Chapter 2
Recollections of Hope Park Square, 1970-73

J Strother Moore

Abstract I reminisce about the time and place the DReaM Group started: the Metamathematics Unit and its sister, the Department of Machine Intelligence and Perception, in Hope Park Square, Meadow Lane, Edinburgh, in the early 1970s. This is not meant as a scholarly history but just a personal recollection that tries to capture the spirit of the times.

2.1 Arrival

"2 Hope Park Square, Meadow Lane, Edinburgh" read the address on my letter of acceptance. I couldn't imagine a more romantic address, nor a more improbable one for an academic department with the equally improbable name "Department of Machine Intelligence and Perception." In was September, 1970, and I had just arrived from MIT where I'd gotten my bachelor's degree in mathematics. Now I was going to start my PhD studies. My academic career seemed to keep taking me back in time. In Texas, where I grew up, a building was old if it was built in the 1930s. Cambridge, Massachusetts, where MIT is located, pushed "old" back to the 18th century. But then there was Edinburgh, whose "New Town" was started in 1766. Parts of Hope Park Square predated that.

Hope Park Square was a collection of stone buildings surrounding an overgrown garden that you entered through an archway. The buildings housed both Machine Intelligence and Perception and the Metamathematics Unit.[1] I went through the

J Strother Moore
The University of Texas at Austin
Computer Science Department
2317 Speedway, Stop D9500
Austin, TX 78712 - 1757 USA e-mail: moore@cs.utexas.edu

[1] The Metamathematics Unit, whose name was soon changed to the Department of Computational Logic, became a founding member of the School of Artificial Intelligence.

arch and entered the building to be greeted by the Servitor – what a strange idea to have a door man for an academic department. "I'm here to see Rod Burstall," I explained, and was directed up the stairs. Rod had signed my letter of acceptance and so it seemed natural to start with him. The next person I met turned out to be his secretary, Eleanor Kerse, who pointed me to his door. I knocked and upon being invited in I found him standing on his head. "I'll be done in a few minutes," he said.

After introductions, Rod took me to a nearby office with two desks. "There are pencils and paper in that cupboard. This sheet explains how to log on to the timesharing system." He also handed me a slim silver book, "This describes the programming language, POP-2." He didn't note that he was a co-author of it, along with Robin Popplestone.[2] POP-2 was an elegant Lisp-like programming language with an Algol-like syntax. "We have tea every day at 11 am and 4 pm. Welcome to Hope Park Square." And then he left.

My research career had begun. My first surprise was how different the British educational system was from the American one I had experienced. My office mate in the Machine Intelligence department was Mike Gordon, who like me was a new PhD students that year. From Mike I learned that my undergraduate education at MIT differed greatly from his at Cambridge. We both majored in Mathematics, but at MIT I was required to take a broad range of science and technical courses and consequently less maths, whereas at Cambridge Mike focused almost entirely on maths. But the biggest difference in the systems, as I came to experience it, was in the PhD programs. At Edinburgh there were no classes, no lectures, no homework, no exams – except the oral exam for my dissertation. And, at least in my case, no direction. While I wasn't aware of it at the time, a PhD in Edinburgh was typically completed in 3 years, as opposed to 5-7 in the US. I've come to appreciate this shortened time span because it can enable more imaginative dissertation topics! You are unlikely to solve the problem in 3 years so you can aim high and try to demonstrate the plausibility of an approach. While there were no formal classes, Hope Park Square hosted a steady stream of outstanding scientists who gave seminars on cutting edge research in AI and theorem proving – Marvin Minksy, John McCarthy, Christopher Strachey, Dana Scott, J. Alan Robinson, Woody Bledsoe, Danny Bobrow, Robin Milner. In addition, tea time was invariably interesting and educational, with Burstall, Popplestone, Donald Michie, Harry Barrow, our visitors, and half a dozen PhD students including Mike, Gordon Plotkin, and John Darlington.

My plan in 1970 was to build a natural language understanding system that could read and answer questions about children's stories. My adviser was Donald Michie.

