

# Design of the next 700 Proof Assistants

G rard Huet

Projet Coq

INRIA - Rocquencourt



[ FLoC - Rutgers - July 28th 1996 - 1 ]

## PLAN

- General Design Issues
- Zippers
- Böhm Trees
- Mathematics modelling

## General Design Issues

- Why yet another proof assistant?
- Overall architecture

## A few popular proof assistants

- ACL2
- LCF-HOL
- NuPRL
- PVS
- IMPS , *MIZAR*
- Isabelle
- Coq-Alf-Lego
- B
- Elf
- LP, etc etc

YAPA?

- Who will use it?
- For what concrete purpose?
- What is the competition?
- How could we share the effort?

## Example 1 : proof trees

- Should one keep proof trees around?
- + Program extraction
- + Self-certified mobile code
- + Auditing proofs in natural language
  - Size vs Speed
  - Decision procedures, rewriting

## Example 2 : libraries

- What should the library structure be?
- Modularity, Naming
- Search for relevant lemmas
- Phase distinction
- Dependency analysis
- Sharing with other sites
- Version management
- Sharing with other provers (QED!)



## Overall architecture

- Logical Framework
- Algorithms
- Programmability
- User interface
- User-defined notation?
- Shared data base of proven facts
- User assistance
- Software engineering, tools
- Will it scale up?



I got this in the mail today

The following articles are submitted to the Mizar Mathematical Library:

1. Piotr Rudnicki and Andrzej Trybulec submitted an article entitled: "On the compositions of macro instructions"
2. Yatsuka Nakamura and Andrzej Trybulec submitted articles entitled: "Some Topological Properties of Cells in  $R^2$ " and "The First Part of Jordan's Theorem for Special Polygons"
3. Noriko Asamoto , Yatsuka Nakamura , Piotr Rudnicki and Andrzej Trybulec submitted articles entitled: "On the compositions of macro instructions, Part II" and "On the compositions of macro instructions, Part III"

The preprints of the above articles are available at:

<http://math.uw.bialystok.pl/~Form.Math/Preprints>

Czeslaw Bylinski

Library Committee of the Association of Mizar Users



[ FLoC - Rutgers - July 28th 1996 - 9 ]

## Abstract Syntax

Any implementer of a proof assistant, a programming environment, or a formal computation system, is faced early on with the problem of designing an abstract syntax framework. Typical concerns are:

- well-formedness wrt arities of operators
- list structures
- binding operators, local definitions
- recursion

Usual solutions are:

- LISP
- free algebras + lists
- pure  $\lambda$ -calculus
- $\lambda - \Pi$  | Automath | Elf



[ FLoC - Rutgers - July 28th 1996 - 10 ]

## Managing the state

- Version management
- Dumps, Saves, Updates
- Global context
- Undo
- Pragmas and flags
- Clicking, structure editing
- Notation
- Applicative structures vs Destructive editing

