How to Prove False using the Variable Convention

Christian Urban, Technical University of Munich
(Email: **urbanc@in.tum.de**)

Abstract

Bound variables play an important role in many branches of formal methods. Nearly all informal induction proofs involving bound variables make use of the variable convention. This poster shows by giving an example that this convention is in general an **unsound** reasoning principle, i.e. one can use it to prove false. The poster also shows how Nominal Isabelle implements this reasoning principle in a safe manner and illustrates its use in a formal proof of the substitution lemma.

Informal Reasoning in the "Wild"

The variable convention is perhaps one of the most frequently used reasoning principles when reasoning informally about syntax involving binders. Barendregt formulates this convention in his classic book, "The Lambda-Calculus: Its Syntax and Semantics", as follows:

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.

He uses it in proofs like the following:

Substitution Lemma: If $x \neq y$ and $x \notin FV(L)$, then
$$M[x := N][y := L] = M[y := L][x := N[y := L]].$$

Proof: By induction on the structure of M.

- **Case 1:** M is a variable.
  Case 1.1. $M = x$. Then both sides equal $N[y := L]$ since $x \neq y$.
  Case 1.2. $M = y$. Then both sides equal L, for $x \notin FV(L)$ implies $L[x := \ldots] = L$.
  Case 1.3. $M = z \neq x, y$. Then both sides equal z.
- **Case 2:** $M = \lambda z.M_1$. By the variable convention we may assume that $z \neq x, y$ and z is not free in N, L.
$$\begin{aligned}(\lambda z.M_1)[x := N][y := L] &= \lambda z.(M_1[x := N][y := L]) \\ &= \lambda z.(M_1[y := L][x := N[y := L]]) \\ &= (\lambda z.M_1)[y := L][x := N[y := L]].\end{aligned}$$
- **Case 3:** $M = M_1 M_2$. The statement follows again from the induction hypothesis. □

The point of Nominal Isabelle is to provide all proving technologies necessary for conveniently formalising such proofs. It turns out, however, that one has to be careful with the variable convention.

Faulty Reasoning with the Variable Convention

To see why the variable convention is in general an unsound reasoning principle, consider the following inductively defined relation taking two lambda-terms as arguments:

Relation of Interest:
$$\frac{}{x \mapsto x}\ \text{SVar} \qquad \frac{}{M_1\, M_2 \mapsto M_1\, M_2}\ \text{SApp} \qquad \frac{M \mapsto M'}{\lambda x.M \mapsto M'}\ \text{SLam}$$

Note that the SLam-rule reads as "for all x, M and M', if $M \mapsto M'$ is in the relation, then so is $\lambda x.M \mapsto M'$". Proving a property by induction over an inductively defined relation means that we have to establish this property for each conclusion of the rules, assuming the property holds already for the corresponding premises. If we use the variable convention, however, we can "prove" in this way the following faulty lemma.

Faulty Lemma: Suppose $M \mapsto M'$. If $y \notin FV(M)$ then $y \notin FV(M')$.

The "proof" goes as follows:

- **Cases SVar and SApp:** These two cases are straightforward as we only have to establish:
  - if $y \notin FV(x)$ then $y \notin FV(x)$, and
  - if $y \notin FV(M_1\, M_2)$ then $y \notin FV(M_1\, M_2)$
- **Case SLam:** In this case we have the induction hypothesis
$$\text{if } y \notin FV(M) \text{ then } y \notin FV(M')$$
and the assumption that $y \notin FV(\lambda x.M)$. The goal is to show $y \notin FV(M')$. We use the variable convention to infer that $y \neq x$ where x is the bound variable from $\lambda x.M$ and y is the free variable from the lemma. Using this fact, we know that $y \notin FV(\lambda x.M)$ holds if and only if $y \notin FV(M)$ holds. Hence we can use the induction hypothesis to conclude also this case. "□"

This proof is of course bogus and we can easily find a counter example.

Counter Example: We have that $\lambda x.x \mapsto x$ is in the relation and $x \notin FV(\lambda x.x)$ holds. But $x \notin FV(x)$ is clearly false. Therefore we have a contradiction, i.e. can prove false.

There are other similar examples. The formal reasoning in Nominal Isabelle is protected from such bogus reasoning, as it builds the variable convention into the induction principles and in case of rule inductions derives them only if the relations satisfy a condition ensuring that they are compatible with the variable convention. This compatibility condition states (roughly):

- the relation must be equivariant (closed under permutations) and
- every binder in a rule must not be free in the conclusion of that rule.

The "interesting" relation on the left-hand side does not satisfy this condition, but the typing relation for simply-typed lambda terms, for example, does.

Typing Relation for Simple Types:
$$\frac{valid\,\Gamma \quad (x, T) \in \Gamma}{\Gamma \vdash x : T}\ \text{TVar} \qquad \frac{\Gamma \vdash M_1 : T_1 \to T_2 \quad \Gamma \vdash M_2 : T_1}{\Gamma \vdash M_1\, M_2 : T_2}\ \text{TApp} \qquad \frac{(x, T_1) :: \Gamma \vdash M : T_2}{\Gamma \vdash \lambda x.M : T_1 \vdash T_2}\ \text{TLam}$$

Consequently, you can use the variable convention in proofs about the typing relation. So there is no need to be afraid of this convention in Nominal Isabelle: on the contrary, it is a reasoning principle which simplifies many formal arguments!