Donald and I spoke often in those early days. He sometimes talked about his days at Bletchley Park, even though in 1970 that work was still classified. I now know he worked on the German "Tunny" encryption, which was the British code name for the Lorenz cipher, which was considered harder than Enigma. The work at Bletchley

[2] I got to know Robin soon after that. He had a sailboat berthed in Leith and wanted it in Inverness so he could take his family down the Great Glen, through Loch Ness to Fort William. But he didn't want to sail his family through the North Sea. He needed someone to crew for him and I volunteered. I'd spent a lot of time in boats growing up – mainly canoes on the bayou – but occasionally in dinghies and sail boats in Galveston Bay.

Park was not declassified until after I left Edinburgh, so Donald never told me much. But he did say they didn't have the code machine and so they built a simulator for it. And I asked a really naive question without thinking: "Did you do that in software or actually build a physical machine?" He had to remind me: they were inventing digital computers then. "Software" was a thing of the future.

Meanwhile, I needed a job in Edinburgh. I had financed most of my MIT education as a computer programmer. I learned to program, in Fortran, in 1964, while in high school, by taking a Summer Science Training course sponsored by the National Science Foundation. My first job at MIT was in the MIT Laser Research Group, where I was hired to fetch and carry glass tubes, electrodes, machined glass, etc., from the physics stores and machine shops to the lab. I wasn't very good at that – I'd often bring back the wrong thing. And while I was waiting for my next assignment, I'd watch the grad students try to figure out what was wrong with their Fortran programs. "You need a comma there." Or, "I think you mean X here." After a few weeks of unsuccessful fetch-and-carry, one of the students said "I'll go get the tubing. You try to fix this program." By my second semester at MIT I was their programmer and I spent my time on numerical methods for solving differential equations.

But in 1968, back home in Texas for the summer, I was an intern working for TRW Systems Group at the Manned Spacecraft Center, debugging the Lunar Orbit Insertion procedure for Apollo by simulating missions: preparing card decks that put the simulator in some initial state and then simulating, second by second, 30 minutes of flight through variations of the events allowed by the flight plans. That summer changed my appreciation of computing: it wasn't just for solving differential equations. Inside a computer, a person could build a whole solar system – or any world at all. I returned to MIT, finished my math requirements perfunctorily, and focused on Artificial Intelligence. The next summer I helped write a page fault simulator for IBM Cambridge to explore memory management on the then-new System 360. And in my last year of MIT I was a full-time employee of State Street Bank and Trust, Boston, working on their mutual fund software in PL/1.

So when Donald spoke of simulating the coding machine, digital simulation sprang naturally to mind. And when I needed a job in Edinburgh, the natural thing to do was to look for a programming job. I found one next door, at 9 Hope Park Square, in Bernard Meltzer's Metamathematics Unit. As far as I can recall, the Metamathematics Unit then consisted of Bernard, his quite remarkable secretary, Jean Pollock, two post-doc research fellows, Bob Kowalski and Pat Hayes, and Don Kuehner who was finishing up a PhD. We called the institution simply "the Unit" and I wore a Dr. Who pin reading "U.N.I.T." on my blue jean jacket the entire time I was in Edinburgh.

Kowalski and Kuehner had invented "Selected Literal (SL-) resolution," a restriction of resolution that was complete and enjoyed certain properties related to proof search efficiency and the shortness of proofs. Neither Kowalski nor Kuehner enjoyed programming and so I was hired to be Kowalski's programmer and my job was to create the first implementation of SL-resolution.

The job required getting a work permit. I remember sitting across from Bernard in his office and him talking on the phone to some official who would pursue the

work permit. Bernard would listen to the person and then cover the phone and ask me "How long have you been programming?" "6 years," I replied. So Bernard said, "We need someone with 5 or 6 years of programming experience." Then, covering the phone, he asked me "In what languages?" "Fortran, assembly language, PL/1, and Lisp." And so it went. The eventual job description probably described exactly one person in the UK. So I got my work permit.

Bob Kowalski then described what he wanted and I started implementing my first theorem prover. Before the first year was out I realized I was a lot better at writing theorem provers than I was at getting computers to understand children's stories. So I went to Donald and asked if I could change advisers. He agreed and Rod Burstall, the Machine Intelligence faculty member most familiar with logic and mechanized proof, became my official dissertation adviser. I moved from the Machine Intelligence office that Rod had first shown me to an office in the Unit.