## The zipper data structure

- Mentor
- Applicative arrays
- Emacs
- hierarchical local Turing machine

## The zipper type

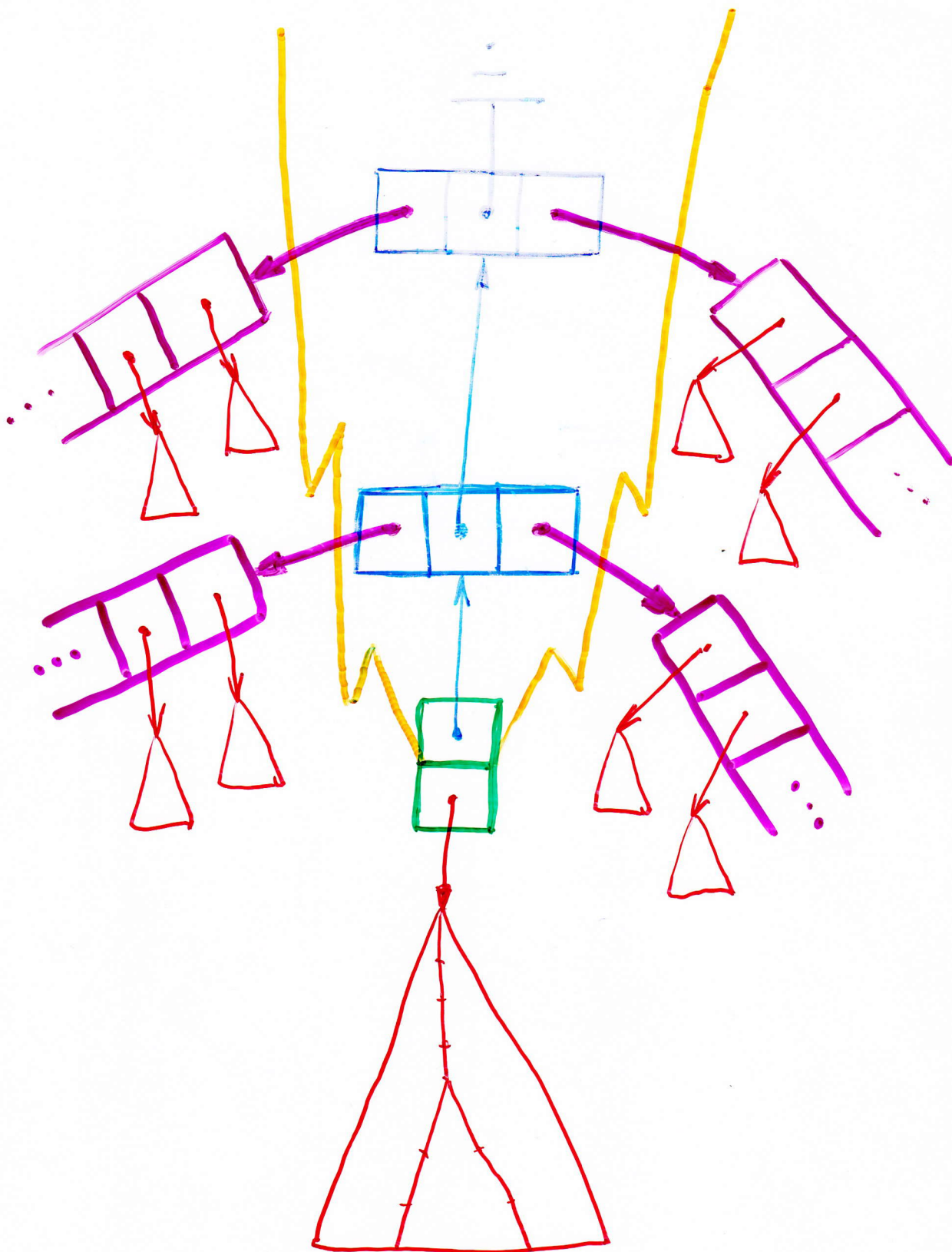
```
type tree =  
  Item of int  
  | Section of tree list;;  
  
type path =  
  Top  
  | Node of path * tree list * tree list;;  
  
type location = Loc of tree * path;;
```

$Node(p, l, r)$  contains the father path  $p$ , its left  $l$  and right  $r$  brother trees. Note: a path has brother trees, uncle trees, great-uncle trees, etc But its father is a path, not a tree like in Mentor's tree processor.



## Updating and Inserting in a zipper

```
let change (Loc(_,p)) t = Loc(t,p);;  
  
let insert_left (Loc(t,p)) t' = match p with  
  Top -> failwith "insert at top"  
  | Node(up,left,right) -> Loc(t',Node(up,left,t::right));;
```





## Navigating in a zipper

```
let left (Loc(t,p)) = match p with
  Top -> failwith "left of top"
  | Node(up,l::left,right) -> Loc(l,Node(up,left,t::right))
  | _ -> failwith "left of first";;

let down (Loc(t,p)) = match t with
  Item(_) -> failwith "down of item"
  | Section(t1::trees) -> Loc(t1,Node(p,[],trees))
  | _ -> failwith "down of empty";;

let up (Loc(t,p)) = match p with
  Top -> failwith "up of top"
  | Node(up,left,right) ->
    Loc(Section(concat (t::right) left),up);;
```

## Scoping and parameterization

( $\lambda$ -calculus issues)

- naming of variables
- non-locality of  $\beta$ -rule
- definitions
- recursion
- approximations
- proof search

## Binding/Scoping/Naming


A perplexing state of affairs.

- $\alpha$ -conversion
- de Bruijn indexes
- combinators
- Miller patterns
- Pfenning HOAS
- Talcott binding structures
- Pollack's meta-theory of LEGO
- Explicit substitutions

## Explicit substitutions

- Abadi Cardelli Curien Lévy
- Hardin Lévy
- Lescanne; Rios; Kamareddine
- Muñoz
- Dowek Hardin Kirchner
- etc

## Sharing

- Dags (UNIX links, TRS, etc)
- $\lambda$ -calculus
  - + Wadsworth
  - + Lévy
  - + Lamping
  - + Gonthier 
- Sharing modulo computing
  - + symbolic link
  - + URL book.ps.gz
  - + BDDs
  - + Sharing substitutions

## Annotations

Annotations are essential. They represent points of view. It should be possible to add new annotations without changing the type of the core structure, and without disturbance for processes unconcerned by this point of view.

