

TinyHM: Hindley-Milner type inference

Peano numbers and step by step guide

Tomas Petricek, Charles University

✉ tomas@tomasp.net

✈ [@tomasp.net](https://tomasp.net)

🌐 <https://tomasp.net>

🌐 <https://d3s.mff.cuni.cz/teaching/nprg077>



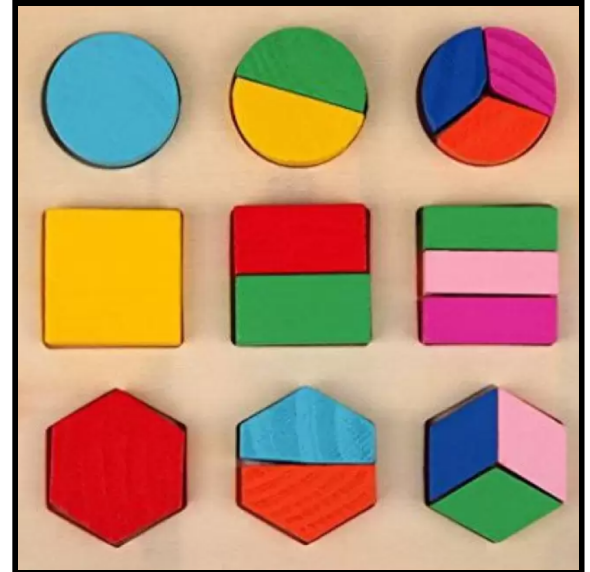
Constraint solver structure

Simplest possible example

- Peano numbers: **Zero**, **Succ(x)**
- Equality constraints with variables
- e.g. **Succ(x) = Succ(Succ(Zero))**

Creating a solver

- Discharge matching constraints
- Fail on mismatching constraints
- Generate more for matching nested
- Needs to handle substitutions...



Demo

Solving numerical constraints

Remaining work

Substitution (#1)

Replace variable in remaining constraints

Substitution (#2)

Apply substitutions to assigned type

Occurs check (#3)

Check for unsolvable constraints

```
let rec solve constraints =  
  match constraints with  
  | [] -> []  
  | (Zero, Zero)::cs -> solve cs  
  | (Succ n1, Succ n2)::cs -> solve ((n1, n2)::cs)  
  | (Zero, Succ _)::_ | (Succ _, Zero)::_ ->  
    failwith "cannot be unified"  
  | (n, Variable v)::cs | (Variable v, n)::cs ->  
    let subs = solve cs  
    (v, n)::subs
```

#1 apply all 'subs' to 'n'

#2 substitute 'n' for 'v' in remaining constraints

#3 check that 'v' does not appear in 'n'

Demo

Substitutions and occurs check

TinyHM

Code structure

```
(* All possible types you may
   support: type variables,
   primitives and composed *)
type Type =
  | TyVariable of string
  | TyBool
  | TyUnit
  | TyNumber
  | TyFunction of Type * Type
  | TyTuple of Type * Type
  | TyUnion of Type * Type
  | TyList of Type
  | TyForall of string * Type

(* Types of known variables *)
type TypingContext =
  Map<string, Type>
```

Types supported

Type variables

For constraint solving!

Primitive types

Match/mismatch

Composed types

Generate one or two
new constraints

Polymorphic type

Forall (bonus)

```
(* Given a list of
   constraints, produce a
   list of substitutions *)
val solve :
  list<Type * Type>
-> list<string * Type>

(* Given a typing context
   (known variables) and
   expression, return the type
   of the expression and
   list of constraints *)
val generate :
  TypingContext
-> Expression
-> Type * list<Type * Type>
```

Type inference operations

Constraint solving

Takes constraints

Produces substitution

Constraint generating

Takes an expression

Produces constraints

Also check variables

Lab overview

Tiny Hindley-Milner step-by-step

TinyHM - Basic tasks

1. Complete the simple numerical constraint solver
Add the two missing substitutions to make it work!
2. Solving type constraints with numbers and Booleans
Follow the same structure, but now for type constraints...
3. Type inference for binary operators and conditionals
Add constraint generation for a subset of TinyML
4. Supporting more TinyML expressions
Add let, functions, application and occurs check
5. Adding simple data types
Constraint generation for tuples

TinyHM - Bonus & super tasks

1. Supporting more TinyML data types
Add type checking for discriminated unions
2. Type inference for lists - poor method
Add recursion & units and try this on list code!
3. Adding proper support for generic lists
New type, but without explicit type declarations
4. Inferring polymorphic code for let bindings
Implementing proper Hindley-Milner let-polymorphism
5. Exploring pathological cases
Did you know HM has DEXPTIME complexity?

Lessons learned

Tiny Hindley-Milner type inference

-  A remarkable quality of ML language(s)
-  Types as a communication mechanism
-  Nice introduction to constraint solving
-  Much more can be done with this idea...