The programming environment was fairly primitive, at least by the standards of MIT in the late 1960s. Researchers at Hope Park Square shared an ICL 4130 processor with 64K bytes of 2 microsecond memory, three 4MB disc drives, two paper tape readers and punches, and a 300 line-per-minute line printer. It could support 8 interactive jobs, each controlled by teletype, all permanently core resident. Time sharing was strictly round-robin with all users having equal priority. Job control and programming was in POP-2, so from the user's perspective it was a "POP-2 machine" akin to but pre-dating the "Lisp machine." The CPU could be scheduled for exclusive use at night. Memory limitations drove almost all programming decisions. Teletypes connected Hope Park Square to the CPU; but all other hardware was located about 10 minutes' walk away at Forrest Hill.

Files were edited with a software-implemented paper tape editor, a machine that copies a paper tape by punching corresponding holes in a blank tape but permitting the user to skip sections or manually punch different holes. To edit a file you copied the bytes in it to another file, stopping at the first place you wanted to change. You'd type the new text and/or delete the old text, and then continue copying. You could not look at more than one byte at a time, back up, or undo. Repeated passes through the evolving file were typically required. Editing a program was tedious and error prone. If you printed your file on the line printer to see the final result of editing you needed to walk along The Meadows to Forrest Hill to get your output.

Memory constraints made my SL-resolution prover quite limited, and my attempts to save space by representing clauses in a compressed string format slowed it down. But by September, 1971, I could demonstrate SL-resolution on simple sets of clauses. Kowalski could ignore the performance issues but was disappointed by the size of the search space. One of his responses to this was to develop an amazing knack of creating clause sets whose refutation was straightforward, almost like the SL-resolution engine was interpreting code.

When Don Kuehner graduated with his PhD, I bought his automobile: a diesel-powered, decommissioned, Austin FX4 London taxi. It cost me £75 in 1971 and I sold it for £5 in December, 1973, because it could no longer pass its MOT inspection. In the meantime, it was an excellent vehicle for exploring Scotland – its tight turning radius made it ideal for Highland roads – and its roomy passenger compartment was

great for hauling furniture. Don and I both regarded the taxi as a shared resource at Hope Park Square.

In September, 1971, two very influential people joined the Unit as post-docs: Alan Bundy and Bob Boyer.

Because Boyer and I were both from Texas we shared an office: it was the warmest room in Hope Park Square. Our office overlooked the bowling green and practice range of the Royal Company of Archers, the Queen's ceremonial bodyguard in Scotland. At other times there were bowlers on the green. The police bagpipe band also practiced there.

Boyer and I began to collaborate on a variety of projects including an efficient, structure-shared way to represent clauses in a resolution prover[BM72]. Instead of copying clauses to rename variables apart and implement substitution and resolvent formation, resolvents were represented by pointing to the two parents, noting which literals were resolved upon, and storing the unifying substitution. Linear proofs produced just stacks of frames, reminiscent to the way conventional programming languages implement procedure calls.[3]

In December, 1971, J. A. Robinson – who invented resolution and who made several long visits to Edinburgh – playfully awarded us one of my prized honors, a handwritten certificate that reads:

> Foundation for the Advancement of Computational Logic
> by the Taking Out of Fingers
>
> 1971 PROGRAMMING PRIZE
>
> In 1971, by the gift of an anonymous (he couldn't even remember his own name!) donor, an annual programming prize was set up, to be awarded to:
>
> > "...that person, or those persons, who, in the opinion of the Board of Trustees, shall have, in the given year, contributed the most valuable, beautiful, or just plain deep and satisfying, idea to the world in the area of actual writing of programs in computational logic, as opposed to simply waving hands and hoping things will work out all right on the night...";
>
> In 1971, the prize is award, by unanimous agreement of the Board, to
>
> > Robert S. Boyer
> > and
> > J Strother Moore
>
> for their idea, explained in "The Sharing of Structure in Resolution

[3] The connection of structure sharing to Prolog and the Warren Abstract Machine is mentioned later.

