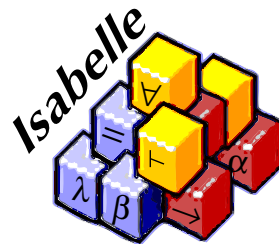


# Nominal Datatypes in Isabelle/HOL

Stefan Berghofer  
with

Christian Urban, Julien Narboux and Markus Wenzel



# Motivation

## A paper proof [Barendregt, 1981]

**Substitution lemma:** If  $x \neq y$  and  $x \notin FV(L)$ , then

$$M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]].$$

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

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

Case 1.1.  $M = x$ . Then both sides equal  $N[y \mapsto L]$ , since  $x \neq y$ .

Case 1.2.  $M = y$ . Then both sides equal  $L$ , for  $x \notin FV(L)$  implies  $L[x \mapsto \dots] = L$ .

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  $N, L$ . Then by induction hypothesis

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

**Case 3:**  $M = M_1 M_2$ . The statement follows again from the induction hypothesis.

□

## What the experts say...

*“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.”*

Stephen C. Kleene in *Journal of Symbolic Logic* 21(1):11-18, 1962

*“When doing the formalization, I discovered that the core part of the proof ... is fairly straightforward and only requires a good understanding of the paper version. However, in completing the proof I observed that in certain places I had to invest much more work than expected, e.g. proving lemmas about substitution and weakening.”*

Thorsten Altenkirch in *Proceedings of TLCA*, 1993

*“Proving theorems about substitutions (and related operations such as  $\alpha$ -conversion) required far more time than any other variety of theorem.”*

