Robin Milner, 1934–2010

His work in theorem proving and verification

John Harrison

Intel Corporation

January 28th, 2011 (09:15–09:27)

Invited speaker at TPHOLs 2000?

```
From: Robin Milner <Robin.Milner@cl.cam.ac.uk>
To: John Harrison <johnh@ichips.intel.com>
Date: Tue, 25 Jan 2000 11:32:39 +0000

Dear John

Thanks very much for inviting me to speak at
TPHOLs.  I would enjoy it, but the main question
is whether I can offer enough of a perspective on
automated and interactive theorem proving, as I
haven't done any to speak of for 20 years!
```

1968: Arrival as a researcher in Swansea

What really sparked me off was getting interested in program verification and what semantics might mean. When I went to Swansea in 1968 I took a research job, I gave up teaching and became a research assistant with David Cooper who was head of the department in Swansea. He had a small group there, working on program verification and automatic theorem-proving and semantics.

1968: Arrival as a researcher in Swansea

> *What really sparked me off was getting interested in program verification and what semantics might mean. When I went to Swansea in 1968 I took a research job, I gave up teaching and became a research assistant with David Cooper who was head of the department in Swansea. He had a small group there, working on program verification and automatic theorem-proving and semantics.*

Cooper is perhaps most famous for the first elementary-time decision procedure for linear integer (Presburger) arithmetic.

1968: Arrival as a researcher in Swansea

> *What really sparked me off was getting interested in program verification and what semantics might mean. When I went to Swansea in 1968 I took a research job, I gave up teaching and became a research assistant with David Cooper who was head of the department in Swansea. He had a small group there, working on program verification and automatic theorem-proving and semantics.*

Cooper is perhaps most famous for the first elementary-time decision procedure for linear integer (Presburger) arithmetic. (In fact, arguably the first for any significant first-order theory.)

1969: Dana Scott's Oxford lectures

That was at the time when Dana Scott produced his famous domain theory. He gave a series of talks then, in '69, and I went over to Oxford and heard him. That was very exciting.
So, in some sense, it began to move very fast. The idea of a machine proving theorems in logic, and the idea of using logic to understand what a machine was doing ... this double relationship began to inspire me because it was clearly not very simple.

1969: Dana Scott's Oxford lectures

> *That was at the time when Dana Scott produced his famous domain theory. He gave a series of talks then, in '69, and I went over to Oxford and heard him. That was very exciting.*
> *So, in some sense, it began to move very fast. The idea of a machine proving theorems in logic, and the idea of using logic to understand what a machine was doing ... this double relationship began to inspire me because it was clearly not very simple.*

Dana Scott's influence on semantics is well-known, but his work was also an important factor in the development of interactive theorem proving.

1970: From automated to interactive proving

I wrote an automatic theorem prover in Swansea for myself and became shattered with the difficulty of doing anything interesting in that direction and I still am. I greatly admired Robinson's resolution principle, a wonderful breakthrough; but in fact the amount of stuff you can prove with fully automatic theorem proving is still very small. So I was always more interested in amplifying human intelligence than I am in artificial intelligence.

1970: From automated to interactive proving

> *I wrote an automatic theorem prover in Swansea for myself and became shattered with the difficulty of doing anything interesting in that direction and I still am. I greatly admired Robinson's resolution principle, a wonderful breakthrough; but in fact the amount of stuff you can prove with fully automatic theorem proving is still very small. So I was always more interested in amplifying human intelligence than I am in artificial intelligence.*

Interest in 'interactive' theorem proving was growing at the time, either because

- Abilities of ATP systems had grown but were plateauing
- More interactive computers made it natural/convenient

1971–2: Move to Stanford and Stanford LCF

I spoke to Zohar Manna [...] As a result I got a job with McCarthy from 1970, from the beginning of '71, at the AI lab in Stanford.

