To arrive where we started: experience of mechanizing binding

Tom Ridge, University of Cambridge

Experience of mechanizing binding

- type soundness for MiniML;
- type soundness for TAPL fragments;
- a verified theorem prover for first-order logic;
- an investigation into transferring results between binding representations using isomorphisms;
- Craig's interpolation theorem;
- various POPLmark solutions;
- generalised term models;
- operational reasoning for Caml programs.

POPLmark solutions include versions using De Bruijn, a version of Nominal/locally nameless, and raw terms

What is this talk about?

- *To arrive where we started . . .*
- About "raw terms" i.e. `Lam "x" (Var "x")`

$$\frac{\Gamma, x' : T1 \vdash [x'/x]t : T2}{\Gamma \vdash \lambda x\ t : T1 \to T2}$$

  (side-conditions $x' \notin \mathsf{fv}\ \lambda x\ t$ and $x' \notin \mathsf{dom}\ \Gamma$)
- I believe they can be competitive with the best of the other approaches.
- I'll try to explain why I think this.
- This talk is not supposed to convince you to use raw terms: *raw terms are not currently competitive*.
- Orthogonal issues: whether to develop a package or a library; strengthened induction schemes; altering formal systems to suit mechanization

Outline of the talk

- Part I. Raw terms are not so bad
- Part II. POPLmark with raw terms, based on . . .
- Part III. . . . a general library
  - a foundation of raw terms, as a universal datatype
  - theory expressed as a library of lemmas about raw terms, including alpha, substitution etc.

Some of this is technically interesting, regardless of what you think about raw terms

Part I. Raw terms are good

Raw terms . . .

“this approach has not proved convenient for formal development”

. . . too raw to digest?

Possible reasons for this view

- Raw terms have significant startup costs (e.g. compared to De Bruijn)
- These costs are incurred for every new mechanization, unless steps are taken to make the lemmas reusable. Typically this was several steps further than people were prepared to go.
- The tools were not as good as they are now
- The subject was not as well understood as it is now

Raw terms are good

- Raw terms are an elementary approach
- Good fit with existing technology and automation
- e.g. simple case analysis principles
- Function definition over terms is straightforward
- Program language implementations and theorem provers often use raw terms; if you want to reason about their implementations, you have to tackle raw terms

Alpha is not so important anyway

- e.g. POPLmark proofs are mostly structural
- e.g. Strong normalization for STLC only needs alpha in one place (see Girard “Logic, Proofs and Types”)
- e.g. Craig’s interpolation theorem does not need any alpha at all (see my Isabelle/HOL formalization)
- e.g. Owen’s formalization of Caml needs no alpha at the term level (although De Bruijn appears at the type level)

N.B. a structural proof is one that doesn’t use alpha-reasoning

Fine control

- Working upto alpha means e.g. $\lambda x\ s = \lambda y\ t$ should not imply $x = y$ and $s = t$.
- But if $\lambda x\ s = \lambda y\ t$ occurs in an informal proof, you might implicitly rename one bound variable to the other, so that $x = y$ and $s = t$.
- Induction principles do not yet support this, so you have to do it manually, which can be a lot of work
- Working “not upto alpha” has the same effect
- e.g. $\lambda x\ s = \lambda y\ t$ implies $x = y$ and $s = t$
- For structural proofs, I want to work “not upto alpha”
- This is supported by raw terms, which in general allow very fine control over all aspects of binding

POPLmark rules, informal

$\boxed{\Gamma \vdash S <: T}$

$$\frac{\Gamma \vdash S <: \mathbf{Top}\ \mathsf{ok}}{\Gamma \vdash S <: \mathbf{Top}} \quad \text{SA\_TOP}$$

$$\frac{\Gamma \vdash X <: X\ \mathsf{ok}}{\Gamma \vdash X <: X} \quad \text{SA\_REFL\_TVAR}$$

$$\frac{\begin{array}{c} X <: U \in \Gamma \\ \Gamma \vdash U <: T \end{array}}{\Gamma \vdash X <: T} \quad \text{SA\_TRANS\_TVAR}$$

$$\frac{\begin{array}{c} \Gamma \vdash T_1 <: S_1 \\ \Gamma \vdash S_2 <: T_2 \end{array}}{\Gamma \vdash S_1 \rightarrow S_2 <: T_1 \rightarrow T_2} \quad \text{SA\_ARROW}$$

$$\frac{\begin{array}{l} \Gamma \vdash T_1 <: S_1 \\ \Gamma, X <: T_1 \vdash S_2 <: T_2 \end{array}}{\Gamma \vdash \forall X{<:}S_1.\, S_2 <: \forall X{<:}T_1.\, T_2} \quad \text{SA\_ALL}$$