> Programs", of representing clauses as their own genesis. The Board declared, on making the announcement of the award, that this idea is "... bloody marvelous".
>
> Because of the state of the economy, and what with one thing and another, the prize this first year is somewhat less than it will be in future years.
>
> J. A. Robinson
> Secretary
> December 12, 1971

Once Boyer and I got the SL-resolution prover converted to a structured shared clause representation, we began experimenting with proofs. But because we were passionate about programming, we tended to focus on proofs about programs.

We formalized an assembly-like language called Baroque in predicate calculus and used the SL-resolution prover as an interpreter for it. Then, programming in Baroque, we axiomatized a simple subset of Pure Lisp and could use SL-resolution to run Lisp programs and to prove simple theorems about Lisp. See pages 73–75 of my dissertation [Moo73], where, for example, I define

```
MEMBER: (MEMBER X Y) -> U
        WHERE
        (COND Y
              (COND (EQUAL X (CAR Y))
                    T
                    (MEMBER X (CDR Y)))
              NIL) -> U;
        END;
```

(At the time we used `COND` as a 3-place if-then-else.) If one executed the statement

```
(MEMBER 2 (CONS 1 (CONS 2 (CONS 3 NIL)))) -> U;
```

the SL-resolution "interpreter" would bind `U` to `T`. More interesting to us at the time was that we could "run" programs "backwards" finding input that satisfied certain output constraints, e.g.,

```
(MEMBER 2 (CONS 1 (CONS X (CONS 3 NIL)))) -> T;
```

would bind `X` to `2`, proving "there exists an `X` such that `2` is a `MEMBER` of `(CONS 1 (CONS X (CONS 3 NIL)))`.

In Part I of my 1973 dissertation I explored various capabilities allowed by structure sharing and "programming in the predicate calculus," including pre-computing unifiers, "shallow binding" to short-circuit the dive through the stack for a binding, how to attach pragmatic restrictions on variables to prevent any instantiation from forming certain "useless" terms, and Baroque.

There were many other ramifications of structure sharing we could have explored, but our abiding interest was in proving more interesting theorems like the associativity of list concatenation, or that something is a `MEMBER` of the concatenation of two

lists iff it is a `MEMBER` of one or the other. These theorems were out of reach of our SL-prover: they require induction. So we set aside resolution and structure sharing and focused on proving inductive theorems about Lisp functions.

Following McCarthy [McC63] we adopted a simple subset of Pure Lisp as our logic and took turns defining functions and proving theorems at the blackboard, questioning each move. "Why did you expand that function?" "Why induct on `A`?" "Is the conjecture general enough for induction?" As strange as it may seem today, these were groundbreaking questions in the early 1970s because "theorem proving" was almost synonymous with uniform proof procedures for first-order predicate calculus. After several months we had a collection of heuristics to help decide when to induct and what inductive argument to use, as well as heuristics for controlling the application of axioms as rewrite rules, including the unfolding of recursively defined functions and rudimentary generalization.

We then set about implementing this in POP-2, creating the Edinburgh Pure Lisp Theorem Prover (PLTP), which was running by March, 1973, in the style of Woody Bledsoe [BBH72]. The techniques developed for PLTP have been widely adopted in virtually all modern theorem provers aimed at hardware and software verification. Indeed, I think we founded the now-thriving field of mechanized inductive theorem proving.

PLTP was a fully automatic, heuristic theorem prover for Pure Lisp, focused on induction and recursion. The user presented it with Lisp function definitions and conjectures to prove. It printed a narrative description of its evolving proof attempt. Unlike resolution provers, PLTP did not backtrack or do much search. As I said in my dissertation ([Moo73] page 208) "The program is designed to make the right decision the first time, and then pursue one goal with power and perseverance."

Its proof techniques included simplification via rewriting with Lisp axioms and definitions, heuristic use of equality, generalization, and induction. These techniques were tightly coupled so that induction set up simplification, simplification was used to determine appropriate inductions, and equality substitution and generalization were used to produce conjectures intended for inductive proofs and tended to "discover" interesting subgoals. For example, the associativity of Peano multiplication required three inductions and discovered the distributivity of multiplication over addition and the associativity of addition – a curious level of competence demonstrated too often by PLTP to be random or coincidental and explained years later by Alan Bundy's work on rippling.