1971–2: Move to Stanford and Stanford LCF

> *I spoke to Zohar Manna [...] As a result I got a job with McCarthy from 1970, from the beginning of '71, at the AI lab in Stanford.*

Stanford LCF was a proof assistant for Scott's Logic of Computable Functions (LCF), developed by Milner together with Whitfield Diffie, Richard Weyhrauch and Malcolm Newey.

1971–2: Move to Stanford and Stanford LCF

> *I spoke to Zohar Manna [...] As a result I got a job with McCarthy from 1970, from the beginning of '71, at the AI lab in Stanford.*

Stanford LCF was a proof assistant for Scott's Logic of Computable Functions (LCF), developed by Milner together with Whitfield Diffie, Richard Weyhrauch and Malcolm Newey.

- Support for backward, goal-directed proof

1971–2: Move to Stanford and Stanford LCF

> *I spoke to Zohar Manna [...] As a result I got a job with McCarthy from 1970, from the beginning of '71, at the AI lab in Stanford.*

Stanford LCF was a proof assistant for Scott's Logic of Computable Functions (LCF), developed by Milner together with Whitfield Diffie, Richard Weyhrauch and Malcolm Newey.

- Support for backward, goal-directed proof
- A powerful simplification mechanism

1971–2: Move to Stanford and Stanford LCF

> *I spoke to Zohar Manna [...] As a result I got a job with McCarthy from 1970, from the beginning of '71, at the AI lab in Stanford.*

Stanford LCF was a proof assistant for Scott's Logic of Computable Functions (LCF), developed by Milner together with Whitfield Diffie, Richard Weyhrauch and Malcolm Newey.

- Support for backward, goal-directed proof
- A powerful simplification mechanism

Just one of many significant proof assistants being developed at around the same time (AUTOMATH, Mizar, SAM), and had significant drawbacks:

1971–2: Move to Stanford and Stanford LCF

> *I spoke to Zohar Manna [...] As a result I got a job with McCarthy from 1970, from the beginning of '71, at the AI lab in Stanford.*

Stanford LCF was a proof assistant for Scott's Logic of Computable Functions (LCF), developed by Milner together with Whitfield Diffie, Richard Weyhrauch and Malcolm Newey.

- Support for backward, goal-directed proof
- A powerful simplification mechanism

Just one of many significant proof assistants being developed at around the same time (AUTOMATH, Mizar, SAM), and had significant drawbacks:

- Memory limitations made it hard to store large proofs

1971–2: Move to Stanford and Stanford LCF

> *I spoke to Zohar Manna [...] As a result I got a job with McCarthy from 1970, from the beginning of '71, at the AI lab in Stanford.*

Stanford LCF was a proof assistant for Scott's Logic of Computable Functions (LCF), developed by Milner together with Whitfield Diffie, Richard Weyhrauch and Malcolm Newey.

- Support for backward, goal-directed proof
- A powerful simplification mechanism

Just one of many significant proof assistants being developed at around the same time (AUTOMATH, Mizar, SAM), and had significant drawbacks:

- Memory limitations made it hard to store large proofs
- The set of proof commands could not easily be extended.

1973-1978: Move to Edinburgh and Edinburgh LCF

Edinburgh LCF, developed with Malcolm Newey, Lockwood Morris, Mike Gordon and Chris Wadsworth, tackled the two shortcomings of Stanford LCF:

1973-1978: Move to Edinburgh and Edinburgh LCF

Edinburgh LCF, developed with Malcolm Newey, Lockwood Morris, Mike Gordon and Chris Wadsworth, tackled the two shortcomings of Stanford LCF:

- Did not store complete proofs, just remembering the *conclusions* of proofs

1973-1978: Move to Edinburgh and Edinburgh LCF

Edinburgh LCF, developed with Malcolm Newey, Lockwood Morris, Mike Gordon and Chris Wadsworth, tackled the two shortcomings of Stanford LCF:

- Did not store complete proofs, just remembering the *conclusions* of proofs
- Provided a full programming 'meta-language' (ML) so that the user could extend the set of proof commands

1973-1978: Move to Edinburgh and Edinburgh LCF

Edinburgh LCF, developed with Malcolm Newey, Lockwood Morris, Mike Gordon and Chris Wadsworth, tackled the two shortcomings of Stanford LCF:

- Did not store complete proofs, just remembering the *conclusions* of proofs
- Provided a full programming ‘meta-language’ (ML) so that the user could extend the set of proof commands

But how to ensure that theorems were proved correctly, not just arbitrarily asserted or created by buggy user proof commands?

