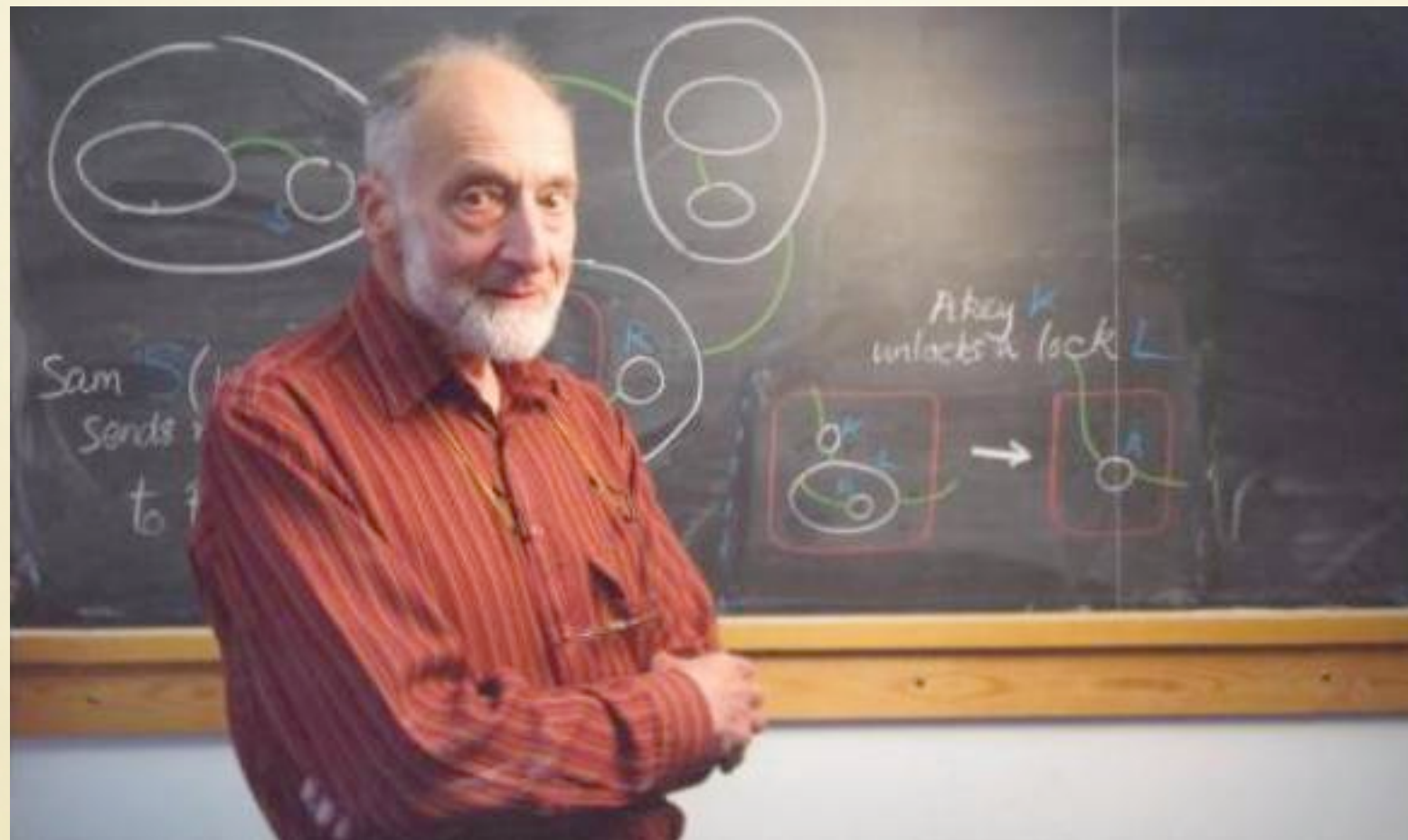


ROBIN MILNER, FRS

1934-2010



PROGRAMMING LANGUAGE DESIGN AND SEMANTICS

Quite simply, the versatility of computers is exactly equal to the versatility of the languages by which we prescribe their behaviour, and this appears to be unbounded.

1997 University of Bologna

THE INFLUENCE OF ROBIN'S WORK

What impresses me most about Robin's contributions to PL's is not just their **influence**, but the extent to which his ideas were born **fully formed**.

TYPED FUNCTIONAL PROGRAMMING

We believe that ML contain[s] features worthy of serious consideration; these are the escape mechanism, and the polymorphic type discipline ..., and also the attempt to make programming with functions-including those of higher type-as easy and natural as possible.