Boyer and I kept a file containing all the definitions and theorems the prover succeeded on. We called it "the proveall." Every time we'd change the heuristics we ran the proveall because we'd learned from past experience that it was easy to "improve" a theorem prover so that it could find a previously undiscoverable proof without us realizing it could no longer discover some "old" proofs. The proveall grew as we refined the heuristics through 1973. We established the discipline of never accepting an "improvement" until the new system had passed the proveall test. (We sometimes found that failures had more to do with peculiar aspects of the statements of "old" theorems than with faulty heuristics and restated those theorems

to take advantage of new techniques.) We follow this discipline today and highly recommend it to developers of theorem provers.

Sometime during late 1971 or early 1972, we simply couldn't stand to use the paper-tape editor to maintain our software. Inspired by our structure sharing work, we invented a way to edit a file without using much memory: build a data structure that described the edited document in terms of segments of the original file (on disk) and the text entered by the user during the edit session. We named the editor the "77-Editor" because it resided on disk track 77. We had help from D.J.M. Davies dealing with some system programming issues. The editor supported the illusion that the entire document was in memory, you could search backwards and forwards, move around in it, undo changes, etc. [BDM73]. We used the 77-Editor extensively to create PLTP.

For lunch there was a little shop at the end of Meadow Lane that sold seven kinds of sandwiches: beans, cheese, fried egg, beans and cheese, beans and fried egg, cheese and fried eggs, and beans, cheese, and fried eggs.

When Boyer and I were stuck we'd often take walks, usually to Holyrood Park, and often the solution would come to us there. I sometimes went jogging with Alan Robinson around Holyrood Park. Once going up the road around Arthur's Seat I said to him "I wish I had a low gear, like my bike," and he replied "You do: take smaller steps."

Some afternoons Boyer and I would go out to the Meadows behind Hope Park Square and join a pickup game of (British) football, often with school boys. They could dribble circles around the clumsy Americans. But we paid them back when we'd play (American) football or baseball, both of which require throwing and catching.

Once returning from the Meadows, we found a man trying to steal Boyer's bicycle, which was parked in the archway of Hope Park Square. Boyer chased him with the baseball bat and it is fortunate for the thief as well as for theorem proving that Boyer didn't catch him. As someone said when we got back to the Unit, "A liberal is someone who's never been robbed."

In the summer of 1973, Rod Burstall came into the office and told me "You should write this up." My 3 years were ending! In the 23 months that Boyer and I worked together in Edinburgh – creating structure sharing, a text editor, and an inductive theorem prover – I'm not sure that either of us thought about my PhD. We were just doing research, chasing our shared dream of automated reasoning about programs.

I wrote my dissertation that summer. Part I was about structure sharing. Part II was about PLTP. I wrote "two" dissertations because it was impossible to separate my contributions from Boyer's. The PLTP proveall then contained 47 function definitions and 67 theorems about recursive list processing (e.g., concatenation, reverse, member, union, intersection, sorting), Peano arithmetic (e.g., addition, multiplication, exponentiation), tree processing (copy, flatten, searching), and connections between them (e.g., the length of a concatenation is the sum of the lengths). All the PLTP theorems were proved completely automatically. Because PLTP was the first mechanical prover designed around induction and most of these theorems required induction, most of these theorems had never been proved mechanically before.

My oral exam was in the Fall, 1973, and my internal examiners were Bernard and Rod and my external examiners were David Cooper and Robin Milner. Milner was visiting from Stanford where he was developing LCF [Mil79]; he had read my dissertation very carefully and was especially interested in induction. We sat in Bernard's office and had tea over a teletype. They'd challenge PLTP and I'd explain either why it failed or how it succeeded. Some of the challenge theorems were familiar because Boyer and I had seen it prove them, e.g., that insertion sort produced ordered output. But some were proved for the first time during the oral exam, including that insertion sort preserved the number of occurrences of an element. The committee objected to my use of the non-word "normalation" as the name of the proof technique finally called "simplification." And, with a few edits to the dissertation, I was done.[4]

We left Edinburgh in December, 1973, in part because of the Lighthill Report and the coming "AI winter" in the UK. But the ideas we developed in the Metamathematics Unit lived on and had real impact. Structure sharing, which had helped reify the idea of programming in the predicate calculus, played a role in the creation and implementations of Prolog and the Warren Abstract Machine. As for our text editor, it remained in use in the Edinburgh AI community for several years, until the ICL 4130 was replaced by a PDP-10. More importantly, our document representation became an integral part of Microsoft Word.