1973-1978: Move to Edinburgh and Edinburgh LCF

Edinburgh LCF, developed with Malcolm Newey, Lockwood Morris, Mike Gordon and Chris Wadsworth, tackled the two shortcomings of Stanford LCF:

- Did not store complete proofs, just remembering the *conclusions* of proofs
- Provided a full programming 'meta-language' (ML) so that the user could extend the set of proof commands

But how to ensure that theorems were proved correctly, not just arbitrarily asserted or created by buggy user proof commands?

- Make theorems an abstract type in the metalanguage ('`thm`') with its only constructors being primitive inference rules of the logic.

1973-1978: Move to Edinburgh and Edinburgh LCF

Edinburgh LCF, developed with Malcolm Newey, Lockwood Morris, Mike Gordon and Chris Wadsworth, tackled the two shortcomings of Stanford LCF:

- Did not store complete proofs, just remembering the *conclusions* of proofs
- Provided a full programming 'meta-language' (ML) so that the user could extend the set of proof commands

But how to ensure that theorems were proved correctly, not just arbitrarily asserted or created by buggy user proof commands?

- Make theorems an abstract type in the metalanguage ('`thm`') with its only constructors being primitive inference rules of the logic.

The requirements of the LCF system directly motivated many features of ML.

How an LCF-style prover works

A logical inference rule such as $\Rightarrow$-elimination (*modus ponens*)

$$\frac{\Gamma \vdash p \Rightarrow q \qquad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}$$

How an LCF-style prover works

A logical inference rule such as $\Rightarrow$-elimination (*modus ponens*)

$$\frac{\Gamma \vdash p \Rightarrow q \qquad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}$$

becomes a *function*, say `MP : thm->thm->thm` in the metalanguage.

How an LCF-style prover works

A logical inference rule such as $\Rightarrow$-elimination (*modus ponens*)

$$\frac{\Gamma \vdash p \Rightarrow q \qquad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}$$

becomes a *function*, say `MP : thm->thm->thm` in the metalanguage.

For example, if `th1` is the theorem $\vdash p \Rightarrow (q \Rightarrow p)$ and `th2` is $p \vdash p$, then `MP th1 th2` gives $p \vdash q \Rightarrow p$.

How an LCF-style prover works

A logical inference rule such as $\Rightarrow$-elimination (*modus ponens*)

$$\frac{\Gamma \vdash p \Rightarrow q \qquad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}$$

becomes a *function*, say `MP : thm->thm->thm` in the metalanguage.

For example, if `th1` is the theorem $\vdash p \Rightarrow (q \Rightarrow p)$ and `th2` is $p \vdash p$, then `MP th1 th2` gives $p \vdash q \Rightarrow p$.

Highly automated or convenient *derived* inference rules can be programmed using these as the basic building-blocks, including support for backward proof via ‘tactics’.

The LCF diaspora

[LCF] didn't immediately get applied a great deal, but Mike Gordon brought it to Cambridge [...] He started doing hardware verification. And then one or two other people began to design verifications systems, or rather systems to perform computer-assisted proof, on the model of our system, particularly Constable at Cornell with his NuPrl.