Myra VanInwegen in her PhD-thesis, 1996

⇒ **Better tool support necessary**

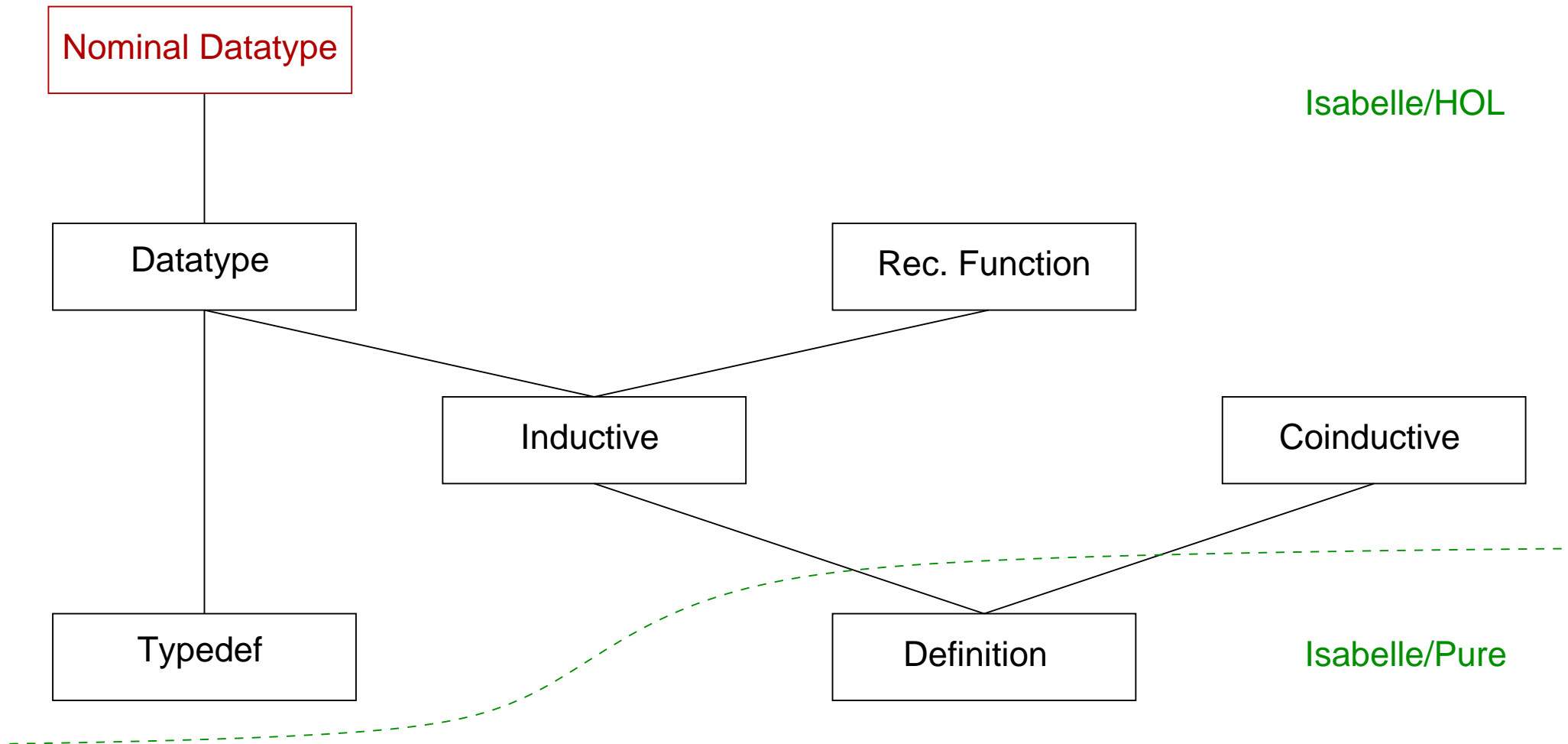
## Our tool: Isabelle

- Developed (since 1986) by Larry Paulson (Cambridge) and Tobias Nipkow
- Interactive theorem prover
- Logical Framework
  - Description of various **object logics** using a **meta logic** (Isabelle/Pure)
- Most well-developed object logic: Isabelle/HOL
- Design philosophy
  - Inferences may only be performed by a **small kernel** (“LCF approach”)
  - **Definitional theory extension**
    - New concepts (such as inductive datatypes and predicates) must be defined using already existing concepts.

*“The method of ‘postulating’ what we want has many advantages; they are the same as the advantages of theft over honest toil. Let us leave them to others and proceed with our honest toil.”*

Bertrand Russell, Introduction to Mathematical Philosophy

# Hierarchy of definitional packages



# Existing approaches for reasoning with bound variables

- “Name-carrying” syntax
  - + readable
  - $\alpha$ -equivalence must be formalized explicitly
  - substitution function requires variable renaming
- De Bruijn indices
  - +  $\alpha$ -equivalence coincides with syntactic equality
  - + simple induction / recursion principles
  - substitution function requires index calculations
  - unreadable
- “Locally nameless” approach
  - +  $\alpha$ -equivalence coincides with syntactic equality
  - + substitution function does not require index calculations
  - well-formedness must be formalized explicitly
- Higher order abstract syntax (in Isabelle or Coq, for example)
  - + abstraction and substitution “for free”
  - exotic terms

# Our approach

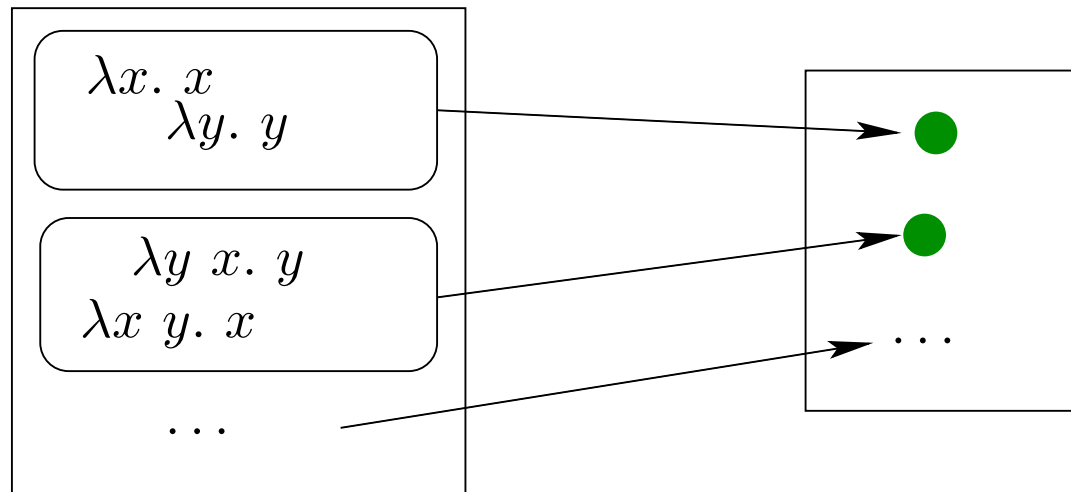
## A more abstract approach

**Problem:** How can we hide details of the representation from the user?

### Possible solution:

Introduce a new type, whose elements correspond to...

