Machines Reasoning about Machines

A Personal Perspective

J Strother Moore
Department of Computer Sciences
University of Texas at Austin

Prologue

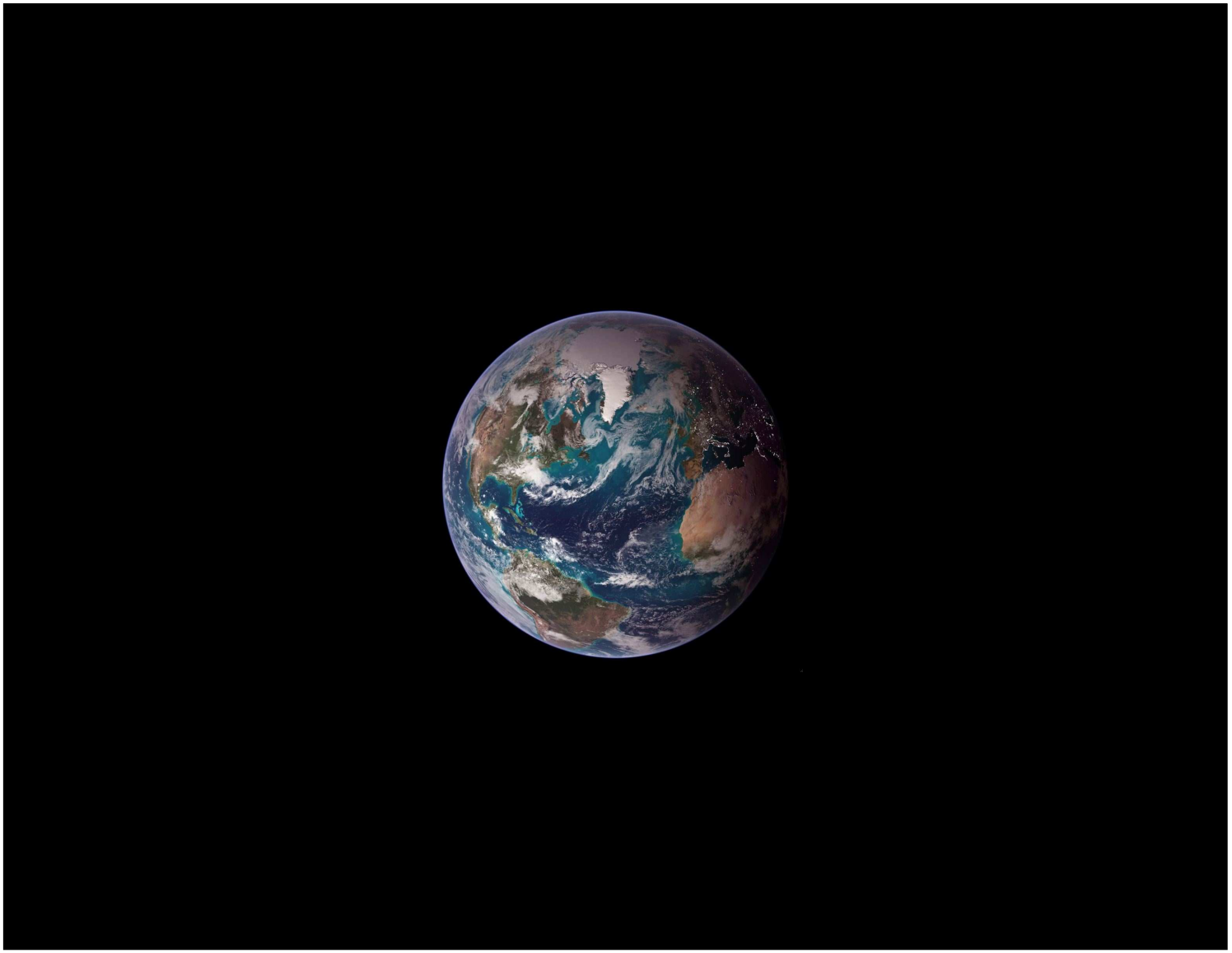
For forty years I have been working toward one goal: to make it practical to reason mechanically about hardware and software.

