# Nominal Techniques

## or, How Not to be Intimidated by the Variable Convention

Christian Urban (TU Munich)

http://isabelle.in.tum.de/nominal/

**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.

Barendregt in "The Lambda-Calculus: Its Syntax and Semantics"

# Nominal Techniques

- In 2000 I did my PhD on a strong normalisation result. I had very good reviewers:

Andy Pitts

Henk Barendregt

# Nominal Techniques

- In 2000 I did my PhD on a strong normalisation result. I had very good reviewers:

Andy Pitts

Henk Barendregt

- Kleene in a journal paper: “We thank T. Thacher Robinson for showing us on August 19, 1962 by a counterexample the existence of an error in our handling of bound variables.”

# Nominal Techniques

- **Xavier Leroy** in his PhD: We define the set SchTyp of type schemes, with typical element $\sigma$, by the following grammar:

  $$\sigma ::= \forall\{\alpha_1..\alpha_n\}.\tau$$

  In this syntax, the quantified variables $\alpha_1..\alpha_n$ are treated as a set of variables: their relative order is not significant, and they are assumed to be distinct. ... We identify two type schemes that differ only by a renaming of the variables bound by $\forall$ ($\alpha$-conversion operation), and by the introduction or suppression of quantified variables that are not free in the type part. More precisely, we quotient the set of schemes by the following two equations:

  $$\begin{array}{ll} \forall\{\alpha_1..\alpha_n\}.\tau = \forall\{\beta_1..\beta_n\}.(\tau[\alpha_1 := \beta_1..\alpha_n := \beta_n]) & \\ \forall\{\alpha, \alpha_1..\alpha_n\}.\tau = \forall\{\alpha_1..\alpha_n\}.\tau & \text{if } \alpha \text{ not in } \mathsf{fv}(\tau) \end{array}$$

# Nominal Techniques

- **Xavier Leroy** in his PhD: We define the set SchTyp of type schemes, with typical element $\sigma$, by the following grammar:

$$\sigma ::= \forall\{\alpha_1..\alpha_n\}.\tau$$

In this syntax, the quantified variables $\alpha_1..\alpha_n$ are treated as a set of gnificant, and they ar fy two type schemes that differ only by a renaming of the variables bound by $\forall$ ($\alpha$-conversion operation), and by the introduction or suppression of quantified variables that are not free in the type part. More precisely, we quotient the set of schemes by the following two equations:

$$\forall\{\alpha\}.\alpha \to \alpha \;=_\alpha \forall\{\beta\}.\alpha \to \beta$$

$$\forall\{\alpha_1..\alpha_n\}.\tau = \forall\{\beta_1..\beta_n\}.(\tau[\alpha_1 := \beta_1..\alpha_n := \beta_n])$$

$$\forall\{\alpha, \alpha_1..\alpha_n\}.\tau = \forall\{\alpha_1..\alpha_n\}.\tau \qquad \text{if } \alpha \text{ not in } \mathsf{fv}(\tau)$$

# Nominal Techniques

- Moral of my PhD: The reviewers did not find any errors, also the reviewers of a conference and journal paper.

# Nominal Techniques

- Moral of my PhD: The reviewers did not find any errors, also the reviewers of a conference and journal paper.

- The result was correct, but I did find errors in the proof (in quite central lemmas).

# Nominal Techniques

- Moral of my PhD: The reviewers did not find any errors, also the reviewers of a conference and journal paper.
- The result was correct, but I did find errors in the proof (in quite central lemmas).
- Starting from around 2000, Andy Pitts introduced many ideas about the proper handling of bound names. One central idea of him is:

**Use permutations instead of renaming substitutions.**

# Plan of the Lectures

1.) **Thursday:** How to deal with the variable convention: “Can always pick bound variables to avoid clashes with other variables”.

2.) **Friday:** How to deal with stetaments such as “Expressions differing only in names of bound variables are equivalent”.

3.) **Saturday:** The Real Thing: I hope to walk you through a formalisation of a small CK Machine.

# Plan of the Lectures

1.) **Thursday:** How to deal with the variable convention: “Can always pick bound variables to avoid clashes with other variables”.

2.) **Friday:** How to deal with stetaments such as “Expressions differing only in names of bound variables are equivalent”.

3.) **Saturday:** The Real Thing: I hope to walk you through a formalisation of a small CK Machine.

- I will show you formalised proofs, but the lectures won’t be hands-on. If you need help, I am here until Thursday. **Please ask me!!**

# Plan

- We will have a look at the substitution and weakening lemma.
- I will show you an example where the variable convention leads to faulty reasoning.
- We derive a structural induction principle for lambda-terms that is safe and has the variable convention already built in.

# Plan

- We will have a look at the substitution and weakening lemma.
- I will show you an example where the variable convention leads to faulty reasoning.
- We derive a structural induction principle for lambda-terms that is safe and has the variable convention already built in.
- The **main point** of nominal techniques is to make sense out of informal reasoning.

**Substitution Lemma:** If $x \not\equiv y$ and $x \notin \mathsf{fv}(L)$, then

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$$

**Proof:** By induction on the structure of $M$.

- **Case 1:** $M$ is a variable.

  Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

  Case 1.2. $M \equiv y$. Then both sides equal $L$, for $x \notin \mathsf{fv}(L)$ implies $L[x := \ldots] \equiv L$.

  Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal $z$.

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and $z$ is not free in $N, L$.

  $$\begin{aligned}(\lambda z.M_1)[x := N][y := L] &\equiv \lambda z.(M_1[x := N][y := L])\\ &\equiv \lambda z.(M_1[y := L][x := N[y := L]])\\ &\equiv (\lambda z.M_1)[y := L][x := N[y := L]].\end{aligned}$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis. □

**Substitution Lemma:** If $x \not\equiv y$ and $x \notin \mathsf{fv}(L)$, then

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$$

**Proof:** By induction on the structure of $M$.

- **Case 1:** $M$ is a variable.

  Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

  Case 1.2. $M \equiv y$. Then both sides equal $L$, for $x \notin \mathsf{fv}(L)$ implies $L[x := \ldots] \equiv L$.

  Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal $z$.

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and $z$ is not free in $N, L$.

$$\begin{aligned}(\lambda z.M_1)[x := N][y := L] &\equiv \lambda z.(M_1[x := N][y := L]) \\ &\equiv \lambda z.(M_1[y := L][x := N[y := L]]) \\ &\equiv (\lambda z.M_1)[y := L][x := N[y := L]].\end{aligned}$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis. □

**Substitution Lemma:** If $x \not\equiv y$ and $x \notin \mathsf{fv}(L)$, then
$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$$

**Proof:** By induction on the structure of $M$.

- **Case 1:** $M$ is a variable.

  Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

  Case 1.2. $M \equiv y$. Then both sides equal $L$, for $x \notin \mathsf{fv}(L)$ implies $L[x := \ldots] \equiv L$.

  Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal $z$.

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and $z$ is not free in $N, L$.

$$\begin{aligned}(\lambda z.M_1)[x := N][y := L] &\equiv \lambda z.(M_1[x := N][y := L])\\ &\equiv \lambda z.(M_1[y := L][x := N[y := L]])\\ &\equiv (\lambda z.M_1)[y := L][x := N[y := L]].\end{aligned}$$

- **Case 3:** $M \equiv M_1M_2$. The statement follows again from the induction hypothesis. □

**Substitution Lemma:** If $x \not\equiv y$ and $x \notin \mathsf{fv}(L)$, then
$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$$

**Proof:** By induction on the structure of $M$.

- **Case 1:** $M$ is a variable.

  Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

  Case 1.2. $M \equiv y$. Then both sides equal $L$, for $x \notin \mathsf{fv}(L)$ implies $L[x := \ldots] \equiv L$.

  Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal $z$.

- **Case 2:** $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and $z$ is not free in $N, L$.

$$\begin{aligned}(\lambda z.M_1)[x := N][y := L] &\equiv \lambda z.(M_1[x := N][y := L])\\ &\equiv \lambda z.(M_1[y := L][x := N[y := L]])\\ &\equiv (\lambda z.M_1)[y := L][x := N[y := L]].\end{aligned}$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis. □

**Substitution Lemma:** If $x \not\equiv y$ and $x \notin \mathsf{fv}(L)$, then
$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$$

**Proof:** By induction on the structure of $M$.

- **Case 1:** $M$

  Case 1.1. $M$

  $x$

  Case 1.2. $M$

  in

  Case 1.3. $M$

- **Case 2:** $M$

  assume th

  $(\lambda z.M_1)[x$

Remember only if $y \neq x$ and $x \notin \mathsf{fv}(N)$ then

$$(\lambda y.M)[x := N] = \lambda y.(M[x := N])$$

$$
\begin{array}{lll}
& (\lambda z.M_1)[x := N][y := L] & \\
\equiv & (\lambda z.(M_1[x := N]))[y := L] & \xleftarrow{1} \\
\equiv & \lambda z.(M_1[x := N][y := L]) & \xleftarrow{2} \\
\equiv & \lambda z.(M_1[y := L][x := N[y := L]]) & \text{IH} \\
\equiv & (\lambda z.(M_1[y := L]))[x := N[y := L]]) & \xrightarrow{2} \; ! \\
\equiv & (\lambda z.M_1)[y := L][x := N[y := L]]. & \xrightarrow{1}
\end{array}
$$

