

how to implement a typechecker?

Freek Wiedijk

Radboud University Nijmegen

CHIT/CHAT workshop

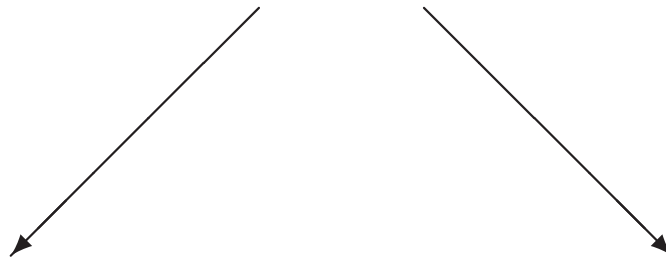
Radboud University Nijmegen

2006 12 19, 11:05

looking for a project

two kinds of applications

proof assistants



formalize mathematics

fundamental theorem of algebra

C-CoRN

Mizar library

correctness of software

?

when will proof assistants finally become mature?

- mature programming language:

compiler



compiles own source code

- mature mathematical language:

proof assistant



should verify its own correctness

(the *actual* source code)

Gödel's second

but, system cannot prove 'system is consistent'!

but system **can** prove:

'system implements this set of logic rules'

'this trivial axiom implies that the system is consistent'



an inaccessible cardinal exists

```
new_type("I",0);;
let I_AXIOM = new_axiom
  'UNIV:ind_model->bool <_c UNIV:I->bool /\
  (!s:A->bool. s <_c UNIV:I->bool ==> {t | t SUBSET s} <_c UNIV:I->bool)';;
```

what I would like to do

- fully verified typechecker of $\mathbf{LF} \approx \lambda P \approx \text{Automath}$
- **real** program
 - including input/output
parser/pretty-printer
 - meaningful error messages
- not too much slower than fast typechecker written in C

how to do this?

this talk

- three approaches to verification of software correctness
- **state of the art** in self-verification of proof assistants
- a few small & aborted **experiments**

approach one: **write the program in the formalization**

programming in type theory

formalization \longrightarrow program

most impressive examples

- **four color theorem programs** by Georges Gonthier
- **verified C compiler** by Xavier Leroy

$\text{Clight} \longrightarrow \text{Cminor} \xrightarrow{\text{Compcert}} \text{PowerPC code}$

state of the art one: Coc

- Bruno Barras & Benjamin Werner
Coq in Coq

no definitions = less than rudimentary

It seems reasonable to continue the effort, especially by extending the encoded formalism and closing the gap with formalism of Coq itself. We consider the following extensions:

- Definitions: this would require the formalization of δ -reduction, **which should be easy.**

...

extracted code: 167 lines of purely functional ML

other executable formalizations of type theory

- Randy Pollack & James McKinna

Some Lambda Calculus and Type Theory Formalized

‘Lego in Lego’

- Gilles Dowek & Bob Boyer

Toward Checking Proof-Checkers

‘Coq in ACL2’

- Nils Danielsson

A Partial Formalisation of a Dependently Typed Language as an Inductive-Recursive Family

‘Agda in Agda’

all of them: **no definitions**

experiment one: boiling down real systems

- **Coq 8.0**

kernel ('`kernel`'): ~ 8000 lines of non-functional ML
state, exceptions

LF part: ~ 1400 lines

- **Twelf**

real kernel ('`lambda`'): ~ 7600 lines of non-functional ML

show kernel ('`typecheck`'): 468 lines of non-functional ML
reuses code from real kernel

- **Agda**

also: both real & show kernels

- **Epigram**

experiment two: LCF-style kernel

abstract datatypes that guarantee type-correctness

purely functional ML

supports definitions = more than rudimentary

255 lines

no exceptions, only algebraic datatypes

datatypes:

	<i>not typechecked</i>	<i>typechecked</i>	
<i>terms</i>	●	○	← incompatible contexts?
<i>environments</i>		●	

experiment three: specification versus implementation in Coq

- **straightforward formalization of PTS rules in Coq**

totally trivial rendering of informal version

de Bruijn variables

108 lines of code

```
Inductive derivable : judgment -> Prop :=
| axiom : forall c s,
  is_axiom c s ->
  derivable (infer empty c s)
| start : forall Gamma A (s : sort),
  derivable (infer Gamma A s) ->
  derivable (infer (extend Gamma A) (var 0) (lift A))
| weaken : forall Gamma A (s : sort) B C,
  derivable (infer Gamma A s) ->
  derivable (infer Gamma B C) ->
  derivable (infer (extend Gamma A) (lift B) (lift C))
...