When I left Edinburgh I joined Xerox Palo Alto Research Center (PARC). There I learned that Charles Simonyi was implementing the first WYSIWYG text editor, Bravo, for the Xerox Alto personal computer and was facing severe memory limitations. I explained our document representation to him and implemented a package of text editing utilities in Interlisp as an experimental prototype of the basic operations of search, insertion, deletion, etc. He adopted the representation in his implementation of Bravo. I maintained the Interlisp package and added features at his request even after I left PARC in 1976 to join Boyer at SRI [Moo81]. The representation not only maintains a small memory footprint and facilitates undoing but enables metadata, like font and change tracking, to be attached to text without changing the text. When Simonyi left PARC and joined Microsoft he created Microsoft Word inspired in part by Bravo. He used our document representation there too and it is still in use in Word today.

And the Pure Lisp Theorem Prover remained the focus of Boyer's and my work. In subsequent years we explored a plethora of topics to improve the prover, including use of previously proved lemmas, verified metafunctions, integrated decision procedures, the adoption of a subset of an ANSI standard programming language as the logic, programming the system in its own logical language, and the dual use of formal models as specifications and efficiently executable prototypes. By 1979, PLTP had become Thm, the prover described in our 1979 book *A Computational Logic* [BM79]; by the mid-1980s, Thm had become Nqthm [BM88, BM97]; and by the early 1990s, Nqthm had become ACL2, *A C*omputational *L*ogic for *A*pplicative *C*ommon *L*isp [KMM00b, KMM00a, KM19]. For a sketch of that evolutionary sequence and the

[4] PLTP was reproduced several times in the 1970s. There are at least two modern re-implementations, one by me in ACL2 and one by Grant Passmore in ML. See the PLTP Archive at `http://www.cs.utexas.edu/users/moore/best-ideas/pltp/index.html`.

changes we made see [Moo19]. I'm still working on ACL2, but for the past 26 years my co-author has been Matt Kaufmann. We release a new version of ACL2 about twice a year. ACL2 is in routine use in industry including ARM, AMD, Centaur, Kestrel, Intel, Oracle, and Rockwell Collins – nightly in some cases – to verify microprocessor components and critical algorithms [WAHKMS17], dealing with conjectures that Boyer and I could not have imagined in 1973. The "proveall" has grown from PLTP's 67 theorems to ACL2's 153,823[5].

To understand how remarkable the Metamathematics Unit had become with the arrival of Boyer and Bundy consider this: Kowalski and Kuehner were primarily pursuing uniform proof procedures, specifically resolution in first-order predicate calculus. Early applications focused on formalizing a robot's world and using theorem proving to plan a sequence of actions to achieve some goal, following the formalization of McCarthy and Hayes' situational calculus [MH69]. Kowalski was particularly adept at formalizing clausal problems in a way that made SL proofs easy to find and that probably led to his view that one could program in the predicate calculus. Into this strictly first-order, resolution group Meltzer added two post-docs, Bundy and Boyer, who came at mechanized reasoning from completely different perspectives. Bundy did his dissertation on proofs of Gödel's incompleteness theorems in restricted formal systems of arithmetic [Bun71], supervised by Reuben Goodstein, a master of constructive mathematics and the foundations of logic. Arithmetic is inherently inductive and so Bundy's view of theorem proving was necessarily broader than resolution. Boyer was very familiar with resolution. His dissertation was on a restriction of resolution [Boy71]), but his supervisor, Woody Bledsoe, was a fierce advocate of *non*-uniform, heuristic provers. Boyer had co-authored a heuristic prover with Bledsoe and W.H. Henneman [BBH72] on proofs of limit theorems. And then there was me, a programmer learning theorem proving. Within a few months of Boyer's arrival at Hope Park Square, we were exploring inductive proofs about recursive data structures. Meltzer's genius is indicated by the group he assembled. We represented a wide variety of theorem proving backgrounds, styles, and applications. He basically just turned us loose.