- **Case 3:** $M \equiv M_1 M_2$. The statement follows again from the induction hypothesis. □

# Nominal Datatypes

- Define lambda-terms as:

```
atom_decl name

nominal_datatype lam =
    Var "name"
  | App "lam" "lam"
  | Lam "«name»lam" ("Lam [_]._")
```

- These are <u>**named**</u> alpha-equivalence classes, for example

$$\text{Lam [a].(Var a)} = \text{Lam [b].(Var b)}$$

```
lemma forget:
  assumes a: "x # L"
  shows "L[x::=P] = L"
using a by (nominal_induct L avoiding: x P rule: lam.strong_induct)
           (auto simp add: abs_fresh fresh_atm)

lemma fresh_fact:
  fixes z::"name"
  assumes a: "z # N" "z # L"
  shows "z # N[y::=L]"
using a by (nominal_induct N avoiding: z y L rule: lam.strong_induct)
           (auto simp add: abs_fresh fresh_atm)

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

```
lemma forget:
  assumes a: "x # L"
  shows "L[x::=P] = L"
using a by (nominal_induct L avoiding: x P rule: lam.strong_induct)
           (auto simp add: abs_fresh fresh_atm)

lemma fresh_fact:
  fixes z::"name"
  assumes a: "z # N" "z # L"
  shows "z # N[y::=L]"
using a by (nominal_induct N a                                duct)
           (auto simp add: abs_
```

- stands for $x \notin \mathrm{fv}(L)$
- reads as “$x$ fresh for $L$”

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

```
lemma forget:
  assumes a: "x # L"
  shows "L[x::=P] = L"
using a by (nominal_induct L avoiding: x P rule: lam.strong_induct)
           (auto simp add: abs_fresh fresh_atm)

lemma fresh_fact:
  fixes z::"name"
  assumes a: "z # N" "z # L"
  shows "z # N[y::=L]"
using a by (nominal_induct N avoiding: z y L rule: lam.strong_induct)
           (auto simp add: abs_fresh fresh_atm)

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

# (Weak) Induction Principles

- The usual induction principle is as follows:

$$\frac{\begin{array}{l}\forall x.\ P\,x \\ \forall t_1\,t_2.\ P\,t_1 \wedge P\,t_2 \Rightarrow P\,(t_1\ t_2) \\ \forall x\,t.\ P\,t \Rightarrow P\,(\lambda x.t)\end{array}}{P\,t}$$

- It requires us in the lambda-case to show the property $P$ for all binders $x$.
  (This nearly always requires renamings and they can be tricky to automate.)

# Strong Induction Principles

- Therefore we will use the following strong induction principle:

$$\frac{\begin{array}{l}\forall x\, c.\; P\, c\, x \\ \forall t_1\, t_2\, c.\; (\forall d.\, P\, d\, t_1) \wedge (\forall d. P\, d\, t_2) \Rightarrow P\, c\, (t_1\, t_2) \\ \forall x\, t\, c.\; x \,\#\, c \wedge (\forall d. P\, d\, t) \Rightarrow P\, c\, (\lambda x. t)\end{array}}{P\;\, c\;\, t}$$

# Strong Induction Principles

- Therefore we will use the following strong induction principle:

$$\frac{\begin{array}{l} \forall x\,c.\; P\,c\,x \\ \forall t_1\,t_2\,c.\; (\forall d.\, P\,d\,t_1) \wedge (\forall d. P\,d\,t_2) \Rightarrow P\,c\,(t_1\,t_2) \\ \forall x\,t\,c.\; x\,\#\,c \wedge (\forall d. P\,d\,t) \Rightarrow P\,c\,(\lambda x.t) \end{array}}{P\;\;c\;\;t}$$

The variable over which the induction proceeds:

"…By induction over the structure of $M$…"

# Strong Induction Principles

- Therefore we will use the following strong induction principle:

$$\frac{\begin{array}{l} \forall x\,c.\; P\,c\,x \\ \forall t_1\,t_2\,c.\;(\forall d.\,P\,d\,t_1) \wedge (\forall d.P\,d\,t_2) \Rightarrow P\,c\,(t_1\,t_2) \\ \forall x\,t\,c.\; x \,\#\, c \wedge (\forall d.P\,d\,t) \Rightarrow P\,c\,(\lambda x.t) \end{array}}{P\;c\;t}$$

The **context** of the induction; i.e. what the binder should be fresh for $\Rightarrow$ $(x, y, N, L)$:

"...By the variable convention we can assume $z \not\equiv x, y$ and $z$ not free in $N, L$..."

# Strong Induction Principles

- Therefore we will use the following strong induction principle:

$$\frac{\begin{array}{l}\forall x\,c.\ P\,c\,x\\ \forall t_1\,t_2\,c.\ (\forall d.\ P\,d\,t_1) \wedge (\forall d.P\,d\,t_2) \Rightarrow P\,c\,(t_1\,t_2)\\ \forall x\,t\,c.\ x \,\#\, c \wedge (\forall d.P\,d\,t) \Rightarrow P\,c\,(\lambda x.t)\end{array}}{P\;\;c\;\;t}$$

The property to be proved by induction:

$$\begin{array}{l}\lambda(x,y,N,L).\,\lambda M.\ \ x \neq y\ \wedge\ x \,\#\, L\ \Rightarrow\\ \qquad M[x:=N][y:=L] = M[y:=L][x:=N[y:=L]]\end{array}$$

```
proof (nominal_induct M avoiding: x y N L rule: lam.strong_induct)
 case (Var z)
 show "Var z[x::=N][y::=L] = Var z[y::=L][x::=N[y::=L]]"  (is "?LHS = ?RHS")
 proof -
  { assume  "z=x"
   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"
   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"
   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 . . .
```

```
proof (nominal_induct M avoiding: x y N L rule: lam.strong_induct)
 case (Var z)
 show "Var z[x::=N][y::=L] = Var z[y::=L][x::=N[y::=L]]"   (is "?LHS = ?RHS")
 proof -
  { assume  "z=x"
    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"
    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"
    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 . . .
```

```
proof (nominal_induct M avoiding: x y N L rule: lam.strong_induct)
 case (Var z)
 show "Var z[x::=N][y::=L] = Var z[y::=L][x::=N[y::=L]]"   (is "?LHS = ?RHS")
 proof -
  { assume  "z=x"
   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"
   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"
   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 . . .
```

```
proof (nominal_induct M avoiding: x y N L rule: lam.strong_induct)
 case (Var z)
 show "Var z[x::=N][y::=L] = Var z[y::=L][x::=N[y::=L]]"   (is "?LHS = ?RHS")
 proof -
  { assume  "z=x"
   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"
   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"
   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 . . .
```

```
proof (nominal_induct M avoiding: x y N L rule: lam.strong_induct)
 case (Var z)
 show "Var z[x::=N][y::=L] = Var z[y::=L][x::=N[y::=L]]"   (is "?LHS = ?RHS")
 proof -
  { assume  "z=x"
    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"
    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"
    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 . . .
```

```
proof (nominal_induct M avoiding: x y N L rule: lam.strong_induct)
 case (Var z)
 show "Var z[x::=N][y::=L] = Var z[y::=L][x::=N[y::=L]]"   (is "?LHS = ?RHS")
 proof -
  { assume  "z=x"
    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"
    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"
    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 . . .
```

```
proof (nominal_induct M avoiding: x y N L rule: lam.strong_induct)
 case (Var z)
 show "Var z[x::=N][y::=L] = Var z[y::=L][x::=N[y::=L]]"   (is "?LHS = ?RHS")
 proof -
  { assume  "z=x"
    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"
    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"
    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 . . .
```

```
proof (nominal_induct M avoiding: x y N L rule: lam.strong_induct)
 case (Var z)
 show "Var z[x::=N][y::=L] = Var z[y::=L][x::=N[y::=L]]"  (is "?LHS = ?RHS")
 proof -
  { assume  "z=x"
    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"
    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"
    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 . . .

```
proof (nominal_induct M avoiding: x y N L rule: lam.strong_induct)
 case (Var z)
 show "Var z[x::=N][y::=L] = Var z[y::=L][x::=N[y::=L]]"   (is "?LHS = ?RHS")
 proof -
  { assume  "z=x"
    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"
    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"
    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 . . .
```

```
next
 case (Lam z $M_1$)
 have ih: "⟦x≠y; x#L⟧ ⟹ $M_1$[x::=N][y::=L] = $M_1$[y::=L][x::=N[y::=L]]" by fact
 have "x≠y" by fact
 have "x#L" by fact
 have vc: "z#x" "z#y" "z#N" "z#L" by fact+
 then have "z#N[y::=L]" by (simp add: fresh_fact)
 show "(Lam [z].$M_1$)[x::=N][y::=L]=(Lam [z].$M_1$)[y::=L][x::=N[y::=L]]" (is "?LHS=?RHS")
 proof -
  have "?LHS = Lam [z].($M_1$[x::=N][y::=L])" using vc by simp
  also from ih have "... = Lam [z].($M_1$[y::=L][x::=N[y::=L]])" using ‘x≠y‘ ‘x#L‘ by simp
  also have "... = (Lam [z].($M_1$[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 $M_1$ $M_2$)
 then show "(App $M_1$ $M_2$)[x::=N][y::=L] = (App $M_1$ $M_2$)[y::=L][x::=N[y::=L]]" by simp
qed
```