POPLmark rules, raw terms

$\boxed{\Gamma \vdash S <: T}$

$$\frac{\Gamma \vdash S <: \mathbf{Top}\ \mathsf{ok}}{\Gamma \vdash S <: \mathbf{Top}}\quad \textsf{SA\_TOP}$$

$$\frac{\Gamma \vdash X <: X\ \mathsf{ok}}{\Gamma \vdash X <: X}\quad \textsf{SA\_REFL\_TVAR}$$

$$\frac{\begin{array}{c} X <: U \in \Gamma \\ \Gamma \vdash U <: T \end{array}}{\Gamma \vdash X <: T}\quad \textsf{SA\_TRANS\_TVAR}$$

$$\frac{\begin{array}{c} \Gamma \vdash T_1 <: S_1 \\ \Gamma \vdash S_2 <: T_2 \end{array}}{\Gamma \vdash S_1 \rightarrow S_2 <: T_1 \rightarrow T_2}\quad \textsf{SA\_ARROW}$$

$$\frac{\begin{array}{l} \Gamma \vdash T_1 <: S_1 \\ \neg(\,Z \textbf{ in fv } S_2 - \{X\}\,) \\ \neg(\,Z \textbf{ in fv } T_2 - \{Y\}\,) \\ \Gamma\,,\,Z <: T_1 \vdash [\,Z\,/\,X\,]\,S_2 <: [\,Z\,/\,Y\,]\,T_2 \end{array}}{\Gamma \vdash \forall X {<:} S_1 .\, S_2 <: \forall Y {<:} T_1 .\, T_2}\quad \textsf{SA\_ALL}$$

Fine control over alpha

$$\frac{\begin{array}{l}\Gamma \vdash T_1 <: S_1 \\ \neg(\, Z \ \textbf{in fv}\ S_2 - \{X\}\,) \\ \neg(\, Z \ \textbf{in fv}\ T_2 - \{\,Y\,\}\,) \\ \Gamma\,,\ Z\ <:\ T_1 \vdash [\,Z\,/\,X\,]\,S_2 <: [\,Z\,/\,Y\,]\,T_2\end{array}}{\Gamma \vdash \forall X {<:} S_1.\, S_2 <: \forall Y {<:} T_1.\, T_2}\quad \text{SA\_ALL}$$

- One of several alternatives
- But with this choice, alpha reasoning is restricted to a single proof that the rules are closed under alpha (and subsequent uses of this fact)

Representatives

- Consider a POPLmark sequent $\Gamma \vdash S <: T$.
- The context Γ binds variables in S and T.
- I prefer to treat context binding the same as term binding
- so it is natural to consider sequents "upto alpha".
- Here is a POPLmark rule:

$$\frac{\begin{array}{c} X <: U \in \Gamma \\ \Gamma \vdash U <: T \end{array}}{\Gamma \vdash X <: T} \quad \text{SA\_TRANS\_TVAR}$$

- To avoid unwanted variable capture for U the variables bound by Γ should be distinct.
- "Γ-bound variables are distinct" is natural with raw terms.

Flexibility

- We know what alpha is.
- We don’t know what systems people will formalize.
- Some approaches bake in alpha;
- some bake in alpha, substitution, and context.
- I want to give people flexibility,
- so if people want to talk about representatives, and bound variables being distinct, I want to support that.
- If people want to work upto alpha in some parts of the proof, and not in others, I want to support that.
- Raw terms seem the best way to support this flexibility.

Part II. POPLmark with raw terms

POPLmark types (in HOL)

```
val _ = Hol_datatype ‘
  Type = T_Top
  | T_Var of typevar
  | T_Fun of Type Type
  | T_Forall of typevar Type Type
‘;
```

