no assumptions to hand about y.

It is certainly true that y is fresh for $lam(y.M)$, but it may occur in N. As for the term $M'[y := N']$, we know that any occurrences of y in M' will be masked by the substitution (see (24)), but y may still be free in N'.

We need to reformulate One_4 to read

$$\frac{y \mathrel{\#} N \quad y \mathrel{\#} N' \quad M \underset{1}{\longrightarrow\!\!\!\!\rightarrow} M' \quad N \underset{1}{\longrightarrow\!\!\!\!\rightarrow} N'}{app(lam(y.M), N) \underset{1}{\longrightarrow\!\!\!\!\rightarrow} M'[y := N']}\ \text{One}_4''$$

so that the vc-compatibility conditions can be discharged. In other words, if we have rule One_4'' we can apply Theorem 1, but not if we use One_4. This is annoying because both versions can be shown to define the same relation, but we have no general, and automatable, method for determining this. For the moment, we reject rule One_4 and require the user of the nominal datatype package to use One_4''. If this is done, the substitutivity lemma is almost automatic:

lemma *substitutivity-aux*:
  assumes a: $N \longrightarrow_1 N'$
  shows $M[x{:=}N] \longrightarrow_1 M[x{:=}N']$
using a **by** (*nominal-induct M avoiding*: *x N N′ rule*: *strong-lam-induct*) (*auto*)

lemma *subtitutivity*:
  assumes a_1: $M \longrightarrow_1 M'$ **and** a_2: $N \longrightarrow_1 N'$
  shows $M[x{:=}N] \longrightarrow_1 M'[x{:=}N']$
using a_1 a_2 **by** (*nominal-induct M M′ avoiding*: *N N′ x rule*: *strong-parallel-induct*)
  (*auto simp add*: *substitutivity-aux substitution-lemma fresh-atm*)

The first lemma is proved by a vc-compatible *structural* induction over M; the second, the actual substitutivity property, is proved by a vc-compatible *rule* induction relying on the substitution lemma, and the lemma *fresh-atm*, which states that $x \mathrel{\#} y$ is the same as $x \neq y$ when y is an atom.

6 Related Work

Apart from our own preliminary work in this area [10], we believe the prettiest formal proof of the weakening lemma to be that in Pitts [8]. This proof uses the equivariance property of the typing relation, and includes a renaming step using permutations. Because of the pleasant properties that permutations enjoy (they are bijective renamings, in contrast to substitutions which might identify two names), the renaming can be done with relatively minimal overhead. Our contribution is that we have built this renaming into our vc-compatible induction principles once and for all. Proofs using the vc-compatible principles then do not need to perform any explicit renaming steps.

Somewhat similar to our approach is the work of Pollack and McKinna [6]. Starting from the standard induction principle that is associated with an inductive definition, we derived an induction principle that allows emulation of Barendregt's variable convention. Pollack and McKinna, in contrast, gave a "weak" and "strong" version of the typing relation. These versions differ in the way the rule for abstractions is stated:

$$\frac{x \mathrel{\#} M \quad (x : T_1) :: \Gamma \vdash M[y := x] : T_2}{\Gamma \vdash lam(y.M) : T_1 \to T_2}\ \text{weak}$$

$$\frac{\forall x.\ x \mathrel{\#} \Gamma \Rightarrow (x : T_1) :: \Gamma \vdash M[y := x] : T_2}{\Gamma \vdash lam(y.M) : T_1 \to T_2}\ \text{strong}$$

They then showed that both versions derive the same typing judgements. With this they proved the weakening lemma using the "strong" version of the principle, while knowing that the result held for the "weak" relation as well. The main difference between this and our work seems to be of convenience: we can relatively easily derive, in a uniform way, an induction principle for vc-compatible relations (we have illustrated this point with two examples). Achieving the same uniformity in the style of McKinna and Pollack does not seem as straightforward.