```

experiment three (continued)

- **straightforward typechecker in ML**

122 lines of code

```
type_of : context -> term -> term
```

27 lines of code

- **porting typechecker to Coq**

```
type_of : nat -> context' -> term -> option (option (term * option sort))
```

(context': annotate types with their sort)

61 lines of code

- **proving typechecker correct in Coq**

stopped doing this

PTS theory has been formalized too many times already

did not really see how to fit this into a real program

what I learned

programming in type theory is **very unpleasant**

(even **without** dependent types to make your life interesting)

- **non-termination**

special syntax for Bove-Capretta method?

- **exceptions**

special syntax for monads?

approach two: **formalize a model of the program**

state of the art two: HOL Light

beautiful HOL reimplementations by John Harrison

actual kernel: **438 lines** of non-functional ML

correctness proof of the kernel: **~ 2600 lines** of HOL

```
HOL_IS_SOUND      '!as1 p. as1 |- p ==> as1 |= p'
```

```
HOL_IS_CONSISTENT 'p. p has_type Bool /\ ~([] |- p)'
```

```
let sequent = new_definition
```

```
  'asms |= p <=> ALL (\a. a has_type Bool) (CONS p asms) /\  
    !sigma tau. type_valuation tau /\  
      term_valuation tau sigma /\  
        ALL (\a. semantics sigma tau a = true) asms  
      ==> (semantics sigma tau p = true)';;
```

John did not prove his **actual** code correct

```
let vsubst =
  let rec vsubst ilist tm =
    match tm with
    | Var(_,_) -> rev_assoc tm ilist tm
    | Const(_,_) -> tm
    | Comb(s,t) -> let s' = vsubst ilist s and t' = vsubst ilist t in
                     if s' == s & t' == t then tm else Comb(s',t')
    | Abs(v,s) -> let ilist' = filter (fun (t,x) -> x <> v) ilist in
                  if ilist' = [] then tm else
                  if exists (fun (t,x) -> vfree_in v t & vfree_in x s) ilist'
                  then
                    let fvs = frees s and fvt = map snd ilist' in
                    let fvt' = freesl (map (fun x -> rev_assoc x ilist')
                                           (intersect fvs fvt)) in
                    let v' = variant (union (subtract fvs fvt) fvt') v in
                    Abs(v',vsubst ((v',v)::ilist') s)
                  else Abs(v,vsubst ilist' s) in
  fun theta ->
    if theta = [] then (fun tm -> tm) else
    if forall (fun (t,x) -> type_of t = snd(dest_var x)) theta
    then vsubst theta else failwith "vsubst: Bad substitution list";;
```

John did not prove his **actual** code correct

```
let VSUBST = define
  '(VSUBST ilist (Var x ty) = REV ASSOCD (Var x ty) ilist (Var x ty)) /\
  (VSUBST ilist (Equal ty) = Equal ty) /\
  (VSUBST ilist (Select ty) = Select ty) /\
  (VSUBST ilist (Comb s t) = Comb (VSUBST ilist s) (VSUBST ilist t)) /\
  (VSUBST ilist (Abs x ty t) =
    let ilist' = FILTER (\(s',s). ~(s = Var x ty)) ilist in
    let t' = VSUBST ilist' t in
    if EX (\(s',s). VFREE_IN (Var x ty) s' /\ VFREE_IN s t) ilist'
    then let z = VARIANT t' x ty in
      let ilist'' = CONS (Var z ty, Var x ty) ilist' in
      Abs z ty (VSUBST ilist'' t)
    else Abs x ty t)';;
```

grand unified mathematical/programming language?

HOL

mathematical language: HOL

programming language: ML

but **Coq** is worse . . .

mathematical language: Coq

scripting language: Ltac

programming language: ML

approach three: **extract proof obligations from the program**

'Hoare logic'

program \longrightarrow formalization

implementation of Hoare logic by Jean-Christophe Filliâtre

- **Why:** simple first order programming language
state, exceptions, references (but no non-termination)
- **Caduceus:** (almost) full C
built on top of Why

experiment four: typechecker in C

different ways to represent variable binding:

- named variables
- de Bruijn indices ('nameless dummies')
- higher order abstract syntax
- a graph represented by pointers in memory

342 lines of 'functional' C

no definitions ... because I was thinking of my PTS formalization

not sure my usage of C is supported by Caduceus

```
typedef struct term *term;  
term t = (term) malloc(sizeof(struct term));
```

please help me

how to prove a realistic typechecker correct?

how would **you** do it?

- **just push one of the approaches from this talk through**
but then: how to connect to the real world?
- **pursue some other approach that I did not think about**

tell me about it!