Lightly hand edited, due to confusing HOL syntax

POPLmark 1a proofs

```
  (* The interesting case is SA-Trans-TVar with M = X and we have
  G, X<:Q, D |- Q <: N as a subderivation. *)
have ‘SA (G ++ [(INL X, Q)] ++ D) Q N‘; e(tac[]);
  (* Applying the inner induction hypothesis to this subderivation
  yields G, X<:P, D |- Q <: N. *)
have ‘SA (G ++ [(INL X,P)] ++ D) Q N‘; e(tac[]);
  (* Also, applying weakening (Lemma A.2, part 2) to G |- P <: Q
  yields G, X<:P, D |- P <: Q. *)
have ‘G_ok (G ++ [(INL X,P)] ++ D)‘; e(tac[SA_SA_ok,SA_ok_def]);
have ‘SA_ok (G ++ [(INL X,P)] ++ D) P Q‘; e(MATCH_MP_TAC’ SA_ok_weak); e(s
have ‘SA (G ++ [(INL X,P)] ++ D) P Q‘; e(MATCH_MP_TAC’ SA_weak); e(ssimp[]
  (* Now, by part (1) of the outer induction hypothesis (with the same Q),
  we have G, X<:P, D |- P <: N. *)
have ‘SA (G ++ [(INL X,P)] ++ D) P N‘; e(simp [ttrans_def]);
  (* Rule SA-Trans-TVar yields G, X<:P, D |- X <: N ... *)
have ‘SA (G ++ [(INL X,P)] ++ D) (T_Var X) N‘; e(tac[SA_Trans_TVar,MEM,MEM
  (* ... as required. *)
e(tac[]);
```

Proofs can mirror informal proofs extremely closely.

Part III. A general library

Why a library of lemmas?

We develop a library of lemmas, rather than a package which automatically produces lemmas on demand.

- Largely an orthogonal issue to representation
- A library means you can understand things, and change things, easily
- A package means you have to know about theorem prover internals
- A library means people can pick and choose which bits they want to use
- A library is felt to be more flexible than a package

Universal type of terms

```
tm =  Var of 'var
| Node of 'val (tm list)
| BIND of 'var tm
```

N.B. `'val,'var` are free type variables e.g. for lambda-calculus `'var` might be `string`

POPLmark instantiation

```
val =
    NT_Var
  | NT_Top
  | NT_Fun
  | NT_Forall

  | Nt_Var
  | Nt_Lam
  | Nt_App
  | Nt_TLam
  | Nt_TApp

  | N_vdash
  | N_G_cons
```

POPLmark instantiation

```
  (T_tm (T_Top) = Node NT_Top [])
/\(T_tm (T_Var X) = Var (INL X))
/\(T_tm (T_Fun U V) = Node NT_Fun [T_tm U; T_tm V])
/\(T_tm (T_Forall X U V) = Node NT_Forall [T_tm U; BIND (INL X) (T_tm V)]

/\(Vdash_tm U V = Node N_vdash [U;V])

/\(G_tm ([]) U = U)
/\(G_tm ((xX,V)::xs) U = Node N_G_cons [V; BIND xX (G_tm xs U)])

/\(SA_tm G U V = G_tm G (Vdash_tm U V))

/\(SA_ALPHA G S T G' S' T' =
    DISTINCT (DOM G) /\ DISTINCT (DOM G') /\
    let (G,S,T) = (MAP (\(xX,U).(xX,T_tm U)) G,T_tm S,T_tm T) in
    let (G',S',T') = (MAP (\(xX,U).(xX,T_tm U)) G',T_tm S',T_tm T') in
    let (GST,GST') = (SA_tm G S T,SA_tm G' S' T') in
    closed GST /\ closed GST' /\
    alpha GST GST')
```

N.B. `T_tm` etc. should be defined automatically

Subtyping closed under SA_ALPHA

```
SA G S T
==> SA_ALPHA G S T G' S' T'
==> SA G' S' T'
```

See bindingTalk20070810.{pdf,ps} on my webpage for proof.

Using the theory

Using the theory means using lemmas.

```
have ‘G, Z<:T1 |- [Z/X]S2 <: [Z/Y]Q2‘; ...
have ‘SA_ALPHA
  (G, Z<:T1 |- [Z/X]S2 <: [Z/Y]Q2)
  (G, Z’<:T1 |- [Z’/X]S2 <: [Z’/Y]Q2)
‘; e(tac[SA_ALPHA_lemmas);
have ‘G, Z’<:T1 |- [Z’/X]S2 <: [Z’/Y]Q2‘;
 e(tac[SA_ALPHA]);
```

- Relation SA_ALPHA: two sequents are alpha equivalent (and context bound vars are distinct)!
- Lemma SA_ALPHA: SA is closed under SA_ALPHA
- Very small interface between general theory and particular mechanization

Conclusion

- Some proofs work best with particular systems.
- I’ve given some advantages of raw terms, but they are not conclusive, and they may not apply to your proof.
- But POPLmark proofs with raw terms look very nice.
- Library, with universal datatype of terms, and supporting lemmas, is flexible and allows users to pick and choose.
- Treating sequent binding the same as term binding allows theory to be used more widely.
- This is also technically quite difficult (I haven’t seen other concrete approaches do this)
- so perhaps raw terms are digestible after all!