```
next
 case (Lam z M1)
 have ih: "⟦x≠y; x#L⟧ ⟹ M1[x::=N][y::=L] = M1[y::=L][x::=N[y::=L]]" by fact
 have "x≠y" by fact
 have "x#L" by fact
 have vc: "z#x" "z#y" "z#N" "z#L" by fact+
 then have "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 vc 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)
 then show "(App M1 M2)[x::=N][y::=L] = (App M1 M2)[y::=L][x::=N[y::=L]]" by simp
qed
```

```
next
 case (Lam z M₁)
 have ih: "⟦x≠y; x#L⟧ ⟹ M₁[x::=N][y::=L] = M₁[y::=L][x::=N[y::=L]]" by fact
 have "x≠y" by fact
 have "x#L" by fact
 have vc: "z#x" "z#y" "z#N" "z#L" by fact+
 then have "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 vc 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)
 then show "(App M₁ M₂)[x::=N][y::=L] = (App M₁ M₂)[y::=L][x::=N[y::=L]]" by simp
qed
```

```
next
 case (Lam z M1)
 have ih: "⟦x≠y; x#L⟧ ⟹ M1[x::=N][y::=L] = M1[y::=L][x::=N[y::=L]]" by fact
 have "x≠y" by fact
 have "x#L" by fact
 have vc: "z#x" "z#y" "z#N" "z#L" by fact+
 then have "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 vc 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)
 then show "(App M1 M2)[x::=N][y::=L] = (App M1 M2)[y::=L][x::=N[y::=L]]" by simp
qed
```

```
next
 case (Lam z M1)
 have ih: "⟦x≠y; x#L⟧ ⟹ M1[x::=N][y::=L] = M1[y::=L][x::=N[y::=L]]" by fact
 have "x≠y" by fact
 have "x#L" by fact
 have vc: "z#x" "z#y" "z#N" "z#L" by fact+
 then have "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 vc 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)
 then show "(App M1 M2)[x::=N][y::=L] = (App M1 M2)[y::=L][x::=N[y::=L]]" by simp
qed
```

```
next
 case (Lam z M_1)
 have ih: "⟦x≠y; x#L⟧ ⟹ M_1[x::=N][y::=L] = M_1[y::=L][x::=N[y::=L]]" by fact
 have "x≠y" by fact
 have "x#L" by fact
 have vc: "z#x" "z#y" "z#N" "z#L" by fact+
 then have "z#N[y::=L]" by (simp add: fresh_fact)
 show "(Lam [z].M_1)[x::=N][y::=L]=(Lam [z].M_1)[y::=L][x::=N[y::=L]]" (is "?LHS=?RHS")
 proof -
  have "?LHS = Lam [z].(M_1[x::=N][y::=L])" using vc by simp
  also from ih have "... = Lam [z].(M_1[y::=L][x::=N[y::=L]])" using ‘x≠y‘ ‘x#L‘ by simp
  also have "... = (Lam [z].(M_1[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 M_1 M_2)
 then show "(App M_1 M_2)[x::=N][y::=L] = (App M_1 M_2)[y::=L][x::=N[y::=L]]" by simp
qed
```

```
next
 case (Lam z M1)
 have ih: "⟦x≠y; x#L⟧ ⟹ M1[x::=N][y::=L] = M1[y::=L][x::=N[y::=L]]" by fact
 have "x≠y" by fact
 have "x#L" by fact
 have vc: "z#x" "z#y" "z#N" "z#L" by fact+
 then have "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 vc 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)
 then show "(App M1 M2)[x::=N][y::=L] = (App M1 M2)[y::=L][x::=N[y::=L]]" by simp
qed
```

```
next
 case (Lam z M1)
 have ih: "⟦x≠y; x#L⟧ ⟹ M1[x::=N][y::=L] = M1[y::=L][x::=N[y::=L]]" by fact
 have "x≠y" by fact
 have "x#L" by fact
 have vc: "z#x" "z#y" "z#N" "z#L" by fact+
 then have "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 vc 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)
 then show "(App M1 M2)[x::=N][y::=L] = (App M1 M2)[y::=L][x::=N[y::=L]]" by simp
qed
```

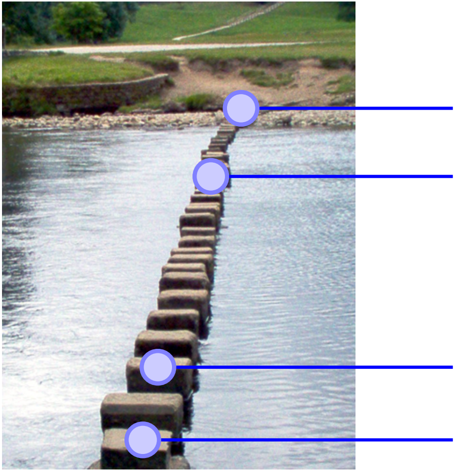
```
next
 case (Lam z M1)
 have ih: "⟦x≠y; x#L⟧ ⟹ M1[x::=N][y::=L] = M1[y::=L][x::=N[y::=L]]" by fact
 have "x≠y" by fact
 have "x#L" by fact
 have vc: "z#x" "z#y" "z#N" "z#L" by fact+
 then have "z#N[y::=L]" by (simp add: fresh_fact)
 show "(Lam [z].M1)[x::=N][y::=L]=(Lam [z].M1)[y::=L][x::=N[y::=L]]" (is "?LHS=?RHS")
 proof -
   have "?LHS = Lam [z].(M1[x::=N][y::=L])" using vc 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)
 then show "(App M1 M2)[x::=N][y::=L] = (App M1 M2)[y::=L][x::=N[y::=L]]" by simp
qed
```

```
next
 case (Lam z M_1)
 have ih: "⟦x≠y; x#L⟧ ⟹ M_1[x::=N][y::=L] = M_1[y::=L][x::=N[y::=L]]" by fact
 have "x≠y" by fact
 have "x#L" by fact
 have vc: "z#x" "z#y" "z#N" "z#L" by fact+
 then have "z#N[y::=L]" by (simp add: fresh_fact)
 show "(Lam [z].M_1)[x::=N][y::=L]=(Lam [z].M_1)[y::=L][x::=N[y::=L]]" (is "?LHS=?RHS")
 proof -
  have "?LHS = Lam [z].(M_1[x::=N][y::=L])" using vc by simp
  also from ih have "... = Lam [z].(M_1[y::=L][x::=N[y::=L]])" using ‘x≠y‘ ‘x#L‘ by simp
  also have "... = (Lam [z].(M_1[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 M_1 M_2)
 then show "(App M_1 M_2)[x::=N][y::=L] = (App M_1 M_2)[y::=L][x::=N[y::=L]]" by simp
qed
```

# An Isar Proof . . .

- The Isar proof language has been conceived by Markus Wenzel, the main developer behind Isabelle.

# An Isar Proof . . .



goal

stepping stones

⋮

stepping stones

assumptions

- The Isar proof language has been conceived by Markus Wenzel, the main developer behind Isabelle.

# Strong Induction Principles

$$\frac{\begin{array}{l}\forall x\, c.\ P\, c\, x\\ \forall t_1\, t_2\, c.\ (\forall d.\, P\, d\, t_1) \wedge (\forall d. P\, d\, t_2) \Rightarrow P\, c\, (t_1\, t_2)\\ \forall x\, t\, c.\ x \,\#\, c \wedge (\forall d. P\, d\, t) \Rightarrow P\, c\, (\lambda x. t)\end{array}}{P\, c\, t}$$

- There is a condition for when Barendregt's variable convention is applicable—it is almost always satisfied, but not always:

  The induction context $c$ needs to be finitely supported (is not allowed to mention all names as free).

# Strong Induction Principles

$$\frac{\begin{array}{l}\forall x\,c.\; P\,c\,x\\ \forall t_1\,t_2\,c.\; (\forall d.\, P\,d\,t_1) \wedge (\forall d.P\,d\,t_2) \Rightarrow P\,c\,(t_1\,t_2)\\ \forall x\,t\,c.\; x \,\#\, c \wedge (\forall d.P\,d\,t) \Rightarrow P\,c\,(\lambda x.t)\end{array}}{P\,c\,t}$$

- In the case of the substitution lemma:

```
proof (nominal_induct M avoiding: x y N L rule: lam.strong_induct)
. . .
```

# Same Problem with Rule Inductions

- We can specify typing-rules for lambda-terms as:

$$\frac{(x\!:\!\tau) \in \Gamma \quad \text{valid } \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash t_1 : \sigma \rightarrow \tau \quad \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1\; t_2 : \tau}$$

$$\frac{x \# \Gamma \quad (x\!:\!\sigma)\!::\!\Gamma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \rightarrow \tau}$$

$$\frac{}{\text{valid } []} \qquad \frac{x \# \Gamma \quad \text{valid } \Gamma}{\text{valid } (x\!:\!\tau)\!::\!\Gamma}$$

- If $\Gamma_1 \vdash t : \tau$ and valid $\Gamma_2$, $\Gamma_1 \subseteq \Gamma_2$ then $\Gamma_2 \vdash t : \tau$.

# Same Problem with Rule Inductions

- We can specify typing-rules for lambda-terms as:

$\overline{(}$ $\overline{\sigma}$

The proof of the weakening lemma is said to be trivial / obvious / routine /... in many places.

(I am actually still looking for a place in the literature where a trivial / obvious / routine /... proof is spelled out — I know of proofs by Gallier, McKinna & Pollack and Pitts, but I would not call them trivial / obvious / routine /...)

valid [] valid $(x : \tau) :: \Gamma$

- If $\Gamma_1 \vdash t : \tau$ and valid $\Gamma_2$, $\Gamma_1 \subseteq \Gamma_2$ then $\Gamma_2 \vdash t : \tau$.

# Recall: Rule Inductions

$$\frac{\mathsf{prem}_1 \ldots \mathsf{prem}_n \;\; \mathsf{scs}}{\mathsf{concl}}\,\mathsf{rule}$$

Rule Inductions:

1.) Assume the property for the premises.
Assume the side-conditions.

2.) Show the property for the conclusion.

# Induction Principle for Typing

- The induction principle that comes with the typing definition is as follows:

$$\forall \Gamma\, x\, \tau.\ (x{:}\tau) \in \Gamma \wedge \mathsf{valid}\, \Gamma \Rightarrow P\, \Gamma\, (x)\, \tau$$

$$\begin{array}{l} \forall \Gamma\, t_1\, t_2\, \sigma\, \tau. \\ P\, \Gamma\, t_1\, (\sigma \to \tau) \wedge P\, \Gamma\, t_2\, \sigma \Rightarrow P\, \Gamma\, (t_1\, t_2)\, \tau \end{array}$$

$$\begin{array}{l} \forall \Gamma\, x\, t\, \sigma\, \tau. \\ x \,\#\, \Gamma \wedge P\, ((x{:}\sigma){::}\Gamma)\, t\, \tau \Rightarrow P\, \Gamma (\lambda x.t)\, (\sigma \to \tau) \end{array}$$

$$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$$

$$\Gamma \vdash t : \tau \Rightarrow P\, \Gamma\, t\, \tau$$

Note the quantifiers!

# Proof of Weakening Lemma

$$\frac{x \# \Gamma \quad (x:\sigma)::\Gamma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \to \tau}$$

- If $\Gamma_1 \vdash t : \tau$ then $\forall \Gamma_2$. valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t : \tau$

# Proof of Weakening Lemma

$$\frac{x \# \Gamma \quad (x{:}\sigma)::\Gamma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \to \tau}$$

- If $\Gamma_1 \vdash t : \tau$ then $\forall \Gamma_2.\, \text{valid}\, \Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t : \tau$

For all $\Gamma_1$, $x$, $t$, $\sigma$ and $\tau$:

- We know:
  $\forall \Gamma_2.\, \text{valid}\, \Gamma_2 \wedge (x{:}\sigma)::\Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t : \tau$
  $x \# \Gamma_1$

- We have to show:
  $\forall \Gamma_2.\, \text{valid}\, \Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash \lambda x.t : \sigma \to \tau$

# Proof of Weakening Lemma

$$\frac{x \;\#\; \Gamma \qquad (x{:}\sigma)::\Gamma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \to \tau}$$

- If $\Gamma_1 \vdash t : \tau$ then $\forall \Gamma_2.$ valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t : \tau$

For all $\Gamma_1$, $x$, $t$, $\sigma$ and $\tau$:

- We know:
  $\forall \Gamma_2.$ valid $\Gamma_2 \wedge (x{:}\sigma)::\Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t : \tau$
  $x \;\#\; \Gamma_1$
  valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2$

- We have to show:
  $\Gamma_2 \vdash \lambda x.t : \sigma \to \tau$

# Proof of Weakening Lemma

$$\frac{x \# \Gamma \quad (x{:}\sigma)::\Gamma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \to \tau}$$

- If $\Gamma_1 \vdash t:\tau$ then $\forall \Gamma_2.$ valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t:\tau$

For all $\Gamma_1$, $x$, $t$, $\sigma$ and $\tau$:

- We know:
  $\forall \Gamma_2.$ valid $\Gamma_2 \wedge (x{:}\sigma)::\Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t:\tau$
  $x \# \Gamma_1$
  valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2$

- We have to show:
  $\Gamma_2 \vdash \lambda x.t:\sigma \to \tau$

# Proof of Weakening Lemma

$$\frac{x \# \Gamma \qquad (x:\sigma)::\Gamma \vdash t:\tau}{\Gamma \vdash \lambda x.t : \sigma \rightarrow \tau}$$

- If $\Gamma_1 \vdash t:\tau$ then $\forall \Gamma_2.$ valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t:\tau$

For all $\Gamma_1, x, t, \sigma$ and $\tau$:

$\boxed{\Gamma_2 \mapsto (x:\sigma)::\Gamma_2}$

- We know:
  $\forall \Gamma_2.$ valid $\Gamma_2 \wedge (x:\sigma)::\Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t:\tau$
  $x \# \Gamma_1$
  valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2$

- We have to show:
  $\Gamma_2 \vdash \lambda x.t : \sigma \rightarrow \tau$

# Proof of Weakening Lemma

$$\frac{x \# \Gamma \quad (x:\sigma)::\Gamma \vdash t:\tau}{\Gamma \vdash \lambda x.t : \sigma \to \tau}$$

- If $\Gamma_1 \vdash t:\tau$ then $\forall \Gamma_2.$ valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t:\tau$

For all $\Gamma_1$, $x$, $t$, $\sigma$ and $\tau$:

$\boxed{\Gamma_2 \mapsto (x:\sigma)::\Gamma_2}$

- We know:

  $\forall \Gamma_2.$ valid $\Gamma_2 \wedge (x:\sigma)::\Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t:\tau$

  $x \# \Gamma_1$

  valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow (x:\sigma)::\Gamma_1 \subseteq (x:\sigma)::\Gamma_2$

- We have to show:

  $\Gamma_2 \vdash \lambda x.t : \sigma \to \tau$

# Proof of Weakening Lemma

$$\frac{x \# \Gamma \quad (x:\sigma)::\Gamma \vdash t:\tau}{\Gamma \vdash \lambda x.t : \sigma \to \tau}$$

- If $\Gamma_1 \vdash t:\tau$ then $\forall \Gamma_2.\,\text{valid}\ \Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t:\tau$

<u>For all $\Gamma_1, x, t, \sigma$ and $\tau$:</u>

$\Gamma_2 \mapsto (x:\sigma)::\Gamma_2$

- We know:

  $\forall \Gamma_2.\,\text{valid}\ \Gamma_2 \wedge (x:\sigma)::\Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t:\tau$

  $x \# \Gamma_1$

  $\text{valid}\ \Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow (x:\sigma)::\Gamma_1 \subseteq (x:\sigma)::\Gamma_2$

  $\text{valid}\ (x:\sigma)::\Gamma_2$ ???

- We have to show:

  $\Gamma_2 \vdash \lambda x.t : \sigma \to \tau$

- The usual proof of strong normalisation for simply-typed lambda-terms establishes first:

Lemma: If for all reducible $s$, $t[x := s]$ is reducible, then $\lambda x.t$ is reducible.

- Then one shows for a closing (simultaneous) substitution:

Theorem: If $\Gamma \vdash t : \tau$, then for all closing substitutions $\theta$ containing reducible terms only, $\theta(t)$ is reducible.

Lambda-Case: By ind. we know $(x \mapsto s \cup \theta)(t)$ is reducible with $s$ being reducible. This is equal* to $(\theta(t))[x := s]$. Therefore, we can apply the lemma and get $\lambda x.(\theta(t))$ is reducible. Because this is equal* to $\theta(\lambda x.t)$, we are done.

*you have to take a deep breath

# Strong Induction Principle

$$\frac{\begin{array}{l}\forall \Gamma\, x\, \tau\, .\ (x\!:\!\tau) \in \Gamma \wedge \mathsf{valid}\,\Gamma \Rightarrow P\ \Gamma\ (x)\ \tau \\[1em] \forall \Gamma\, t_1\, t_2\, \sigma\, \tau\, . \\ P\,\Gamma\,t_1\,(\sigma \to \tau) \wedge P\,\Gamma\,t_2\,\sigma \\ \qquad\qquad\qquad\qquad \Rightarrow P\,\Gamma\,(t_1\,t_2)\,\tau \\[1em] \forall \Gamma\, x\, t\, \sigma\, \tau\, . \\ x \# \Gamma\ \wedge \\ P\,((x\!:\!\sigma)::\Gamma)\,t\,\tau \Rightarrow P\,\Gamma(\lambda x.t)\,(\sigma \to \tau)\end{array}}{\Gamma \vdash t : \tau \Rightarrow P\,\Gamma\,t\,\tau}$$

