

# *Logosphere*

**Carsten Schürmann**

IT University of Copenhagen

February 9, 2010



# The Coq Proof Assistant

## Library Coq.Sets.Finite\_sets\_facts

```
Require Export Finite_sets.
Require Export Constructive_sets.
Require Export Classical_Type.
Require Export Classical_sets.
Require Export Powerset.
Require Export Powerset_facts.
Require Export Powerset_Classical_facts.
Require Export Gt.
Require Export Lt.

Section Finite_sets_facts.
  Variable U : Type.

  Lemma finite_cardinal :
    forall X:Ensemble U, Finite U X -> exists n : nat, cardinal U X n.
```

# PVS

```
complex: THEORY
BEGIN
  complex: NONEMPTY_TYPE FROM number_field
  real_are_complex: AXIOM FORALL (x: real): complex_pred(x)
  JUDGEMENT real SUBTYPE_OF complex
  nonzero_complex: NONEMPTY_TYPE
    = {c: complex | c /= 0} CONTAINING 1
  nzcomplex: NONEMPTY_TYPE = nonzero_complex

  i: complex
  i_ax: AXIOM i * i = -1

  rep_exists: AXIOM
    FORALL (c: complex): EXISTS (x, y: real): c = x + y*i
  rep_unique: AXIOM
    FORALL (x1, x2, y1, y2: real):
      x1 + y1*i = x2 + y2*i <=> (x1 = x2 & y1 = y2)
END complex
```

# Theory Finite\_Set

[Up](#) to index of Isabelle/HOL

**theory** *Finite\_Set*  
**imports** [Power](#)

```
(* Title:      HOL/Finite_Set.thy
   Author:     Tobias Nipkow, Lawrence C Paulson and Markus Wenzel
               with contributions by Jeremy Avigad
*)

header {* Finite sets *}

theory Finite_Set
imports Nat Product_Type Power
begin

subsection {* Definition and basic properties *}

inductive finite :: "'a set => bool"
  where
    emptyI [simp, intro!]: "finite {}"
  | insertI [simp, intro!]: "finite A ==> finite (insert a A)"

lemma ex_new_if_finite: -- "does not depend on def of finite at all"
  assumes "\ finite (UNIV :: 'a set)" and "finite A"
  shows "\ a::'a. a \ A"
proof -
  from assms have "A \ UNIV" by blast
  thus ?thesis by blast
qed
```

# Summary

## *Phenomenon*

Many proof assistants offer very related libraries.

## *Hypothesis*

It is possible to share the development and maintenance of those digital libraries in a systematic and scientific way.

## *Methodology*

Build a digital library of formal proof.

## *State Of The Art*

- A large great unified theory of everything doesn't work. [QED project]
- Proof replay
  - Mizar  $\longrightarrow$  HOL Light [Flyspeck project]
  - Isabelle/HOL  $\longrightarrow$  HOL Light [see Peter's talk yesterday]
- **A foundational uncommitted logical framework needed.**
  - Logical Framework LF, CLF [Twelf project]
- **Proof translations/Theory interpretations**
  - HOL  $\longrightarrow$  Nuprl [Delphin project]
- Proof compression [Reed]
- Proof retrieval [Constable]

# Logical Framework LF

- Edinburgh Logical Framework
- Judgments as types.
- Derivation as objects.

[Harper et al]

$$\frac{\begin{array}{c} \frac{}{\vdash A} u \\ \vdots \\ \vdash B \end{array}}{\vdash A \supset B} \text{disch}^u$$

- In LF:  $\text{disch}: (|- A \rightarrow |- B) \rightarrow |- A \Rightarrow B$
- Encoding is adequate.