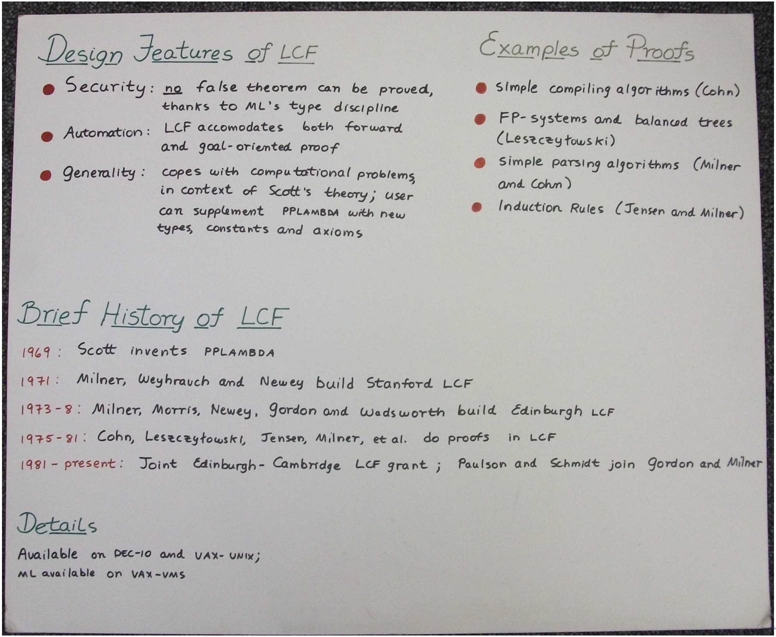
The LCF diaspora

> *[LCF] didn't immediately get applied a great deal, but Mike Gordon brought it to Cambridge [...] He started doing hardware verification. And then one or two other people began to design verifications systems, or rather systems to perform computer-assisted proof, on the model of our system, particularly Constable at Cornell with his NuPrl.*

Despite the name, which has stuck, the LCF approach is not tied to the Logic of Computable Functions, and many other LCF-style provers were written over the years.

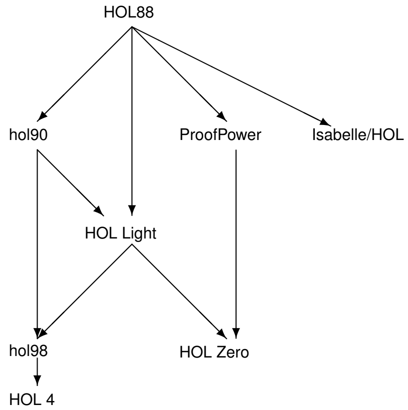
- Cambridge LCF (rationalized system for same LCF logic)
- HOL (higher-order logic based on polymorphic type theory)
- Nuprl (Martin-Löf type theory)
- Coq (the Calculus of Inductive Constructions)
- Isabelle (framework supporting multiple object logics)

Avra Cohn’s LCF poster



The HOL family DAG

The HOL system alone has given rise to numerous different LCF-style implementations of essentially the same logic:



Applications of LCF provers

Several LCF-style systems have been used for major work in formal verification and formalization of mathematics.

- Verification of microprocessors, compilers, floating-point microcode, cryptographic protocols, OS kernels, properties of programming languages and their type systems ....
- Formalization of Jordan Curve Theorem, Prime Number Theorem, 4-Colour Theorem .... Feit-Thompson theorem and Kepler conjecture ('Flyspeck project') in progress.

The ideas that Robin Milner developed almost 40 years ago are central to machine-assisted proof today.

Invited speaker at TPHOLs 2000

From: Robin Milner <Robin.Milner@cl.cam.ac.uk>
To: John Harrison <johnh@ichips.intel.com>
Date: Fri, 28 Jan 2000 17:26:21 +0000

Dear John

I've thought a bit more. I believe I can offer, informally, some interesting reminiscences. What I can't do, given my current preoccupation, is to spend very long on that. (Indeed, a whole talk would be too hard.) At present I am working flat out on the theory I mentioned, hence my interest in how machine assistance could help with it. I can't afford too much time away from this task over the coming year -- life is short.