OO solution? Methods for copying, moving, etc. Chet's OO-engine for adding judgements to Coq's engine.

There are a number of features to be added to our theory in order to meet long term goals. These include:

- (1) annotations;
- (2) generic tools for structure walking, matching and unification; and
- (3) representation of binding structures in terms of mutable structures.

(C. Talcott)



## Constraints, unification

Constraints are essential for delaying the search for existentials.

- Constrained resolution
- Prolog
- Type reconstruction
- Floating universes
- Linear arithmetic



## $\lambda$ -terms vs Böhm trees

$$\lambda u_1 \cdot \lambda u_2 \cdot ((u_1 u_2) \lambda u_3 \cdot u_3)$$

$$\lambda u_1 u_2 \cdot u_1(u_2, \lambda u_3 \cdot u_3)$$

Head normal forms vs unsolvables

$$\lambda u_1 u_2 \dots u_n \cdot w(T_1, \dots, T_p)$$

Separability: Böhm's theorem.

## Curry-Howard for Sequent Calculus

Proof checking is usually explained on natural deduction formulations, for which (typed)  $\lambda$ -calculus is relevant, by the Curry-Howard isomorphism. However, proof search usually corresponds to a sequent calculus structure. This introduces cumbersome translations between the proof trees associated to tactics computations, and the proof terms stored as  $\lambda$ -terms justifications.

However, it is possible to constrain sequent calculus derivations as Böhm trees. This is actually implicit from Howard, and was exactly identified in H. Herbelin's  $\bar{\lambda}$ -calculus in his thesis "Séquents qu'on calcule" (Paris 7, jan. 95). It uses a "stoup-ed" notion of sequent, like in Girard's LU.

## Cut-free LJT

$$\begin{aligned}
 \rightarrow \text{ right} & : \frac{\Gamma, A; \vdash B}{\Gamma; \vdash A \rightarrow B} \quad \lambda \\
 \text{Contr} & : \frac{\Gamma, A; A \vdash B}{\Gamma, A; \vdash B} \quad \text{Head } f \\
 \rightarrow \text{ left} & : \frac{\Gamma; \vdash A \quad \Gamma; B \vdash C}{\Gamma; A \rightarrow B \vdash C} \quad \text{cons} \\
 \text{Axiom} & : \frac{}{\Gamma; A \vdash A} \quad \text{nil}
 \end{aligned}$$

Thus the term  $(f M_1 \dots M_n)$  is coded as  $(\dots((f M_1) M_2) \dots M_n)$  in  $\lambda$ -calculus, and  $f [M_1; M_2; \dots M_n]$  in  $\bar{\lambda}$ -calculus, the type-free structure underlying LJT. Note that the stoup contains the head variable.

Intros; Apply  $f$ .

## Typing Böhm trees

When we use Böhm trees to represent proofs in some logical framework, these trees have types corresponding to formulas (or more generally judgements) of this framework. But we want not to be forced at construction of the abstract syntax trees representing partial proof attempts to be obliged to enforce the possibly complex type maintenance.

On the other side of the spectrum, we may consider a Böhm tree as just an untyped  $\lambda$ -term, i.e. an element of some domain  $D$  verifying some isomorphism  $D \cong [D \rightarrow D]$ . Consistency in the sense of equational logic is just requiring that  $D$  be non-trivial.



## More on typing Böhm trees

Somewhere in between we may type solvable Böhm trees, of the form:

$$\lambda u_1 u_2 \dots u_n \cdot w(T_1, \dots, T_p)$$

by their *shape*  $(w, n - p)$ , consisting of the pair head  $(w)$  and differential arity  $(n - p)$ . For a closed tree,  $w = u_k$ , and then the head indicates the index of sequentiality  $k$ , whereas the differential arity is a kind of coercion specification. The shape may be read as a type  $D^n \rightarrow D \ k[p]$ , specifying: this term is a functional value in  $D^n \rightarrow D$ , whenever its main argument  $x_k$  is coerced to a functional value in  $D^p \rightarrow D$ . Note that this type is invariant by the  $\eta$  rule, and thus makes sense as a partition of the extensional model  $D_\infty$ .

## Systems of guarded combinators

Definitions. We assume given two disjoint denumerable alphabets of symbols:  $\mathcal{X} = \{X_1, X_2, \dots\}$  is the set of *combinator* symbols,  $\mathcal{U} = \{u_1, u_2, \dots\}$  is the set of *parameter* symbols. Intuitively, combinators name Böhm trees, whereas parameters name bound  $\lambda$ -variables.