# HOL - NUPRL Connection

```
hol-nuprl.d

(* Lemma 5 *)

fun lemma5i : world1 -> world2 -> world3 -> world4
  -> <|- H > -> <T:n-tm> -> <transsen H T> -> <M:n-tm> * <|- M !*! T>
  = fn W1 W2 W3 W4 <HD> <T> <TSEN:transsen H T> =>
    let val (<T'>, <TSEN':transsen H T'>, <M>, <ND'>) = lemma5 W1 W2 W3 W4 <HD>
        val <Eq> = equal T' T = uniqueSen W4 <TSEN'> <TSEN>
        val <ND> = equalProp <Eq> <ND'>
    in
      (<M>, <ND>)
    end

and lemma5 : world1 -> world2 -> world3 -> world4
  -> <|- H > -> <T:n-tm> * <transsen H T> * <M:n-tm> * <|- M !*! T>
  = fn W1 W2 W3 W4 <u#> => W3 <u>
    | W1 W2 W3 W4 <bool-cases-ax : |- all (\ [x:tm bool] x == true ∨ x == false)>
    => (< ^ (app (lam ([p:n-tm] v (pi boolean ([x:n-tm] ^ app p x))))
      (lam
        ([x:n-tm]
          app
            (app (lam ([x1:n-tm] lam ([y:n-tm] if x1 tt y)))
              (app (app (=p= boolean) x) tt))
            (app (app (=p= boolean) x) ff))))),
      <t-base (trans@ (tc-all transtpo)
        (trans\
          ([x:tm bool] [y:n-tm] [x1:transtm x y]
            :- hol-nuprl.d 57% (345,27) CVS:1.22 (Text)
Switch to buffer (default *TeX Help*):
```

## *State Of The Art*

- A large great unified theory of everything doesn't work.  
[QED project]
- Proof replay
  - Mizar  $\longrightarrow$  HOL Light [Flyspeck project]
  - Isabelle/HOL  $\longrightarrow$  HOL Light
- A foundational uncommitted logical framework needed.
  - Logical Framework LF, CLF [Twelf project]
- Proof translations/Theory interpretations
  - HOL  $\longrightarrow$  Nuprl [Delphin project]
- Proof compression [Reed]
- **Proof retrieval** [Constable]

Search

### Searching for `ByteString -> ByteString`

#### `Data.ByteString` [copy](#) :: `ByteString -> ByteString`

`byteString`  $\oplus$   $O(n)$  Make a copy of the `ByteString` with its own storage. This is mainly useful to allow the rest of the data pointed to by the `ByteString` to be garbage collected, for example if a large string has been read in, and only a small part

#### `Data.ByteString` [init](#) :: `ByteString -> ByteString`

`byteString`  $\oplus$   $O(1)$  Return all the elements of a `ByteString` except the last one. An exception will be thrown in the case of an empty `ByteString`.

#### `Data.ByteString` [reverse](#) :: `ByteString -> ByteString`

`byteString`  $\oplus$   $O(n)$  `reverse` `xs` efficiently returns the elements of `xs` in reverse order.

#### `Data.ByteString` [sort](#) :: `ByteString -> ByteString`

`byteString`  $\oplus$   $O(n)$  Sort a `ByteString` efficiently, using counting sort.

#### `Data.ByteString` [tail](#) :: `ByteString -> ByteString`

`byteString`  $\oplus$   $O(1)$  Extract the elements after the head of a `ByteString`, which must be non-empty. An exception will be thrown in the case of an empty `ByteString`.

#### `Codec.Compression.GZip` [compress](#) :: `ByteString -> ByteString`

`zlib`  $\oplus$  Compress a stream of data into the `gzip` format. This uses the default compression parameters. In particular it uses the default compression level which favours a higher compression ratio over compression speed, though it

#### `Codec.Compression.Zlib` [compress](#) :: `ByteString -> ByteString`

`zlib`  $\oplus$  Compress a stream of data into the `zlib` format. This uses the default compression parameters. In particular it uses

## *Challenges*

- Identifying base domains. Reals do not necessarily equal reals.
- Bridging the logical strength divide.
- Organization of information in the digital library.
- Handling inconsistencies.
- Accessibility to non-academic institutions.
- How to google a mathematical proof in a meaningful way.

## *Core Consortium*

- myself [IT University of Copenhagen]
- Dale Miller [INRIA, École Polytechnique]
- Kaustuv Chaudhuri [INIRIA]
- Michael Kohlhase [Jacobs University]
- Florian Rabe [Jacobs University]