- ... the  $\alpha$ -equivalence classes of “name-carrying”-terms, or
- ... the well-formed “locally nameless” terms.



**Question:** How does abstract “interface” for this type look like?  $\implies$  **Nominal logic!**

# Nominal logic

[A. M. Pitts et al, TACS 2001, I&C 2003]

- Specific types for **names** (with infinitely many elements)
- **Permutations**:  $\pi \bullet t$  (bijective)  
where  $\pi = [(a_1, b_1), \dots, (a_n, b_n)]$ ,  $a_i$  and  $b_i$  are **names**
- **Support** ( $\approx$  set of free variables):  $supp\ t$ 
  - **Note**: Nominal logic is incompatible with the axiom of choice (which is part of Isabelle/HOL), because “everything” is assumed to have finite support
  - Not all HOL terms and types have finite support
  - We use extra preconditions to characterize terms and types that have finite support
  - Some preconditions can be “hidden” using **axiomatic type classes**
  - Nominal datatypes always have **finite support**
- **Freshness**:  $a \# t \equiv a \notin supp\ t$

# Nominal datatypes

Datatypes with **abstractions**

**nominal-datatype**  $\vec{\alpha} \text{ ty} = \dots | C_i \ll \vec{a}_i^1 \gg \tau_i^1 \dots \ll \vec{a}_i^{m_i} \gg \tau_i^{m_i} | \dots$

where  $\vec{a}_i^j$  are lists of atom types and  $\tau_i^j$  are types, possibly containing  $\vec{\alpha} \text{ ty}$

## Example: untyped $\lambda$ -calculus

**atom-decl**  $\text{name}$

**nominal-datatype**  $\text{term} = \text{Var } \text{name} | \text{App } \text{term } \text{term} | \text{Abs } \ll \text{name} \gg \text{term}$

## Limitations

$Lam_2 (\ll \text{name} \gg \ll \text{name} \gg \text{term})$	OK
$Lams (\ll \text{name list} \gg \text{term})$	Not yet supported
$Let (\ll \text{pattern} \gg \text{term}) \text{ term}$	Not yet supported

# Nominal datatypes – $\alpha$ -equivalence

## Standard datatypes

$$\text{Abs } a \ t = \text{Abs } b \ u \iff (a = b \wedge t = u)$$

## Nominal datatypes

$$\text{Abs } a \ t = \text{Abs } b \ u \iff (a = b \wedge t = u) \vee (a \neq b \wedge t = [(a, b)] \bullet u \wedge a \# u)$$

## Example

$$\text{Abs } a \ (\text{Var } a) = \text{Abs } b \ (\text{Var } b)$$

because

- $a \neq b$
- $\text{Var } a = \text{Var } ([ (a, b) ] \bullet b) = [(a, b)] \bullet (\text{Var } b)$
- $a \# \text{Var } b$

## Nominal datatypes – induction

### Weak induction rule

$$\begin{array}{l} \forall n. P (Var\ n) \\ \forall t\ u. P\ t \implies P\ u \implies P (App\ t\ u) \\ \forall n\ t. P\ t \implies P (Abs\ n\ t) \\ \hline \forall t. P\ t \end{array}$$

### Strong induction rule

$$\begin{array}{l} \forall n\ c. P\ c (Var\ n) \\ \forall t\ u\ c. (\forall d. P\ d\ t) \implies (\forall d. P\ d\ u) \implies P\ c (App\ t\ u) \\ \forall n\ t\ c. n\#\!c \implies (\forall d. P\ d\ t) \implies P\ c (Abs\ n\ t) \\ \hline \forall t\ c. P\ c\ t \end{array}$$

## Using the strong induction rule

**Substitution lemma:** If  $x \neq y$  and  $x \notin FV(L)$ , then

$$M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]].$$

...

**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  $N, L$ .

$$\forall z \ c. \ P \ c \ (\text{Var } z)$$

$$\forall M_1 \ M_2 \ c.$$

$$(\forall d. \ P \ d \ M_1) \implies (\forall d. \ P \ d \ M_2) \implies \\ P \ c \ (\text{App } M_1 \ M_2)$$

$$\forall z \ M_1 \ c. \ z \# c \implies (\forall d. \ P \ d \ M_1) \implies \\ P \ c \ (\text{Abs } z \ M_1)$$

---


$$\forall M \ c. \ P \ c \ M$$

$$P := \lambda(x, y, N, L). \ \lambda M. \ \mathcal{I} \ M \ x \ y \ N \ L$$

$$\mathcal{I} \ M \ x \ y \ N \ L \equiv x \neq y \implies x \notin FV(L) \implies M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]]$$

## Using the strong induction rule

**Substitution lemma:** If  $x \neq y$  and  $x \notin FV(L)$ , then

$$M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]].$$

...

