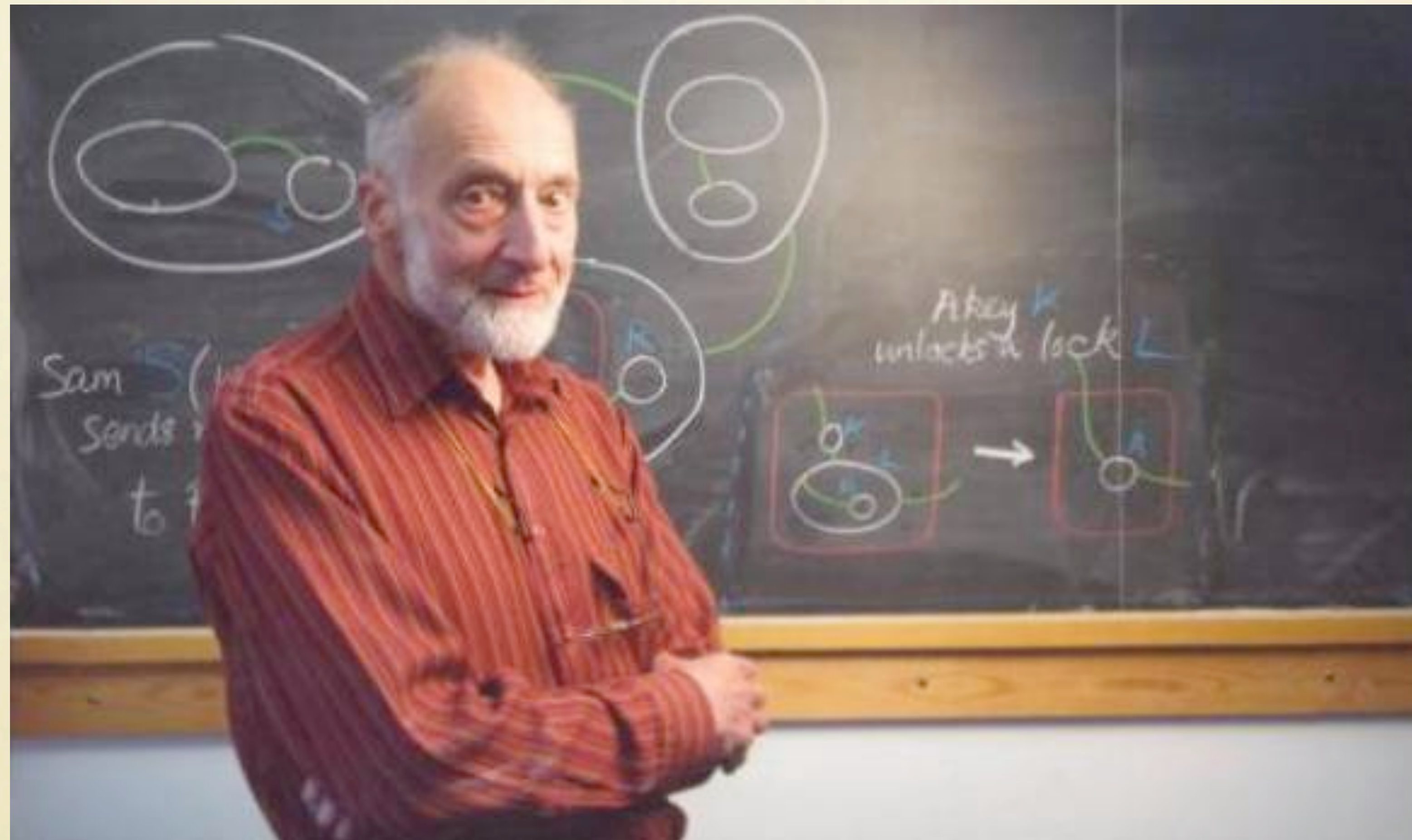
Robin Milner
1934-2010



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

What Robin Taught Me

The essential and fruitful interplay between **theory** and **practice** for Computer Science.

- Building systems inspires and informs theory.
- Mathematical models provide foundations for systems.

There is a **science** of computing, and we should settle for nothing less.

What Robin Taught Me

The importance of **formalisms** for isolating and analyzing models of computation.

- Programming languages, interaction calculi, semantics and logics of programs.
- Analytic techniques for understanding their properties.

The essentially **linguistic** nature of computation.

Robin's Influence

What impresses me most is how many of his ideas were born fully formed, and have survived essentially intact after decades of work.

What impresses me even more is how many of his ideas inspired even greater ideas that are being carried forward today.

Some Themes

Mechanized Reasoning

- Interactive theorem proving
- Concurrency workbench

Typed Functional Programming

- Polymorphic type inference
- Type safety as a design principle

Some Themes

Models of **concurrency, communication,** and **interaction**.

- What is a process? When are processes equal?
- Concepts of **space** and **motion** in computing.

CCS, pi-calculus, action calculi, ambients, bigraphs, stochastic processes, ...

Mechanizing Proof

- **Interactive**, not automatic, theorem proving.
- How to **trust** a mechanized proof?
- Goal-directed proof using **tactics** and **tacticals**.
- Library of formal **theories** built atop one another.

Mechanizing Proof

Use **abstract types** to ensure soundness.

- Adequacy: a value of type `thm` is a valid proof.
- Requires polymorphism, anticipates theory of parametricity.

- Confines **trust** to the implementation of the abstraction.

Mechanizing Proof

Goal-directed search using **tactics** and **tacticals**.

- Exploits higher-order functions.
- Supports decomposition of goal into subgoals.

- Soundness is ensured by a **validation** that transforms proofs of subgoals into proof of goal.
  - Abstraction ensures validity of the validation!

