Why dependent types matter

Thorsten Altenkirch

School of Computer Science
University of Nottingham

February 25, 2008
What are dependent types?

- Data and programs may occur within types.
- Type checking requires to carry out symbolic computations.
- *Full blown dependent types*: Full language can be used within types.
- *Phase sensitive languages*: Type level language different from object level language.
- See Conor’s famous slides on DTP and social order.
  
  [http://strictlypositive.org/a-case/](http://strictlypositive.org/a-case/)

**Question**

Does this cover everything we want to call DTP?
Or is it maybe too liberal?
Dependently typed programs

- Vectors instead of lists
- Decidable instead of Bool
- Tagless interpreter and type checker
- Structural recursive unification
- Verified sort
- Generic programming with universes

Question

What is your favorite DTP pearl?
Dependently typed languages

**LF inspired**
DML  ML indexed by natural numbers.
ATS  extending and generalizing DML
Delfin uses HOAS

**FP inspired**
Haskell  Multiparameter type classes, GADTs
Ωmega  Rationalizing use of GADTs

**TT inspired**
CIC  Coq’s language
Epigram  influenced by LEGO, inspired by ALF
Agda  inspired by ALF and Cayenne
Cayenne  influenced by ALF, based on LML.

**Question**
Is this an accurate picture? What is missing?
Hindley-Milner alignments

Data : Types
Explicit : Implicit
Runtime : Compiletime
Partial : Total

Question

Does this alignment work for DTP?
If not, what are the alternatives?
Partial vs total

- Partiality at the type level $\iff$ type checking undecidable.
- Does this matter?
- Partiality forces us to run proofs at runtime.
- This does matter!
- Model partiality as an effect?
- Phase-sensitive or full-blown + phase polymorphism? c.f. Edwin Brady’s PhD.

**Design alternatives**

- Phase-sensitive: partial runtime, total compile time
- Partial core + termination checker
- Total core
Dependent pattern matching

- Different programs typable in different branches!
- Inductive families or recursive only?
- Instantiation of indizes.
- Impossible branches.
- Equational inference, automatic or explicit?
- Pattern matching as primitive?

**Question**

Which are the viable alternatives in this design space?
Elaboration

- Implicit parameters, not just types.
- Hidden proofs, automatisation of reasoning?
- User extensible elaboration, library + type inference?

Question

What are the design principles here?
Reusability?

- More precise types $\implies$ less reusability?
- Conversion is too intensional.
- Cannot substitute Peano numbers by binary numbers?
- Loss of modularity!!

**Question**

How can we have the DTP cake and eat (reuse) it?
Dependent types and the real world

- EffTT workshop in Tallinn in December
- Use Monads (like Haskell)?
- IO should not be opaque.
- *Hoare Type Theory* by Greg Morrisett and Aleksandar Nanevsky.
- Functional specifications of IO (Wouter Swierstra and myself)

**Question**

What is the best way to integrate effects into DTP?
Killer Apps?

- Proof carrying code
- Program correctness, pay as you go
- Domain specific libraries with rich type disciplines.

Question

Other proposals for killer apps?
Can DTP affect the software industry?