- Instead we are going to use the strong induction principle and set up the induction so that it “avoids” $\Gamma_2$ (in case of the weakening lemma) and $\theta$ (in case of SN).

# Proof of Weakening Lemma

$$\frac{x \# \Gamma \quad (x\!:\!\sigma)::\Gamma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \to \tau}$$

- If $\Gamma_1 \vdash t : \tau$ then valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t : \tau$

For all $\Gamma_1$, $x$, $t$, $\sigma$ and $\tau$:

- We know:
  $\forall \Gamma_2.$ valid $\Gamma_2 \wedge (x\!:\!\sigma)::\Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t : \tau$
  $x \# \Gamma_1$
  valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2$
  $x \# \Gamma_2$
- We have to show:
  $\Gamma_2 \vdash \lambda x.t : \sigma \to \tau$

# Proof of Weakening Lemma

$$\frac{x \# \Gamma \quad (x{:}\sigma)::\Gamma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \to \tau}$$

- If $\Gamma_1 \vdash t : \tau$ then valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t : \tau$

For all $\Gamma_1$, $x$, $t$, $\sigma$ and $\tau$:

- We know:
  $\forall \Gamma_2.$ valid $\Gamma_2 \wedge (x{:}\sigma)::\Gamma_1 \subseteq \Gamma_2 \Rightarrow \Gamma_2 \vdash t : \tau$
  $x \# \Gamma_1$
  valid $\Gamma_2 \wedge \Gamma_1 \subseteq \Gamma_2 \quad \Rightarrow (x{:}\sigma)::\Gamma_1 \subseteq (x{:}\sigma)::\Gamma_2$
  $x \# \Gamma_2 \quad \Rightarrow$ valid $(x{:}\sigma)::\Gamma_2$
- We have to show:
  $\Gamma_2 \vdash \lambda x.t : \sigma \to \tau$

# In Nominal Isabelle

**abbreviation**
  "sub_ctx" :: "(name $\times$ ty) list $\Rightarrow$ (name $\times$ ty) list $\Rightarrow$ bool" ("_ $\subseteq$ _")
**where**
  "$\Gamma_1 \subseteq \Gamma_2 \equiv \forall$ x T. (x,T) $\in$ set $\Gamma_1 \longrightarrow$ (x,T) $\in$ set $\Gamma_2$"

**lemma** weakening_lemma:
  **fixes** $\Gamma_1$ $\Gamma_2$::"(name $\times$ ty) list"
  **assumes** a: "$\Gamma_1 \vdash$ t : T"
  **and** b: "valid $\Gamma_2$"
  **and** c: "$\Gamma_1 \subseteq \Gamma_2$"
  **shows** "$\Gamma_2 \vdash$ t : T"
**using** a b c
**by** (nominal_induct $\Gamma_1$ t T avoiding: $\Gamma_2$ rule: typing.strong_induct)
    (auto simp add: atomize_all atomize_imp)

# SN (Again)

Theorem: If $\Gamma \vdash t : \tau$, then for all closing substitutions $\theta$ containing reducible terms only, $\theta(t)$ is reducible.

- Since we say that the strong induction should avoid $\theta$, we get the assumption $x \# \theta$ then:

  Lambda-Case: By ind. we know $(x \mapsto s \cup \theta)(t)$ is reducible with $s$ being reducible. This is **equal** to $(\theta(t))[x := s]$. Therefore, we can apply the lemma and get $\lambda x.(\theta(t))$ is reducible. Because this is **equal** to $\theta(\lambda x.t)$, we are done.

$$x \# \theta \Rightarrow \quad (x \mapsto s \cup \theta)(t) \; = \; (\theta(t))[x := s]$$
$$\theta(\lambda x.t) \; = \; \lambda x.(\theta(t))$$

# So Far So Good

- A Faulty Lemma with the Variable Convention?

> **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.
>
> Barendregt in "The Lambda-Calculus: Its Syntax and Semantics"

Inductive Definitions:

$$\frac{\text{prem}_1 \ldots \text{prem}_n \;\; \text{scs}}{\text{concl}}$$

Rule Inductions:

1.) Assume the property for the premises. Assume the side-conditions.

2.) Show the property for the conclusion.

# Faulty Reasoning

- Consider the two-place relation foo:

$$\frac{}{x \mapsto x} \qquad \frac{}{t_1\ t_2 \mapsto t_1\ t_2} \qquad \frac{t \mapsto t'}{\lambda x.t \mapsto t'}$$

# Faulty Reasoning

- Consider the two-place relation foo:

$$\frac{}{x \mapsto x} \qquad \frac{}{t_1\ t_2 \mapsto t_1\ t_2} \qquad \frac{t \mapsto t'}{\lambda x.t \mapsto t'}$$

- The lemma we going to prove:

  Let $t \mapsto t'$. If $y \# t$ then $y \# t'$.

# Faulty Reasoning

- Consider the two-place relation foo:

$$\frac{}{x \mapsto x} \qquad \frac{}{t_1\ t_2 \mapsto t_1\ t_2} \qquad \frac{t \mapsto t'}{\lambda x.t \mapsto t'}$$

- The lemma we going to prove:

  Let $t \mapsto t'$. If $y \# t$ then $y \# t'$.

- Cases 1 and 2 are trivial:
  - If $y \# x$ then $y \# x$.
  - If $y \# t_1\, t_2$ then $y \# t_1\, t_2$.

# Faulty Reasoning

- Consider the two-place relation foo:

$$\frac{}{x \mapsto x} \qquad \frac{}{t_1\ t_2 \mapsto t_1\ t_2} \qquad \frac{t \mapsto t'}{\lambda x.t \mapsto t'}$$

- The lemma we going to prove:

  Let $t \mapsto t'$. If $y \# t$ then $y \# t'$.

- Case 3:
  - We know $y \# \lambda x.t$. We have to show $y \# t'$.
  - The IH says: if $y \# t$ then $y \# t'$.

**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.

**In our case:**

The free variables are $y$ and $t'$; the bound one is $x$.

By the variable convention we conclude that $x \neq y$.

Let $t \mapsto t'$. If $y \;\#\; t$ then $y \;\#\; t'$.

- Case 3:
  - We know $y \;\#\; \lambda x.t$. We have to show $y \;\#\; t'$.
  - The IH says: if $y \;\#\; t$ then $y \;\#\; t'$.

**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.

**In our case:**

The free variables are $y$ and $t'$; the bound one is $x$.

By the variable convention we conclude that $x \neq y$.

Let $t \mapsto t'$. If $y \# t$ then $y \# t'$

$$y \notin \mathsf{fv}(\lambda x.t) \iff y \notin \mathsf{fv}(t) - \{x\} \overset{x \neq y}{\iff} y \notin \mathsf{fv}(t)$$

- Case 3:
  - We know $y \# \lambda x.t$. We have to show $y \# t'$.
  - The IH says: if $y \# t$ then $y \# t'$.

**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.

**In our case:**

The free variables are $y$ and $t'$; the bound one is $x$.

By the variable convention we conclude that $x \neq y$.

Let $t \longrightarrow t'$. If $y \# t$ then $y \# t'$.

$$y \notin \mathsf{fv}(\lambda x.t) \iff y \notin \mathsf{fv}(t) - \{x\} \overset{x \neq y}{\iff} y \notin \mathsf{fv}(t)$$

- Case 3:
  - We know $y \# \lambda x.t$. We have to show $y \# t'$.
  - The IH says: if $y \# t$ then $y \# t'$.
  - So we have $y \# t$. Hence $y \# t'$ by IH. Done!

# Faulty Reasoning

- Consider the two-place relation foo:

$$\frac{}{x \mapsto x} \qquad \frac{}{t_1\ t_2 \mapsto t_1\ t_2} \qquad \frac{t \mapsto t'}{\lambda x.t \mapsto t'}$$

- The lemma we going to prove:

  Let $t \mapsto t'$. If $y \# t$ then $y \# t'$.

- Case 3:
  - We know $y \# \lambda x.t$. We have to show $y \# t'$.
  - The IH says: if $y \# t$ then $y \# t'$.
  - So we have $y \# t$. Hence $y \# t'$ by IH. Done!

# VC-Compatibility

- We introduced two conditions that make the VC safe to use in rule inductions:
  - the relation needs to be **equivariant**, and
  - the binder is not allowed to occur in the **support** of the conclusion (not free in the conclusion)

- Once a relation satisfies these two conditions, then Nominal Isabelle derives the strong induction principle automatically.

# VC-Compatibility

- We introduced two conditions that make the VC safe to use in rule inductions:
  - the relation needs to be **equivariant**, and

A relation $R$ is **equivariant** iff

$$\begin{array}{l} \forall \pi\, t_1 \ldots t_n \\ \qquad R\, t_1 \ldots t_n \Rightarrow R (\pi \bullet t_1) \ldots (\pi \bullet t_n) \end{array}$$

This means the relation has to be invariant under permutative renaming of variables.

(This property can be checked automatically if the inductive definition is composed of equivariant "things".)

# VC-Compatibility

- We introduced two conditions that make the VC safe to use in rule inductions:
  - the relation needs to be **equivariant**, and
  - the binder is not allowed to occur in the **support** of the conclusion (not free in the conclusion)

- Once a relation satisfies these two conditions, then Nominal Isabelle derives the strong induction principle automatically.

# Honest Toil, No Theft!

- The sacred principle of HOL:

> “The method of ‘postulating’ what we want has many advantages; they are the same as the advantages of theft over honest toil.”
>
> B. Russell, Introduction of Mathematical Philosophy

- I will show next that the weak structural induction principle implies the strong structural induction principle.

  (I am only going to show the lambda-case.)

# Permutations

A permutation **acts** on variable names as follows:

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

- $[]$ stands for the empty list (the identity permutation), and
- $(a_1\,a_2)::\pi$ stands for the permutation $\pi$ followed by the swapping $(a_1\,a_2)$.

# Permutations on Lambda-Terms

- Permutations act on lambda-terms as follows:

$$\begin{array}{rcl} \pi \bullet x & \overset{\text{def}}{=} & \text{“action on variables”} \\ \pi \bullet (t_1\ t_2) & \overset{\text{def}}{=} & (\pi \bullet t_1)\ (\pi \bullet t_2) \\ \pi \bullet (\lambda x.t) & \overset{\text{def}}{=} & \lambda(\pi \bullet x).(\pi \bullet t) \end{array}$$

- Alpha-equivalence can be defined as:

$$\frac{t_1 = t_2}{\lambda x.t_1 = \lambda x.t_2}$$

$$\frac{x \neq y \quad t_1 = (x\ y) \bullet t_2 \quad x \ \#\ t_2}{\lambda x.t_1 = \lambda y.t_2}$$

# Permutations on Lambda-Terms

- Permutations act on lambda-terms as follows:

$$\begin{array}{rcl} \pi \bullet x & \overset{\text{def}}{=} & \text{“action on variables”} \\ \pi \bullet (t_1\ t_2) & \overset{\text{def}}{=} & (\pi \bullet t_1)\ (\pi \bullet t_2) \\ \pi \bullet (\lambda x.t) & \overset{\text{def}}{=} & \lambda(\pi \bullet x).(\pi \bullet t) \end{array}$$

- Alpha-equivalence can be defined as:

$$\frac{t_1 = t_2}{\lambda x.t_1 = \lambda x.t_2}$$

$$\frac{x \neq y \quad t_1 = (x\ y) \bullet t_2 \quad x \# t_2}{\lambda x.t_1 = \lambda y.t_2}$$

Notice, I wrote equality here!

# My Claim

$$\frac{\begin{array}{l}\forall x.\ P\ x\\ \forall t_1\,t_2.\ P\ t_1 \wedge P\ t_2 \Rightarrow P\ (t_1\ t_2)\\ \forall x\,t.\ P\ t \Rightarrow P\ (\lambda x.t)\end{array}}{P\ t}$$

implies

$$\frac{\begin{array}{l}\forall x\,c.\ P\,c\,x\\ \forall t_1\,t_2\,c.\ (\forall d.\,P\,d\,t_1) \wedge (\forall d.\,P\,d\,t_2) \Rightarrow P\,c\,(t_1\,t_2)\\ \forall x\,t\,c.\ x\ \#\ c \wedge (\forall d.\,P\,d\,t) \Rightarrow P\,c\,(\lambda x.t)\end{array}}{P\,c\,t}$$

# Proof for the Strong Induction Principle

- We prove $P\,c\,t$ by induction on $t$.

# Proof for the Strong Induction Principle

- We prove $\forall \pi\, c.\ P c\, (\pi \bullet t)$ by induction on $t$.

# Proof for the Strong Induction Principle

- We prove $\forall \pi\, c.\ P c\, (\pi \bullet t)$ by induction on $t$.
- I.e., we have to show $P c\, (\pi \bullet (\lambda x.t))$.

# Proof for the Strong Induction Principle

- We prove $\forall\pi\, c.\ P c\,(\pi \bullet t)$ by induction on $t$.
- I.e., we have to show $P c\, \lambda(\pi \bullet x).(\pi \bullet t)$.

# Proof for the Strong Induction Principle

- We prove $\forall \pi\, c.\ P c\, (\pi \bullet t)$ by induction on $t$.
- I.e., we have to show $P c\, \lambda(\pi \bullet x).(\pi \bullet t)$.
- We have $\forall \pi\, c.\ P c\, (\pi \bullet t)$ by induction.

# Proof for the Strong Induction Principle

- We prove $\forall \pi\, c.\ P c\, (\pi \bullet t)$ by induction on $t$.
- I.e., we have to show $P c\, \lambda(\pi \bullet x).(\pi \bullet t)$.
- We have $\forall \pi\, c.\ P c\, (\pi \bullet t)$ by induction.
- Our weaker precondition says that:

$$\forall x\, t\, c.\, x \# c \wedge (\forall c.\, P c\, t) \Rightarrow P c\, (\lambda x.t)$$

# Proof for the Strong Induction Principle

- We prove $\forall \pi\, c.\ P c\, (\pi \bullet t)$ by induction on $t$.
- I.e., we have to show $P c\, \lambda(\pi \bullet x).(\pi \bullet t)$.
- We have $\forall \pi\, c.\ P c\, (\pi \bullet t)$ by induction.
- Our weaker precondition says that:

$$\forall x\, t\, c.\, x \,\#\, c \wedge (\forall c.\, P c\, t) \Rightarrow P c\, (\lambda x.t)$$

- We choose a fresh $y$ such that $y \,\#\, (\pi \bullet x, \pi \bullet t, c)$.

# Proof for the Strong Induction Principle

- We prove $\forall \pi\, c.\ P\,c\,(\pi \bullet t)$ by induction on $t$.
- I.e., we have to show $P\,c\,\lambda(\pi \bullet x).(\pi \bullet t)$.
- We have $\forall \pi\, c.\ P\,c\,(\pi \bullet t)$ by induction.
- Our weaker precondition says that:

$$\forall x\,t\,c.\, x \,\#\, c \wedge (\forall c.\, P\,c\,t) \Rightarrow P\,c\,(\lambda x.t)$$

- We choose a fresh $y$ such that $y \,\#\, (\pi \bullet x, \pi \bullet t, c)$.
- Now we can use $\forall c.\ P\,c\,(((y\ \ \pi \bullet x) :: \pi) \bullet t)$

# Proof for the Strong Induction Principle

- We prove $\forall \pi\, c.\ P\, c\, (\pi \bullet t)$ by induction on $t$.
- I.e., we have to show $P\, c\, \lambda(\pi \bullet x).(\pi \bullet t)$.
- We have $\forall \pi\, c.\ P\, c\, (\pi \bullet t)$ by induction.
- Our weaker precondition says that:

$$\forall x\, t\, c.\, x \,\#\, c \wedge (\forall c.\, P\, c\, t) \Rightarrow P\, c\, (\lambda x.t)$$

- We choose a fresh $y$ such that $y \,\#\, (\pi \bullet x, \pi \bullet t, c)$.
- Now we can use $\forall c.\ P\, c\, ((y\ \ \pi \bullet x) \bullet \pi \bullet t)$

# Proof for the Strong Induction Principle

- We prove $\forall\pi\, c.\ P c\,(\pi\bullet t)$ by induction on $t$.
- I.e., we have to show $P c\,\lambda(\pi\bullet x).(\pi\bullet t)$.
- We have $\forall\pi\, c.\ P c\,(\pi\bullet t)$ by induction.
- Our weaker precondition says that:

$$\forall x\, t\, c.\, x \,\#\, c \wedge (\forall c.\, P c\, t) \Rightarrow P c\,(\lambda x.t)$$

- We choose a fresh $y$ such that $y \,\#\, (\pi\bullet x, \pi\bullet t, c)$.
- Now we can use $\forall c.\ P c\,((y\ \pi\bullet x)\bullet\pi\bullet t)$ to infer

$$P\, c\,\lambda y.((y\ \pi\bullet x)\bullet\pi\bullet t)$$

# Proof for the Strong Induction Principle

- We prove $\forall \pi\ c.\ P c\, (\pi \bullet t)$ by induction on $t$.
- I.e., we have to show $P c\, \lambda(\pi \bullet x).(\pi \bullet t)$.
- We have $\forall \pi\ c.\ P c\, (\pi \bullet t)$ by induction.
- Our weak

$$\frac{x \neq y \quad t_1 = (x\, y) \bullet t_2 \quad y \,\#\, t_2}{\lambda y.t_1 = \lambda x.t_2}$$

$$\forall x\, t \qquad \cdot t)$$

- We choose a fresh $y$ such that $y \,\#\, (\pi \bullet x, \pi \bullet t, c)$.
- Now we can use $\forall c.\ P c\, ((y\ \ \pi \bullet x) \bullet \pi \bullet t)$ to infer

$$P\, c\, \lambda y.((y\ \ \pi \bullet x) \bullet \pi \bullet t)$$

- However

$$\lambda y.((y\ \ \pi \bullet x) \bullet \pi \bullet t) = \lambda(\pi \bullet x).(\pi \bullet t)$$

# Proof for the Strong Induction Principle

- We prove $\forall \pi\, c.\ P c\, (\pi \bullet t)$ by induction on $t$.
- I.e., we have to show $P c\, \lambda(\pi \bullet x).(\pi \bullet t)$.
- We have $\forall \pi\, c.\ P c\, (\pi \bullet t)$ by induction.
- Our weaker precondition says that:

$$\forall x\, t\, c.\, x \# c \wedge (\forall c.\, P c\, t) \Rightarrow P c\, (\lambda x.t)$$

- We choose a fresh $y$ such that $y \# (\pi \bullet x, \pi \bullet t, c)$.
- Now we can use $\forall c.\ P c\, ((y\ \ \pi \bullet x) \bullet \pi \bullet t)$ to infer

$$P\, c\, \lambda y.((y\ \ \pi \bullet x) \bullet \pi \bullet t)$$

- However

$$\lambda y.((y\ \ \pi \bullet x) \bullet \pi \bullet t) = \lambda(\pi \bullet x).(\pi \bullet t)$$

- Therefore $P\, c\, \lambda(\pi \bullet x).(\pi \bullet t)$ and we are done.

# This Proof in Isabelle