Mechanizing Proof

```
type tactic =
  goal -> (goal list # validation)

type validation =
  thm list -> thm

type tactical =
  tactic list -> tactic

(REPEAT resolve) THEN simplify
```

Mechanizing Proof

- Most of the major proof development systems are based on essentially the **same** ideas!
  - Isabelle, HOL, HOL-Light, NuPRL, Coq
  - (With many improvements and elaborations.)
- Ideas have **scaled** to become a serious tool for the practicing mathematician.
  - 4CT (Gonthier); certifying compilers (Leroy)

Mechanizing Proof

- HOL-Light:

```
e(CONV_TAC(REWRITE_CONV[LE_ANTISYM]) THEN SIMP_TAC[] THEN
ONCE_REWRITE_TAC[EQ_SYM_EQ] THEN DISCH_TAC THEN
ASM_REWRITE_TAC[] THEN CONV_TAC ARITH_RULE);;
```

- NuPRL:

```
∀T:Type ∀as:T List. null(as) ⇔ as = nil By:
UnivCD THEN Analyze -1 THEN AbReduce 0 THEN RW bool_to_pr
opC 0
```

Mechanizing Proof

- Coq (Voevodsky):

```
Definition eqweqmap (T1:Type) (T2:Type) : (paths Type T1
T2) -> (weq T1 T2).

Proof. intros. induction X. apply idweq. Defined.
```

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

- Escape mechanism = **exceptions**.
    - Ideal for backtracking proof search.
- Polymorphism and data abstraction.
    - Essential for limiting trust.
- Higher-order functions.
    - Crucial for writing tactics and tacticals.

Polymorphic Type Inference

- **Polymorphism**: code re-use at **multiple** types.
  - `reverse : * list -> * list`
- **Type inference**: expressions have a **principal** type scheme.
  - If $\Gamma \vdash$ e : τ, then there is type **scheme** σ such that $\Gamma \vdash$ e : σ and τ is an **instance** of σ.
  - Reduces to **unification** of type expressions.

ML Connected Logic With Programming

- As a **meta-language** for LCF and other logics.
  - Proofs are values of an abstract type.
- As an example of the **propositions-as-types** correspondence.
  - “This is how we know that [ML] was given to us by God.” (P. Wadler)

Typed Functional Programming

ML is both an **inspiration** and a **torment** for language designers.

- Very hard to improve upon, especially type inference.
- Directly inspired Hope, Caml, Standard ML, Haskell, F#, and many others.

Two Significant Advances

Haskell-style **type classes**.

- $\forall\, \alpha\ \texttt{Eq}(\alpha) \Rightarrow \alpha\ \texttt{*}\ \texttt{List}(\alpha) \rightarrow \texttt{Bool}$
- `instance Eq (Int) = IntEq`

Polymorphic instantiation finds (unique) instance for a given type.

Two Significant advances

ML-style **module** systems.

- Signatures = types of structures.
- Structures = types with operations.
- Functors = transformations on structures.

Generalizes and **uniformizes** abstract types, type classes, generics, separate compilation.

Standard ML

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

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

Two Significant Advances

Principal Signature Theorem: If $\Gamma \vdash$ M : Π, then there exists Π_0 such that $\Gamma \vdash$ M : Π_0 and Π is more constrained than Π_0.

Constraints arise from **coherence** requirements among components of a module complex.

Two Significant Advances

```
signature INTERP = sig
 structure Lexer : LEXER
 structure Parser : PARSER
 sharing type Lexer.id = Parser.id
 ...
end
```

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

Operational, rather than denotational, methods.

- Directly implementable (purely syntactic).
- Supports lightweight mechanization.

Symmetric treatment of **statics** and **dynamics**.

- An expression **elaborates** to a type and **evaluates** to a value.

Type Safety

Theorem: Well-typed programs do not go wrong.

- **Progress**: well-typed programs have well-defined outcomes.
- **Preservation**: execution preserves typing.

But how to verify this for a full-scale language?

Mechanizing Language Definition

So the language-definer has to develop a small <u>theory</u> of his meanings, in the same way that a mathematician develops a theory.

The Definition of Standard ML 1997

Mechanizing Language Definitions

Building such a theory for a full-scale language **requires** mechanical assistance.

- Hundreds of cases to consider.
- Mostly routine inductions, punctuated by occasional subtleties (and mistakes).

Interactive provers to the rescue!

Mechanizing Language Definitions

Twelf is ideal for language metatheory!

```
tp : type.
tm : type.

of : tm -> tp -> type.
step : tm -> tm -> type.
val : tm -> type.
```

Mechanizing Language Definition

Preservation Theorem

```
pres:
 of E A -> step E E' -> of E' A -> type.
```

Progress Theorem

```
prog:
 of E A -> val_or_step E -> type.
```

Mechanizing Language Definitions

Mechanized proof of type safety for Standard ML.

- Elaboration into an **internal** language.
- Statics and dynamics given to internal language.
- Proof of safety for IL and of elaboration.

About 30K lines of Twelf code representing about two man-years of effort.

Mechanizing Language Definitions

Formalization and mechanization of language definition leads to **certifying compilers**.

- **Proof-carrying code**: object code equipped with proof of safety.
- **Typed intermediate and assembly**: equip object code with a safe type system.

Recent work certifies equivalence of source and object code, even in the presence of optimization!

Robin's Influence on Language Research

Robin's influence on programming language research is both broad and deep.

- Imagined for us what languages could be.
- Initiated a scientific theory of language design.

There's much, much more than I've mentioned here!

Farewell, Robin

I would dearly love to follow this kind of work through to the front line of design and experience, but I don't think I'll be around for long enough. I'm making up for it by listening and talking to those who will be!

Edinburgh 2010

Robin and Lucy Milner