We call *Böhm tree presentation* with respect to these two alphabets any denumerable system of equations:  $\mathcal{E} = \{E_1, E_2, \dots\}$ , with

$$E_i : X_i \ u_1 \ u_2 \dots u_{n_i} := u_{k_i} (M_1, \dots, M_{p_i})$$

where  $1 \leq k_i \leq n_i$ ,  $0 \leq p_i$ ,  $X_i \in \mathcal{X}$ , and

for  $\forall j \leq p_i \ M_j = X_{k_{i,j}}(v_1, \dots, v_{l_{i,j}})$

with  $1 \leq k_{i,j} \leq n_i$  and  $\{v_1, \dots, v_{l_{i,j}}\} \subseteq \{u_1, \dots, u_{n_i}\} \subseteq \mathcal{U}$ .

We assume furthermore the system to be *deterministic*, in the sense that every  $X \in \mathcal{X}$  possesses at most one defining equation in  $\mathcal{E}$ . We say that it is *total* when every  $X \in \mathcal{X}$  possesses exactly one defining equation in  $\mathcal{E}$ .

## Examples

We remark that Böhm tree presentations are general enough to represent arbitrary families of finitely generated Böhm trees, which is enough for instance to represent the Böhm trees of any  $\lambda$ -terms. But they permit to do more, in that we may represent dags and looping structures. For instance, the  $\lambda$ -term in normal form  $\lambda u_1 u_2 \cdot (u_1 \lambda v \cdot (v v) \lambda w \cdot w)$  may be presented as  $X$  in the system

$$\begin{array}{lll} X \ u_1 \ u_2 & := & u_1(D, I) \\ D \ v & := & v(I(v)) \\ I \ w & := & w \end{array}$$

with sharing of combinator  $I$ .

Whereas the single equation  $Z \ u := u(Z)$  defines as  $Z$  the infinite tree  $\lambda u_1 \cdot u_1(\lambda u_2 \cdot u_2(\dots))$ .



## Other examples

Another example is the fixpoint combinator  $Y$ , with

$$Y\ f := f(Y(f))$$

Still another example, also denoting an infinite Böhm tree, is  $J$  presented by the system:  $J\ x\ y := x(J(y))$ . It is the Böhm tree of  $\lambda$ -term  $(\mathbf{Y}\ \lambda j\ \lambda x\ \lambda y\ (x\ (j\ y)))$ , for  $\mathbf{Y}$  any fixpoint combinator such as Curry's.

Remark that recursion is natural and more canonical, and that computation is more local than  $\beta$ -reduction. Also definedness is atomic.

## Regular Böhm trees

**Definition.** We call *regular* any finite Böhm tree presentation. Such presentations define Böhm trees which are regular in the sense of admitting only a finite number of distinct subtrees, up to variable renaming.

**Theorem.** It is decidable whether  $\mathcal{E} \vdash M = N$  for any regular  $\mathcal{E}$  and simple expressions  $M$  and  $N$ .

(For appropriate notions  $\vdash$  and simple).

## Other issues

From  $\bar{\lambda}$ -calculus to Böhm trees to Automath to Coq...

Constants. Let or A-T pairs or A-T couples. Other judgments. Structure of the context. Sections, Paragraphs, Modules. Meta-variables. Unification. Constraints.

Control of totality.

Control of unfolding, opacity, abstraction.

Tactics.

## Induction

Should induction be primitive (Martin-Löf) or axiomatised (Isabelle)?

Four properties of inductive types.

- Closed: no junk
- Free: no confusion
- Induction: finitely generated.
- Dependence: elimination by pattern-matching.

Guarded general recursion vs primitive recursion.

How powerful should the framework be?

In other words, how much mathematical knowledge should be internalized in the proof assistant? E.g. equality, induction, substitution, sharing, decision procedures.

The more we add, the less we may share developments.

Extreme view: PRA as the framework.

Reflexion.

Bootstrapping.



## Mathematics modelling

Mathematics is rigorous knowledge representation. It combines computation in concrete structures with abstract reasoning in logical systems.

There are historically 4 generations of computerized mathematics:

- 1 computers
- 2 programming languages
- 3 symbolic computation systems
- 4 proof-checked formal mathematics

Proof assistants potentially open the way to the 4th generation of computerized mathematics modelling. BUT they will not replace well understood more efficient paradigms of the lower levels: floating point computation, imperative programming, verification of finite-state systems by BDD techniques. The smooth integration of the full range of computerized mathematics tools is the great challenge of the coming years.