```
lemma lam_strong_induct:
  fixes c::"'a::fs_name"
  assumes h1: "⋀x c. P c (Var x)"
  and     h2: "⋀t1 t2 c. ⟦∀d. P d t1; ∀d. P d t2⟧ ⟹ P c (App t1 t2)"
  and     h3: "⋀x t c. ⟦x#c; ∀d. P d t⟧ ⟹ P c (Lam [x].t)"
  shows "P c t"
proof -

  have "∀(π::name prm) c. P c (π•t)"   . . .
  then have "P c (([]::name prm)•t)" by blast
  then show "P c t" by simp
qed
```

interesting bit

# Interesting Bit

```
...
have "∀ (π::name prm) c. P c (π • t)"
proof (induct t rule: lam.induct)
  case (Lam x t)
  have ih: "∀ (π::name prm) c. P c (π • t)" by fact
  { fix π::"name prm" and c::"'a::fs_name"
    obtain y::"name" where fc: "y#(π • x,π • t,c)"
      by (rule exists_fresh) (auto simp add: fs_name1)
    from ih have "∀ c. P c (([(y,π • x)]@π) • t)" by simp
    then have "∀ c. P c ([(y,π • x)] • (π • t))" by (auto simp only: pt_name2)
    with $h_3$ have "P c (Lam [y].[(y,π • x)] • (π • t))" using fc by (simp add: fresh_prod)
    moreover
    have "Lam [y].[(y,π • x)] • (π • t) = Lam [(π • x)].(π • t)"
      using fc by (simp add: lam.inject alpha fresh_atm fresh_prod)
    ultimately have "P c (Lam [(π • x)].(π • t))" by simp
  }
  then have "∀ (π::name prm) c. P c (Lam [(π • x)].(π • t))" by simp
  then show "∀ (π::name prm) c. P c (π • (Lam [x].t))" by simp
qed (auto intro: $h_1$ $h_2$)
...
```

# Interesting Bit

```
. . .
have "∀(π::name prm) c. P c (π•t)"
proof (induct t rule: lam.induct)
  case (Lam x t)
  have ih: "∀(π::name prm) c. P c (π•t)" by fact
  { fix π::"name prm" and c::"'a::fs_name"
    obtain y::"name" where fc: "y#(π•x,π•t,c)"
      by (rule exists_fresh) (auto simp add: fs_name1)
    from ih have "∀c. P c (([(y,π•x)]@π)•t)" by simp
    then have "∀c. P c ([(y,π•x)]•(π•t))" by (auto simp only: pt_name2)
    with h3 have "P c (Lam [y].[(y,π•x)]•(π•t))" using fc by (simp add: fresh_prod)
    moreover
    have "Lam [y].[(y,π•x)]•(π•t) = Lam [(π•x)].(π•t)"
      using fc by (simp add: lam.inject alpha fresh_atm fresh_prod)
    ultimately have "P c (Lam [(π•x)].(π•t))" by simp
  }
  then have "∀(π::name prm) c. P c (Lam [(π•x)].(π•t))" by simp
  then show "∀(π::name prm) c. P c (π•(Lam [x].t))" by simp
qed (auto intro: h1 h2)
. . .
```

# Interesting Bit

```
...
have "∀(π::name prm) c. P c (π•t)"
proof (induct t rule: lam.induct)
  case (Lam x t)
  have ih: "∀(π::name prm) c. P c (π•t)" by fact
  { fix π::"name prm" and c::"'a::fs_name"
    obtain y::"name" where fc: "y#(π•x,π•t,c)"
     by (rule exists_fresh) (auto simp add: fs_name1)
    from ih have "∀ c. P c (([(y,π•x)]@π)•t)" by simp
    then have "∀ c. P c ([(y,π•x)]•(π•t))" by (auto simp only: pt_name2)
    with $h_3$ have "P c (Lam [y].[(y,π•x)]•(π•t))" using fc by (simp add: fresh_prod)
    moreover
    have "Lam [y].[(y,π•x)]•(π•t) = Lam [(π•x)].(π•t)"
     using fc by (simp add: lam.inject alpha fresh_atm fresh_prod)
    ultimately have "P c (Lam [(π•x)].(π•t))" by simp
  }
  then have "∀(π::name prm) c. P c (Lam [(π•x)].(π•t))" by simp
  then show "∀(π::name prm) c. P c (π•(Lam [x].t))" by simp
qed (auto intro: $h_1$ $h_2$)
...
```

# Interesting Bit

```
...
have "∀(π::name prm) c. P c (π•t)"
proof (induct t rule: lam.induct)
  case (Lam x t)
  have ih: "∀(π::name prm) c. P c (π•t)" by fact
  { fix π::"name prm" and c::"'a::fs_name"
    obtain y::"name" where fc: "y#(π•x,π•t,c)"
      by (rule exists_fresh) (auto simp add: fs_name1)
    from ih have "∀c. P c ((([(y,π•x)]@π)•t)" by simp
    then have "∀c. P c ([(y,π•x)]•(π•t))" by (auto simp only: pt_name2)
    with h_3 have "P c (Lam [y].[(y,π•x)]•(π•t))" using fc by (simp add: fresh_prod)
    moreover
    have "Lam [y].[(y,π•x)]•(π•t) = Lam [(π•x)].(π•t)"
      using fc by (simp add: lam.inject alpha fresh_atm fresh_prod)
    ultimately have "P c (Lam [(π•x)].(π•t))" by simp
  }
  then have "∀(π::name prm) c. P c (Lam [(π•x)].(π•t))" by simp
  then show "∀(π::name prm) c. P c (π•(Lam [x].t))" by simp
qed (auto intro: h_1 h_2)
...
```

# Interesting Bit

```
...
have "∀ (π::name prm) c. P c (π • t)"
proof (induct t rule: lam.induct)
  case (Lam x t)
  have ih: "∀ (π::name prm) c. P c (π • t)" by fact
  { fix π::"name prm" and c::"'a::fs_name"
    obtain y::"name" where fc: "y#(π • x,π • t,c)"
      by (rule exists_fresh) (auto simp add: fs_name1)
    from ih have "∀ c. P c (([(y,π • x)]@π) • t)" by simp
    then have "∀ c. P c ([(y,π • x)] • (π • t))" by (auto simp only: pt_name2)
    with h_3 have "P c (Lam [y].[(y,π • x)] • (π • t))" using fc by (simp add: fresh_prod)
    moreover
    have "Lam [y].[(y,π • x)] • (π • t) = Lam [(π • x)].(π • t)"
      using fc by (simp add: lam.inject alpha fresh_atm fresh_prod)
    ultimately have "P c (Lam [(π • x)].(π • t))" by simp
  }
  then have "∀ (π::name prm) c. P c (Lam [(π • x)].(π • t))" by simp
  then show "∀ (π::name prm) c. P c (π • (Lam [x].t))" by simp
qed (auto intro: h_1 h_2)
...
```

# Interesting Bit

```
...
have "∀(π::name prm) c. P c (π•t)"
proof (induct t rule: lam.induct)
 case (Lam x t)
 have ih: "∀(π::name prm) c. P c (π•t)" by fact
 { fix π::"name prm" and c::"'a::fs_name"
  obtain y::"name" where fc: "y#(π•x,π•t,c)"
   by (rule exists_fresh) (auto simp add: fs_name1)
  from ih have "∀c. P c (([(y,π•x)]@π)•t)" by simp
  then have "∀c. P c ([(y,π•x)]•(π•t))" by (auto simp only: pt_name2)
  with h_3 have "P c (Lam [y].[(y,π•x)]•(π•t))" using fc by (simp add: fresh_prod)
  moreover
  have "Lam [y].[(y,π•x)]•(π•t) = Lam [(π•x)].(π•t)"
   using fc by (simp add: lam.inject alpha fresh_atm fresh_prod)
  ultimately have "P c (Lam [(π•x)].(π•t))" by simp
 }
 then have "∀(π::name prm) c. P c (Lam [(π•x)].(π•t))" by simp
 then show "∀(π::name prm) c. P c (π•(Lam [x].t))" by simp
qed (auto intro: h_1 h_2)
...
```

# Interesting Bit

```
...
have "∀(π::name prm) c. P c (π•t)"
proof (induct t rule: lam.induct)
 case (Lam x t)
 have ih: "∀(π::name prm) c. P c (π•t)" by fact
 { fix π::"name prm" and c::"'a::fs_name"
  obtain y::"name" where fc: "y#(π•x,π•t,c)"
   by (rule exists_fresh) (auto simp add: fs_name1)
  from ih have "∀c. P c (([(y,π•x)]@π)•t)" by simp
  then have "∀c. P c ([(y,π•x)]•(π•t))" by (auto simp only: pt_name2)
  with h3 have "P c (Lam [y].[(y,π•x)]•(π•t))" using fc by (simp add: fresh_prod)
  moreover
  have "Lam [y].[(y,π•x)]•(π•t) = Lam [(π•x)].(π•t)"
   using fc by (simp add: lam.inject alpha fresh_atm fresh_prod)
  ultimately have "P c (Lam [(π•x)].(π•t))" by simp
 }
 then have "∀(π::name prm) c. P c (Lam [(π•x)].(π•t))" by simp
 then show "∀(π::name prm) c. P c (π•(Lam [x].t))" by simp
qed (auto intro: h1 h2)
...
```

$h_3$: "$\bigwedge$x t c. ⟦x # c; ∀ d. P d t⟧ $\Longrightarrow$ P c Lam [x].t"

# Interesting Bit