Formal Reasoning in Nominal Isabelle

Nominal Isabelle automatically builds the variable convention into the structural induction principles and if the relation is compatible with the variable convention also into rule inductions. For example in addition to the usual induction principle for lambda-terms

Simple Induction Principle for Lambda-Terms
$$\frac{\begin{array}{l}\forall x.\ P\ x \\ \forall M_1\ M_2.\ P\ M_1 \wedge P\ M_2 \Longrightarrow P\ (M_1\, M_2) \\ \forall x\, M.\ P\ M \Longrightarrow P\ (\lambda x.M)\end{array}}{P\ M}$$

Nominal Isabelle derives also the following stronger induction principle which includes an "avoiding" context that corresponds to the variable convention

Stronger Induction Principle
$$\frac{\begin{array}{l}\forall x\ c.\ P\ c\ x \\ \forall M_1\ M_2\ c.\ (\forall d.\ P\ d\ M_1) \wedge (\forall d.\ P\ d\ M_2) \Longrightarrow P\ c\ (M_1\, M_2) \\ \forall x\, M\ c.\ x \# c \wedge (\forall d.\ P\ d\ M) \Longrightarrow P\ c\ (\lambda x.M)\end{array}}{P\ c\ M}$$

The stronger induction principle allows you to formally prove the substitution lemma with ease:

```
lemma substitution_lemma:
  assumes a: "x≠y" and  b: "x # L"
  shows "M[x:=N][y:=L] = M[y:=L][x:=N[y:=L]]"
using a b by (nominal_induct M avoiding: x y N L rule: lam.induct)
             (auto simp add: forget fresh_fact)
```

. . . if you prefer more details:

```
lemma substitution_lemma:
  assumes a: "x≠y" and  b: "x # L"
  shows "M[x:=N][y:=L] = M[y:=L][x:=N[y:=L]]"
using a b
proof (nominal_induct M avoiding: x y N L rule: lam.induct)
  case (Var z)                                                    (Case 1: variables)
  show "Var z[x:=N][y:=L] = Var z[y:=L][x:=N[y:=L]]"   (is "?LHS = ?RHS")
  proof -
    { assume  "z=x"                                               (Case 1.1)
      have "(1)": "?LHS = N[y:=L]" using 'z=x' by simp
      have "(2)": "?RHS = N[y:=L]" using 'z=x' 'x≠y' by simp
      from "(1)" "(2)" have "?LHS = ?RHS"  by simp }
    moreover
    { assume "z=y" and "z≠x"                                      (Case 1.2)
      have "(1)": "?LHS = L"                 using 'z≠x' 'z=y' by simp
      have "(2)": "?RHS = L[x:=N[y:=L]]" using 'z=y' by simp
      have "(3)": "L[x:=N[y:=L]] = L"        using 'x # L' by (simp add: forget)
      from "(1)" "(2)" "(3)" have "?LHS = ?RHS" by simp }
    moreover
    { assume "z≠x" and "z≠y"                                      (Case 1.3)
      have "(1)": "?LHS = Var z" using 'z≠x' 'z≠y' by simp
      have "(2)": "?RHS = Var z" using 'z≠x' 'z≠y' by simp
      from "(1)" "(2)" have "?LHS = ?RHS" by simp }
    ultimately show "?LHS = ?RHS" by blast
  qed
next
  case (Lam z M1)                                                 (Case 2: lambdas)
  have ih: "⟦x≠y; x # L⟧ ⟹ M1[x:=N][y:=L] = M1[y:=L][x:=N[y:=L]]" by fact
  have fs: "z # x" "z # y" "z # N" "z # L" by fact+               (variable convention)
  hence "z # N[y:=L]" by (simp add: fresh_fact)
  show "(Lam [z].M1)[x:=N][y:=L] = (Lam [z].M1)[y:=L][x:=N[y:=L]]" (is "?LHS=?RHS")
  proof -
    have "?LHS = Lam [z].(M1[x:=N][y:=L])" using 'z # x' 'z # y' 'z # N' 'z # L' by simp
    also from ih have "... = Lam [z].(M1[y:=L][x:=N[y:=L]])" using 'x≠y' 'x # L' by simp
    also have "... = (Lam [z].(M1[y:=L]))[x:=N[y:=L]]" using 'z # x' 'z # N[y:=L]' by simp
    also have "... = ?RHS" using  'z # y' 'z # L' by simp
    finally show "?LHS = ?RHS" .
  qed
next
  case (App M1 M2) thus "(App M1 M2)[x:=N][y:=L] = (App M1 M2)[y:=L][x:=N[y:=L]]" by simp   (Case 3: app's)
qed
```

Acknowledgments: This work arose from joint work with Michael Norrish, Stefan Berghofer and Randy Pollack.

Nominal Isabelle can be downloaded from **http://isabelle.in.tum.de/nominal/** .

Presented at the workshop on the occasion of Prof. Mike Gordon's 60th birthday, Royal Society London, 25 March 2008