The result was an intense, exciting, and fascinating time full of discovery. We discussed and debated everything from simple arithmetic challenges requiring generalization and induction to whether logic was an appropriate way to model human thought.

I remember discovering that some truly simple pragmas attached to "action variables" could make many of the robot problems easy, e.g., disallow the immediate composition of the action `LET-GO` onto the action `PICK-UP` or otherwise the prover would pursue the possibilities allowed by picking up an object and immediately letting it go, *ad infinitum*. Such pragmas were easily implemented in the structure shared representation, c.f. [Moo73], page 48.

[5] This is a conservative estimate as of July, 2019, of the number of theorems explicitly stated by users in ACL2 Community Books repository, `https://github.com/acl2/acl2`, which Kaufmann and I re-run for every new version of ACL2. It is conservative because it only counts conjectures presented with the `defthm` command and not conjectures required to admit definitions or presented or generated by macros, etc.

When debating whether predicate calculus could capture English Boyer clarified the question.

Boyer: “Give me a sentence.”

Kowalski: “The girl guides fish.”

Boyer (writing on the black board):

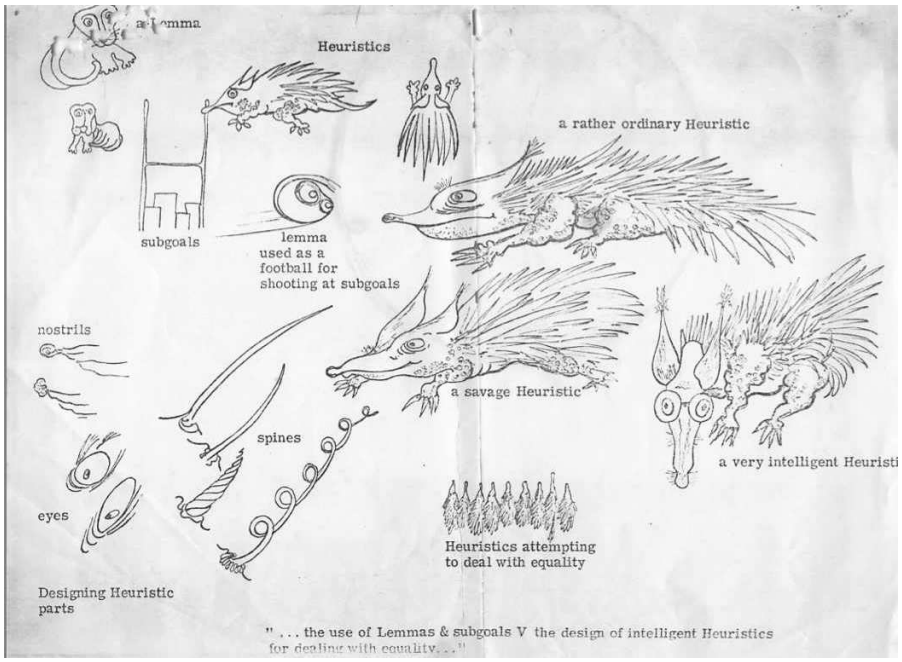
At(0,’T) ∧ At(1,’h) ∧ At(2,’e) ∧ At(3,Space) ∧ At(4,’g) . . ..

On another occasion, somebody asserted “People think in predicate calculus.” Somebody else said, “People think in English,” to which somebody else replied, “Actually, most people think in Chinese.”

Once after a tedious argument on a question I’ve long forgotten Kowalski said “Let’s vote. And then we’ll argue about who won.”

Of course, mostly we discussed theorem proving ideas: hyperresolution, paramodulation, uniform versus non-uniform proof procedures, the role of soundness and completeness, how to deal with the equality relation, the role of induction, the use of lemmas, heuristics for limiting the search space, etc.