**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  $N, L$ .

$$\forall z \ x \ y \ N \ L. \mathcal{I} (\text{Var } z) \ x \ y \ N \ L$$

$$\forall M_1 \ M_2 \ x \ y \ N \ L.$$

$$(\forall x' \ y' \ N' \ L'. \mathcal{I} M_1 \ x' \ y' \ N' \ L') \implies (\forall x' \ y' \ N' \ L'. \mathcal{I} M_2 \ x' \ y' \ N' \ L') \implies \\ \mathcal{I} (\text{App } M_1 \ M_2) \ x \ y \ N \ L$$

$$\forall z \ M_1 \ x \ y \ N \ L. z \#(x, y, N, L) \implies (\forall x' \ y' \ N' \ L'. \mathcal{I} M_1 \ x' \ y' \ N' \ L') \implies \\ \mathcal{I} (\text{Abs } z \ M_1) \ x \ y \ N \ L$$

---


$$\forall M \ x \ y \ N \ L. \mathcal{I} M \ x \ y \ N \ L$$

$$P := \lambda(x, y, N, L). \lambda M. \mathcal{I} M \ x \ y \ N \ L$$

$$\mathcal{I} M \ x \ y \ N \ L \equiv x \neq y \implies x \notin FV(L) \implies M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]]$$

## Recursive functions on $\alpha$ -equivalence classes

Is the following function well-defined?

$$\begin{aligned} bvars (Var\ x) &= \{\} \\ bvars (App\ t_1\ t_2) &= bvars\ t_1 \cup bvars\ t_2 \\ bvars (Abs\ x\ t) &= \{x\} \cup bvars\ t \end{aligned}$$

Unproblematic for standard datatypes...

... but not for nominal datatypes:

$$Abs\ a\ (Var\ a) = Abs\ b\ (Var\ b)$$

but

$$bvars\ (Abs\ a\ (Var\ a)) = \{a\} \neq \{b\} = bvars\ (Abs\ b\ (Var\ b))$$

$\implies$  **Result of function may not depend on choice of bound variable names!**

# Recursion combinator

## Standard datatypes

$$\begin{aligned} \text{term\_rec } f_1 f_2 f_3 (\text{Var } n) &= f_1 n \\ \text{term\_rec } f_1 f_2 f_3 (\text{App } t u) &= f_2 t u (\text{term\_rec } f_1 f_2 f_3 t) (\text{term\_rec } f_1 f_2 f_3 u) \\ \text{term\_rec } f_1 f_2 f_3 (\text{Abs } n t) &= f_3 n t (\text{term\_rec } f_1 f_2 f_3 t) \end{aligned}$$

## Nominal datatypes

$$\begin{aligned} &\text{freshness condition for binders} \\ n\#(f_1, f_2, f_3) \wedge (\forall n t r. n\#f_3 \implies n\#f_3 n t r) \implies \\ \text{term\_rec } f_1 f_2 f_3 (\text{Abs } n t) &= f_3 n t (\text{term\_rec } f_1 f_2 f_3 t) \end{aligned}$$

## Substitution function

$$\begin{aligned} (Var\ x)[y \mapsto u] &= (\text{if } x = y \text{ then } u \text{ else } (Var\ x)) \\ (App\ t_1\ t_2)[y \mapsto u] &= App\ (t_1[y \mapsto u])\ (t_2[y \mapsto u]) \\ x\#(y, u) \implies (Abs\ x\ t)[y \mapsto u] &= Abs\ x\ (t[y \mapsto u]) \end{aligned}$$

### Instantiation of the recursion combinator

$$\begin{aligned} f_1 &= \lambda x. \text{if } x = y \text{ then } u \text{ else } (Var\ x) \\ f_2 &= \lambda t_1\ t_2\ r_1\ r_2. App\ r_1\ r_2 \\ f_3 &= \lambda x\ t\ r. Abs\ x\ r \end{aligned}$$