Edinburgh LCF 1979

CLASSIC ML

- **Polymorphic type inference.**
 - Crucial for concision and convenience.
 - Minimizes bureaucracy of types.
- Escape mechanism = **exceptions.**
 - Supports backtracking proof search.

CLASSIC ML

- **Abstract Types.**
 - Enforcement of representation invariants.
 - Confines trusted computing base in provers.
- **Higher-order functions.**
 - Crucial for tactic-based interactive provers.
 - Mathematically natural.

POLYMORPHIC TYPE INFERENCE

- **Principal Typing Theorem**
 - If $\vdash e : \tau$, then there is **principal type scheme** σ such that $\vdash e : \sigma$ and $\sigma \geq \tau$.
 - The type scheme σ may be found by first-order **unification**.
- One of the most important and influential theorems in the theory of PL's!

POLYMORPHIC TYPE INFERENCE

- Principal type scheme for `map` function:
 - $(* \rightarrow **) \rightarrow * \text{ list} \rightarrow ** \text{ list}$
 - $\forall \alpha, \beta (\alpha \rightarrow \beta) \rightarrow \alpha \text{ list} \rightarrow \beta \text{ list}$
- Specializes to instances such as
 - $(\text{int} \rightarrow \text{string}) \rightarrow \text{int list} \rightarrow \text{string list}$

POLYMORPHIC TYPE INFERENCE

- Milner's Theorem is both an **inspiration** and a **torment** to language designers.
 - Extremely hard to do significantly better!
 - Many useful extensions, including **record** and **object** types, **type classes**, and **modules**.
- Shown to be complete for DEXPTIME, yet is amazingly practical and useful for real code.

ABSTRACT TYPES

- ML featured a rigorous **abstract type** mechanism.
 - **abs/rep** mediate between an abstract type and its representation
 - ensures representation independence
 - tightly connected with polymorphism
- Bounds **trust assumptions** for provers.

ABSTRACT TYPES

```
absrectype thm = form list # form
with hyps t = h
    where h,c = rept hm t
and concl t = ...
and truthI = absthm (nil, TRUE)
and ...
```

The representation of `thm` is hidden from clients so that the *only* values of type `thm` are theorems.

INFLUENCE OF ML

- ML gave rise to Hope, Miranda, Haskell, Caml, and Standard ML (and inspired many more).
 - Pattern matching, modules/type classes, exceptions, mutation, records, objects/classes.
 - The standard against which all functional language designs are judged.
- Numerous papers were inspired by ML.
 - Appel: “POPL is the Principles of ML”

ML MODULES

- **Modules** generalize abstract types and type classes.
 - Components are called **structures**.
 - Interfaces are called **signatures**.
 - Functions are called **functors**.
- Apply functional programming “in the small” to programs “in the large”.

STANDARD ML

```
signature THEOREM = sig
  type thm
  val truthI : thm
  val andI : thm -> thm -> thm
  val andEL : thm -> thm
  ...
end

structure Thm :> THEOREM = ...
```

LANGUAGE DEFINITION

The aim of a language definition is ... to establish a theory of semantic objects upon which the understanding of particular programs may rest.

The Definition of Standard ML 1997

LANGUAGE DEFINITION

- What does it mean for a programming language to **exist**?
 - Precise definitions support rigorous theory.
 - Precise definitions support implementation by ensuring compatibility.
- The Definition of Standard ML remains the best example of rigorous language definition at scale.

LANGUAGE DEFINITION

- **Operational**, rather than **denotational**.
 - Contrary to conventional wisdom (at the time).
 - Readable, scalable, applicable.
- Symmetry between **static** and **dynamic** semantics.
 - Safety as coherence of statics and dynamics.

LANGUAGE DEFINITION

- Static semantics: **elaboration** of programs.
 - $\vdash \textit{program} \Rightarrow \textit{type}$
- Dynamic semantics: **evaluation** of programs.
 - $\vdash \textit{program} \Rightarrow \textit{result}$
- The *result* can be either an answer or **wrong**.

TYPE SAFETY

Theorem: Well-typed programs do not go wrong.

If $\vdash \textit{program} \Rightarrow \textit{type}$ and $\vdash \textit{program} \Rightarrow \textit{result}$, then *result* is a value of *type*, and hence cannot be **wrong**.

States the **coherence** of the static and dynamic semantics. First crisp formulation of “safety”.

FULL ABSTRACTION

- Robin formulated the **full abstraction** problem for language semantics:
 - Find a **compositional** interpretation of a language such that **equality of meaning** coincides with **operational (behavioral) equivalence**.
- Showed that there is a **unique** fully abstract model by an ingenious construction.

