

TinyHM: Hindley-Milner type inference

How type inference in ML works

Tomas Petricek, Charles University

✉ tomas@tomasp.net

🦋 [@tomasp.net](https://twitter.com/tomasp.net)

🌐 <https://tomasp.net>

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

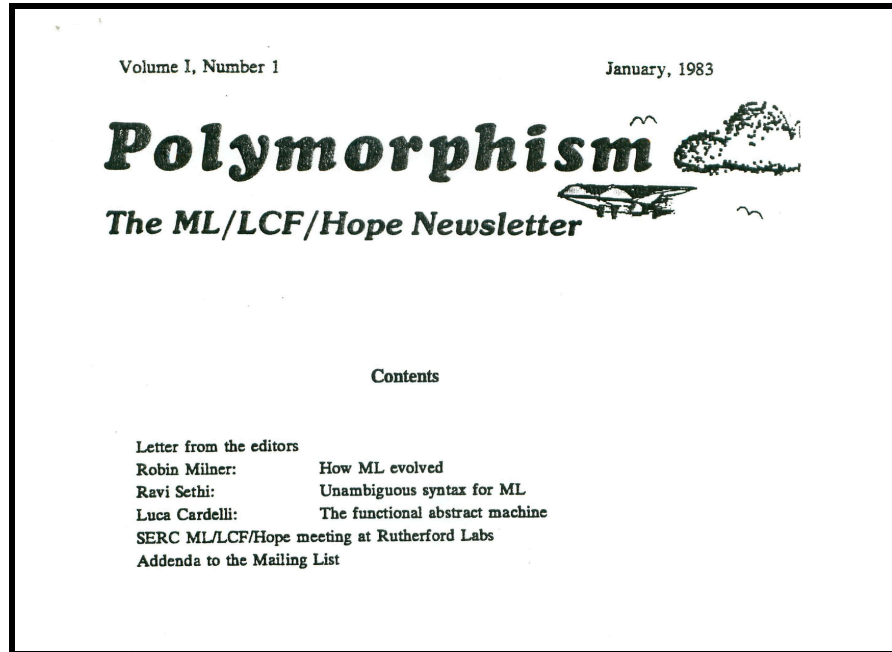


Not a programming system!?

 An important part of the ML experience
Makes ML practical and OCaml efficient

 Learn some subtle aspects of F# type inference
Some discovered late through proofs and errors

 Good example of constraint solving...
Important technique, used in Prolog & elsewhere



Origins of ML

LCF theorem prover

ML used for writing
meta-programs to
generate proofs

Types used to ensure
the validity of proofs

Hindley-Milner

A brief history of type inference

- 📖 Hindley (1969) for Combinatory Logic
- ⌕ Milner (1978) for ML with polymorphism
- Damas (1985) with formal analysis and proofs
- 🚀 Since then - type classes, other extensions

Demo Coeffects playground

Constraint solver
code on GitHub

Choose a coeffect language from the dropdown and load a sample snippet to get started.

Dataflow language (flat) Open sample

```
fun x y ->
  let avg2 = fun y -> (y + prev y) / 2 in
  avg2 x + prev (avg2 y)
```

Check snippet

In the formatted code below, you can see types of variables in a tooltip. Curried functions with multiple parameters and function defined using `let` are expanded.

```
fun x -> fun y ->
  let avg2 = fun y -> (y + prev y) / 2 in
  avg2 x + prev (avg2 y)
```

Now explore the typing derivation. Click on the judgements in the assumptions to navigate through the typing derivation. Compare flat and structural dataflow typing for the same program!

$$\frac{\frac{(\dots)}{x:\text{num}, y:\text{num} @ 1 \vdash (y + \text{prev } y) / 2 : \text{num}}}{x:\text{num} @ 1 \vdash \text{fun } y \rightarrow \dots : \text{num} \rightarrow \text{num}} \quad 1$$

(...)

ML type inference

How does F# figure out the types?

Demo

Basic type inference in F#

How F# type inference works

Constraint-based

- Collect & solve constraints
- No annotations needed for ML!

```
let twice f x = f (f x)  
val twice: f: ('a -> 'a) -> x: 'a -> 'a  
Full name: Inference.twice
```

Let polymorphism

- Infer generic type of let-bound functions

Limitations in ML and F#

- Value restriction for generic values
- Harder to deal with .NET objects

Demo

Type inference limitations in F#