Perhaps the best picture of those years at Hope Park Square was drawn in 1971 or 1972, by Martin Pollock, FRS, husband of Bernard’s secretary Jean. Martin was a founding father of molecular biology as a distinct field. He was also well known at the university for his satirical cartoons. Jean frequently had to type manuscripts that were incomprehensible to her. Below was Martin’s response to a paper by Bernard.



Since this book is dedicated to the research conducted by Alan Bundy’s DReaM Group it seems only fitting for me to comment on that work. Alan’s chapter on

the history of the DReaM group makes an interesting contrast to my history of the evolution of PLTP to ACL2 [Moo19]. There could hardly be two more different research styles: my focus was quite narrow and Alan's was extraordinarily broad. And yet it is interesting that we visited many of the same topics from our different perspectives, e.g., rippling, lemma discovery, the value of a formal meta-theory, and, of course, the concern over soundness and the impact of an *ad hoc* programming style versus a disciplined partition between heuristic and rules of inference. And in almost every case, he and I made different decisions: mine always driven by pragmatic desire to build a sound and effective prover for computational problems and his to understand how that is done. In fact, our almost half-century of pursuing related goals from different perspectives exemplifies the wonderful atmosphere of the Unit.

References

[BBH72] W.W. Bledsoe, R.S. Boyer, and W.H. Henneman. Computer proofs of limit theorems. *Artificial Intelligence*, 3:27–60, 1972.

[BDM73] R. S. Boyer, D. J. M. Davies, and J S. Moore. The 77-editor. Technical Report 62, Department of Computational Logic, University of Edinburgh, 1973.

[BM72] R. S. Boyer and J S. Moore. The sharing of structure in theorem-proving programs. In *Machine Intelligence 7*, pages 101–116. Edinburgh University Press, 1972.

[BM79] R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, New York, 1979.

[BM88] R. S. Boyer and J S. Moore. *A Computational Logic Handbook*. Academic Press, New York, 1988.

[BM97] R. S. Boyer and J S. Moore. *A Computational Logic Handbook, Second Edition*. Academic Press, New York, 1997.

[Boy71] Robert S. Boyer. *Locking: A Restriction of Resolution*. Department of Mathematics, University of Texas at Austin, 1971.

[Bun71] A. Bundy. *The Metatheory of the Elementary Equation Calculus*. PhD thesis, University of Leicester, August 1971.

[KM19] M. Kaufmann and J S. Moore. The ACL2 home page. In `http://www.cs.utexas.edu/users/moore/acl2/`. Dept. of Computer Sciences, University of Texas at Austin, 2019.

[KMM00a] M. Kaufmann, P. Manolios, and J S. Moore, editors. *Computer-Aided Reasoning: ACL2 Case Studies*. Kluwer Academic Press, Boston, MA., 2000.

[KMM00b] M. Kaufmann, P. Manolios, and J S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Press, Boston, MA., 2000.

[McC63] J. McCarthy. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*. North-Holland Publishing Company, Amsterdam, The Netherlands, 1963.

[MH69] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, volume 4, pages 463—502. Edinburgh University Press, 1969.

[Mil79] Robin Milner. Lcf: A way of doing proofs with a machine. In Jiří Bečvář, editor, *Mathematical Foundations of Computer Science 1979*, pages 146—-159. Lecture Notes in Computer Science, Vol. 74, Springer, Berlin Heidelberg, 1979.

[Moo73] J S. Moore. Computational logic: Structure sharing and proof of program properties. Ph.D. dissertation, University of Edinburgh, 1973. See `http://www.era.lib.ed.ac.uk/handle/1842/2245`.

[Moo81] J S. Moore. Text editing primitives – the TXDT package. Technical Report CSL-81-2 (see `http://www.cs.utexas.edu/users/moore/publications/txdt-package.pdf`), Xerox PARC, 1981.

[Moo19] J Strother Moore. Milestones from the pure lisp theorem prover to acl2. *Formal Aspects of Computing*, 2019.

[WAHKMS17] Jr. W. A. Hunt, M. Kaufmann, J S. Moore, and A. Slobodova. Industrial hardware and software verification with ACL2. In *Verified Trustworthy Software Systems*, volume 375. The Royal Society, 2017. (Article Number 20150399).