### Freshness conditions for binders

$$\forall x\ t\ r. x\#f_3 \implies x\#Abs\ x\ r$$

#### Note:

$$\begin{aligned} &x\#Abs\ x\ r \\ &= x \notin \text{supp}\ (Abs\ x\ r) \\ &= x \notin \text{supp}\ r - \{x\} \end{aligned}$$

## Bound variables

$$\begin{aligned} \text{bvars } (\text{Var } x) &= \{\} \\ \text{bvars } (\text{App } t_1 t_2) &= \text{bvars } t_1 \cup \text{bvars } t_2 \\ \text{bvars } (\text{Abs } x t) &= \{x\} \cup \text{bvars } t \end{aligned}$$

### Instantiation of the recursion combinator

$$\begin{aligned} f_1 &= \lambda x. \{\} \\ f_2 &= \lambda t_1 t_2 r_1 r_2. r_1 \cup r_2 \\ f_3 &= \lambda x t r. \{x\} \cup r \end{aligned}$$

### Freshness conditions for binders

$$\forall x t r. x \# f_3 \implies x \# \{x\} \cup r$$

**Note:** a similar problem arises when attempting to define the “immediate subterms” function.

## Isabelle proof of substitution lemma

**lemma** *substitution-lemma*:

**assumes** *fresh*:  $x \neq y$   $x \# L$

**shows**  $M[x \mapsto N][y \mapsto L] = M[y \mapsto L][x \mapsto N[y \mapsto L]]$  **using** *fresh*

**proof** (*nominal-induct M avoiding: x y N L rule: lam.induct*)

**case** (*Abs z M<sub>1</sub>*)

**have**  $(\text{Abs } z \ M_1)[x \mapsto N][y \mapsto L] = \text{Abs } z \ (M_1[x \mapsto N][y \mapsto L])$

**using**  $\langle z \# x \rangle \langle z \# y \rangle \langle z \# N \rangle \langle z \# L \rangle$  **by** *simp*

**also from Abs have**  $\dots = \text{Abs } z \ (M_1[y \mapsto L][x \mapsto N[y \mapsto L]])$

**using**  $\langle x \neq y \rangle \langle x \# L \rangle$  **by** *simp*

**also have**  $\dots = (\text{Abs } z \ (M_1[y \mapsto L]))[x \mapsto N[y \mapsto L]]$

**using**  $\langle z \# x \rangle \langle z \# N \rangle \langle z \# L \rangle$  **by** (*simp add: fresh-fact*)

**also have**  $\dots = (\text{Abs } z \ M_1)[y \mapsto L][x \mapsto N[y \mapsto L]]$

**using**  $\langle z \# y \rangle \langle z \# L \rangle$  **by** *simp*

**finally show**  $(\text{Abs } z \ M_1)[x \mapsto N][y \mapsto L] = (\text{Abs } z \ M_1)[y \mapsto L][x \mapsto N[y \mapsto L]]$  .

**next**

...

**qed**

# Conclusion

# Conclusion

- **Bad news:** proofs about calculi with variable binding are inherently complicated
- **Good news:** some of the complexity can be hidden inside nominal datatype package
- Needs quite a bit of infrastructure  
nominal package is currently the biggest package in Isabelle
- Implementation is contained in **Isabelle Development Snapshot**
- Many applications: see Christian's talk (and mailing list)

## Further work

- Strengthened inversion principles
- More general binding constructs  
 $\text{let } p = t \text{ in } u$
- Generation of code from specifications involving nominal datatypes
- Support for more general recursion schemes

## Literature

- C. Urban and C. Tasson** *Nominal techniques in Isabelle/HOL*. In Proceedings of the 20th International Conference on Automated Deduction (CADE), volume 3632 of LNCS, Springer-Verlag 2005.
- C. Urban and S. Berghofer** *A Recursion Combinator for Nominal Datatypes Implemented in Isabelle/HOL*. In U. Furbach and N. Shankar, editors, Third International Joint Conference on Automated Reasoning (IJCAR 2006), LNCS 4130, Springer-Verlag 2006.
- C. Urban, S. Berghofer, and M. Norrish** *Barendregt's Variable Convention in Rule Inductions*. To appear in proceedings of CADE 2007.
- J. Bengtson and J. Parrow** *Formalising the pi-calculus using nominal logic*. In Proceedings of FOSSACS 2007, LNCS, Springer-Verlag 2007.

**See the web site: `isabelle.in.tum.de/nominal`**

**Thanks for your attention!**