```
...
have "∀ (π::name prm) c. P c (π • t)"
proof (induct t rule: lam.induct)
  case (Lam x t)
  have ih: "∀ (π::name prm) c. P c (π • t)" by fact
  { fix π::"name prm" and c::"'a::fs_name"
    obtain y::"name" where fc: "y#(π • x,π • t,c)"
      by (rule exists_fresh) (auto simp add: fs_name1)
    from ih have "∀ c. P c (([(y,π • x)]@π) • t)" by simp
    then have "∀ c. P c ([(y,π • x)] • (π • t))" by (auto simp only: pt_name2)
    with h3 have "P c (Lam [y].[(y,π • x)] • (π • t))" using fc by (simp add: fresh_prod)
    moreover
    have "Lam [y].[(y,π • x)] • (π • t) = Lam [(π • x)].(π • t)"
      using fc by (simp add: lam.inject alpha fresh_atm fresh_prod)
    ultimately have "P c (Lam [(π • x)].(π • t))" by simp
  }
  then have "∀ (π::name prm) c. P c (Lam [(π • x)].(π • t))" by simp
  then show "∀ (π::name prm) c. P c (π • (Lam [x].t))" by simp
qed (auto intro: h1 h2)
...
```

# Interesting Bit

```
...
have "∀(π::name prm) c. P c (π•t)"
proof (induct t rule: lam.induct)
  case (Lam x t)
  have ih: "∀(π::name prm) c. P c (π•t)" by fact
  { fix π::"name prm" and c::"'a::fs_name"
    obtain y::"name" where fc: "y#(π•x,π•t,c)"
      by (rule exists_fresh) (auto simp add: fs_name1)
    from ih have "∀c. P c (([(y,π•x)]@π)•t)" by simp
    then have "∀c. P c ([(y,π•x)]•(π•t))" by (auto simp only: pt_name2)
    with h3 have "P c (Lam [y].[(y,π•x)]•(π•t))" using fc by (simp add: fresh_prod)
    moreover
    have "Lam [y].[(y,π•x)]•(π•t) = Lam [(π•x)].(π•t)"
      using fc by (simp add: lam.inject alpha fresh_atm fresh_prod)
    ultimately have "P c (Lam [(π•x)].(π•t))" by simp
  }
  then have "∀(π::name prm) c. P c (Lam [(π•x)].(π•t))" by simp
  then show "∀(π::name prm) c. P c (π•(Lam [x].t))" by simp
qed (auto intro: h1 h2)
...
```

# Interesting Bit

```
...
have "∀ (π::name prm) c. P c (π • t)"
proof (induct t rule: lam.induct)
  case (Lam x t)
  have ih: "∀ (π::name prm) c. P c (π • t)" by fact
  { fix π::"name prm" and c::"'a::fs_name"
    obtain y::"name" where fc: "y#(π • x,π • t,c)"
      by (rule exists_fresh) (auto simp add: fs_name1)
    from ih have "∀ c. P c (([(y,π • x)]@π) • t)" by simp
    then have "∀ c. P c ([(y,π • x)] • (π • t))" by (auto simp only: pt_name2)
    with h3 have "P c (Lam [y].[(y,π • x)] • (π • t))" using fc by (simp add: fresh_prod)
    moreover
    have "Lam [y].[(y,π • x)] • (π • t) = Lam [(π • x)].(π • t)"
      using fc by (simp add: lam.inject alpha fresh_atm fresh_prod)
    ultimately have "P c (Lam [(π • x)].(π • t))" by simp
  }
  then have "∀ (π::name prm) c. P c (Lam [(π • x)].(π • t))" by simp
  then show "∀ (π::name prm) c. P c (π • (Lam [x].t))" by simp
qed (auto intro: h1 h2)
...
```

# Some Examples

$$\frac{x \# \Gamma \quad (x, T_1)::\Gamma \vdash t : T_2}{\Gamma \vdash \mathrm{Lam}\,[x].t : T_1 \to T_2}$$

# Some Examples

$$\frac{x \# \Gamma \quad (x, T_1)::\Gamma \vdash t : T_2}{\Gamma \vdash \mathsf{Lam}\ [x].t : T_1 \to T_2}$$

$$\frac{t \mapsto t'}{\mathsf{Lam}\ [x].t \mapsto t'}$$

# Some Examples

$$\frac{x \# \Gamma \quad (x, T_1)::\Gamma \vdash t : T_2}{\Gamma \vdash \mathrm{Lam}\,[x].t : T_1 \rightarrow T_2}$$

$$\frac{t \mapsto t'}{\mathrm{Lam}\,[x].t \mapsto t'}$$

# Some Examples

$$\frac{x \# \Gamma \quad (x, T_1)::\Gamma \vdash t : T_2}{\Gamma \vdash \mathsf{Lam}\,[x].t : T_1 \to T_2}$$

$$\frac{t \mapsto t'}{\mathsf{Lam}\,[x].t \mapsto t'}$$

$$\frac{\Gamma \vdash_\Sigma A_1 : \mathsf{Type} \quad (x, A_1)::\Gamma \vdash_\Sigma M_2 : A_2 \quad x \# (\Gamma, A_1)}{\Gamma \vdash_\Sigma \mathsf{Lam}\,[x{:}A_1].M_2 : \Pi[x{:}A_1].A_2}$$

# Some Examples

$$\frac{x \# \Gamma \qquad (x, T_1)::\Gamma \vdash t : T_2}{\Gamma \vdash \mathrm{Lam}\,[x].t : T_1 \to T_2}$$

$$\frac{t \mapsto t'}{[x].t \mapsto t'}$$

free

$$\frac{\Gamma \vdash_{\Sigma} A_1 : \mathrm{Type} \qquad (x, A_1)::\Gamma \vdash_{\Sigma} M_2 : A_2 \qquad x \# (\Gamma, A_1)}{\Gamma \vdash_{\Sigma} \mathrm{Lam}\,[x{:}A_1].M_2 : \Pi[x{:}A_1].A_2}$$

bound

bound

# Some Examples

$$\frac{x \# \Gamma \qquad (x, T_1)::\Gamma \vdash t : T_2}{\Gamma \vdash \mathsf{Lam}\,[x].t : T_1 \rightarrow T_2}$$

$$\frac{t \mapsto t'}{\mathsf{Lam}\,[x].t \mapsto t'}$$

$$\frac{\Gamma \vdash_\Sigma A_1 : \mathsf{Type} \qquad (x, A_1)::\Gamma \vdash_\Sigma M_2 : A_2 \qquad x \# (\Gamma, A_1)}{\Gamma \vdash_\Sigma \mathsf{Lam}\,[x \qquad \Pi[x{:}A_1].}$$

free free free

$$\frac{(x, \tau_1)::\Delta \vdash_\Sigma \mathsf{App}\,M\,(\mathsf{Var}\,x) \Leftrightarrow \mathsf{App}\,N\,(\mathsf{Var}\,x) : \tau_2 \qquad x \# (\Delta, M, N)}{\Delta \vdash_\Sigma M \Leftrightarrow N : \tau_1 \rightarrow \tau_2}$$

# Some Examples

$$\frac{x \# \Gamma \quad (x, T_1)::\Gamma \vdash t : T_2}{\Gamma \vdash \mathsf{Lam}\,[x].t : T_1 \rightarrow T_2}$$

$$\frac{t \mapsto t'}{\mathsf{Lam}\,[x].t \mapsto t'}$$

$$\frac{\Gamma \vdash_\Sigma A_1 : \mathsf{Type} \quad (x, A_1)::\Gamma \vdash_\Sigma M_2 : A_2 \quad x \# (\Gamma, A_1)}{\Gamma \vdash_\Sigma \mathsf{Lam}\,[x{:}A_1].M_2 : \Pi[x{:}A_1].A_2}$$

$$\frac{\begin{array}{c}(x, \tau_1)::\Delta \vdash_\Sigma \mathsf{App}\;M\;(\mathsf{Var}\;x) \Leftrightarrow \mathsf{App}\;N\;(\mathsf{Var}\;x) : \tau_2 \\ x \# (\Delta, M, N)\end{array}}{\Delta \vdash_\Sigma M \Leftrightarrow N : \tau_1 \rightarrow \tau_2}$$

# Conclusions

- The Nominal Isabelle automatically derives the strong structural induction principle for **all** nominal datatypes (not just the lambda-calculus);
- also for rule inductions (though they have to satisfy a vc-condition).
- They are easy to use: you just have to think carefully what the variable convention should be.
- We can explore the dark corners of the variable convention: when and where it can actually be used.

# Conclusions

- The Nominal Isabelle automatically derives the strong structural induction principle for **all** nominal datatypes (not just the lambda-calculus);
- also for rule inductions (though they have to satisfy a vc-condition).
- They are easy to use: you just have to think carefully what the variable convention should be.
- We can explore the dark corners of the variable convention: when and where it can actually be used.
- **Main Point:** Actually these proofs using the variable convention are all trivial / obvious / routine. . . **provided** you use Nominal Isabelle. ;o)

# Next

- How do we deal with statements such as “Expressions differing only in names of bound variables are equivalent”.

$$\lambda x.x = \lambda y.y$$

- **Exercise**: Find a short proof for the weakening lemma that does not rely on the variable convention.