7 Future Work

Our future work will concentrate on two aspects: first on generalising our definition of schematic rules so that they may, for example, include quantifiers. Second on being more liberal about which variables can be included in the induction context. To see what we have in mind with this, recall that we allowed in the induction context only variables that are in binding position. However there are examples where this is too restrictive: for example Crary gives in [4, Page 231] the following mutual inductive definition for the judgements $\Gamma \vdash s \Leftrightarrow t : T$ and $\Gamma \vdash p \leftrightarrow q : T$ (they represent a type-driven equivalence algorithm for lambda-terms with constants):

$$\frac{s \Downarrow p \quad t \Downarrow q \quad \Gamma \vdash p \leftrightarrow q : T}{\Gamma \vdash s \Leftrightarrow t : b}\,\mathrm{Ae}_1 \quad \frac{(x : T_1) :: \Gamma \vdash s\,x \Leftrightarrow t\,x : T_2}{\Gamma \vdash s \Leftrightarrow t : T_1 \to T_2}\,\mathrm{Ae}_2 \quad \frac{}{\Gamma \vdash s \Leftrightarrow t : \mathit{unit}}\,\mathrm{Ae}_3$$

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x \leftrightarrow x : T}\,\mathrm{Pe}_1 \quad \frac{\Gamma \vdash p \leftrightarrow q : T_1 \to T_2 \quad \Gamma \vdash s \leftrightarrow t : T_1}{\Gamma \vdash p\,s \leftrightarrow q\,t : T_2}\,\mathrm{Pe}_2 \quad \frac{}{\Gamma \vdash k \leftrightarrow k : b}\,\mathrm{Pe}_3$$

What is interesting is that these rules do not contain any variable in binding position. Still, in some proofs by induction over those rules one wants to be able to assume that the variable x in the rule Ae_2 satisfies certain freshness conditions. Our implementation already deals with this situation by explicitly giving the information that x should appear in the induction context. However, we have not yet worked out the theory.

8 Conclusion

In the POPLmark Challenge [1], the proof of the weakening lemma is described as a "straightforward induction". In fact, mechanising this informal proof is *not* straightforward at all (see for example [6, 5, 8]). We have given a novel rule induction principle for the typing relation that makes proving the weakening lemma mechanically as simple as performing the informal proof.

Importantly, this new principle can be derived from the original inductive definition of the typing relation in a mechanical way. This method extends our earlier work [10, 7], where we constructed our new induction principles by hand. By formally deriving principles that avoid the need to rename bound variables, we advance the state-of-the-art in mechanical theorem-proving over syntax with binders. The results of this paper have already been used many times in the nominal datatype package: for example in the proofs of the CR and SN properties in the λ-calculus, in proofs about the pi-calculus, in proofs about logical relations and in several proofs from structural operational semantics.

The fact that our technique may require users to cast some inductive definitions in alternative forms is unfortunate. In the earlier [10], our hand-proofs correctly derived a vc-compatible principle from the original definition of $\underset{1}{\twoheadrightarrow}$; we hope that future work will automatically justify comparable derivations.

Acknowledgements We are very grateful to Andrew Pitts for the many discussions with him on the subject of this paper.

References

1. B. E. Aydemir, A. Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. Mechanized Metatheory for the Masses: The PoplMark Challenge. In *Proc. of the 18th International Conference on Theorem Proving in Higher-Order Logics (TPHOLs)*, volume 3603 of *LNCS*, pages 50–65, 2005.
2. H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1981.
3. J. Bengtson and J. Parrow. Formalising the pi-Calculus using Nominal Logic. In *Proc. of the 10th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 4423 of *LNCS*, pages 63–77, 2007.
4. K. Crary. *Advanced Topics in Types and Programming Languages*, chapter Logical Relations and a Case Study in Equivalence Checking, pages 223–244. MIT Press, 2005.
5. J. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row, 1986.
6. J. McKinna and R. Pollack. Some type theory and lambda calculus formalised. *Journal of Automated Reasoning*, 23(1-4), 1999.
7. M. Norrish. Mechanising λ-calculus using a classical first order theory of terms with permutation. *Higher-Order and Symbolic Computation*, 19:169–195, 2006.
8. A. M. Pitts. Nominal Logic, A First Order Theory of Names and Binding. *Information and Computation*, 186:165–193, 2003.
9. C. Urban and S. Berghofer. A Recursion Combinator for Nominal Datatypes Implemented in Isabelle/HOL. In *Proc. of the 3rd International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *LNAI*, pages 498–512, 2006.
10. C. Urban and M. Norrish. A formal treatment of the Barendregt Variable Convention in rule inductions. In *MERLIN '05: Proceedings of the 3rd ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding*, pages 25–32, New York, NY, USA, 2005. ACM Press.
11. C. Urban and C. Tasson. Nominal techniques in Isabelle/HOL. In *Proc. of the 20th International Conference on Automated Deduction (CADE)*, volume 3632 of *Lecture Notes in Computer Science*, pages 38–53, 2005.
12. M. Wenzel. Isar — A Generic Interpretative Approach to Readable Formal Proof Documents. In *Proc. of the 12th International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, number 1690 in LNCS, pages 167–184, 1999.