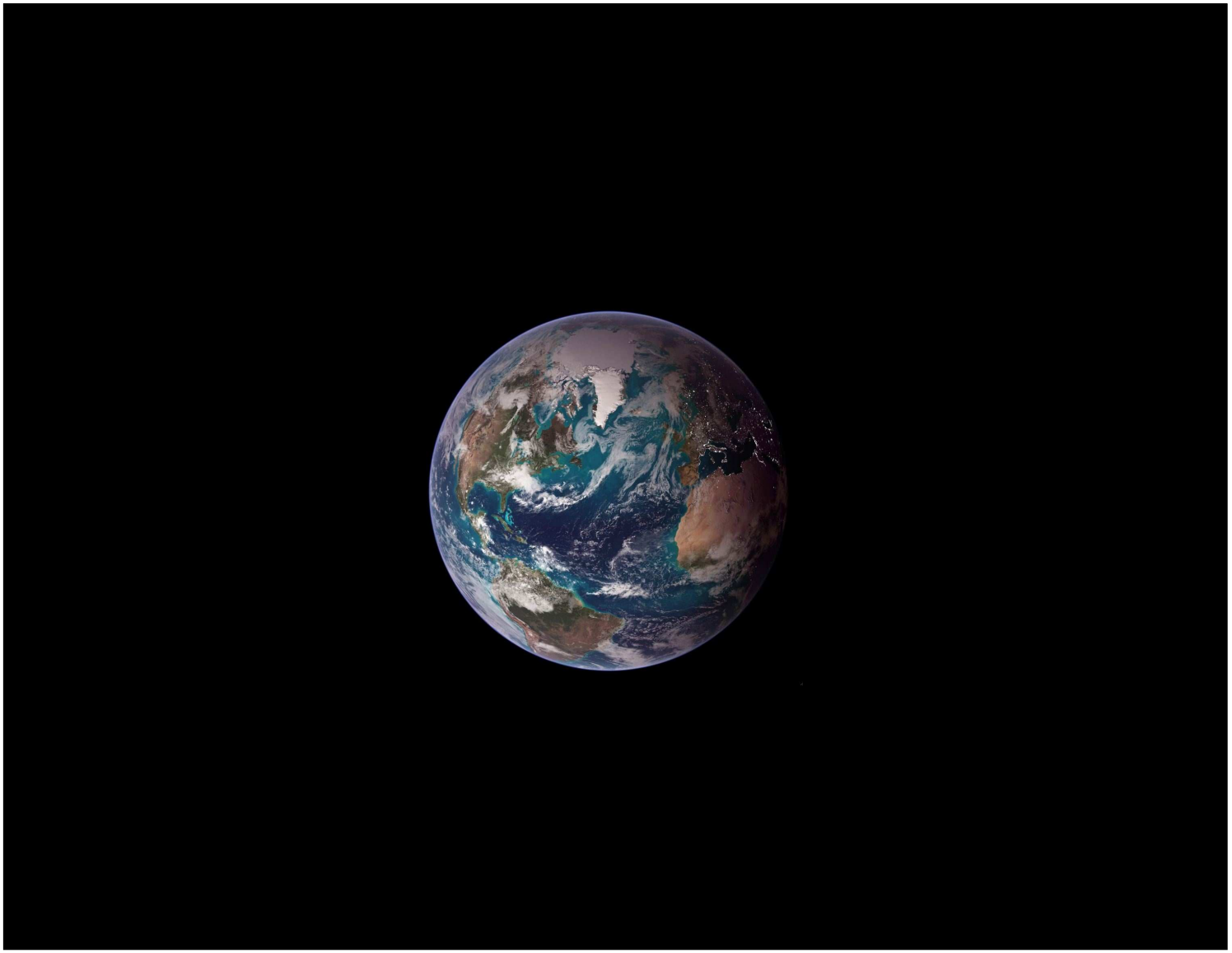
This is a progress report – on a career, not a research grant!

How Far Can You Travel?

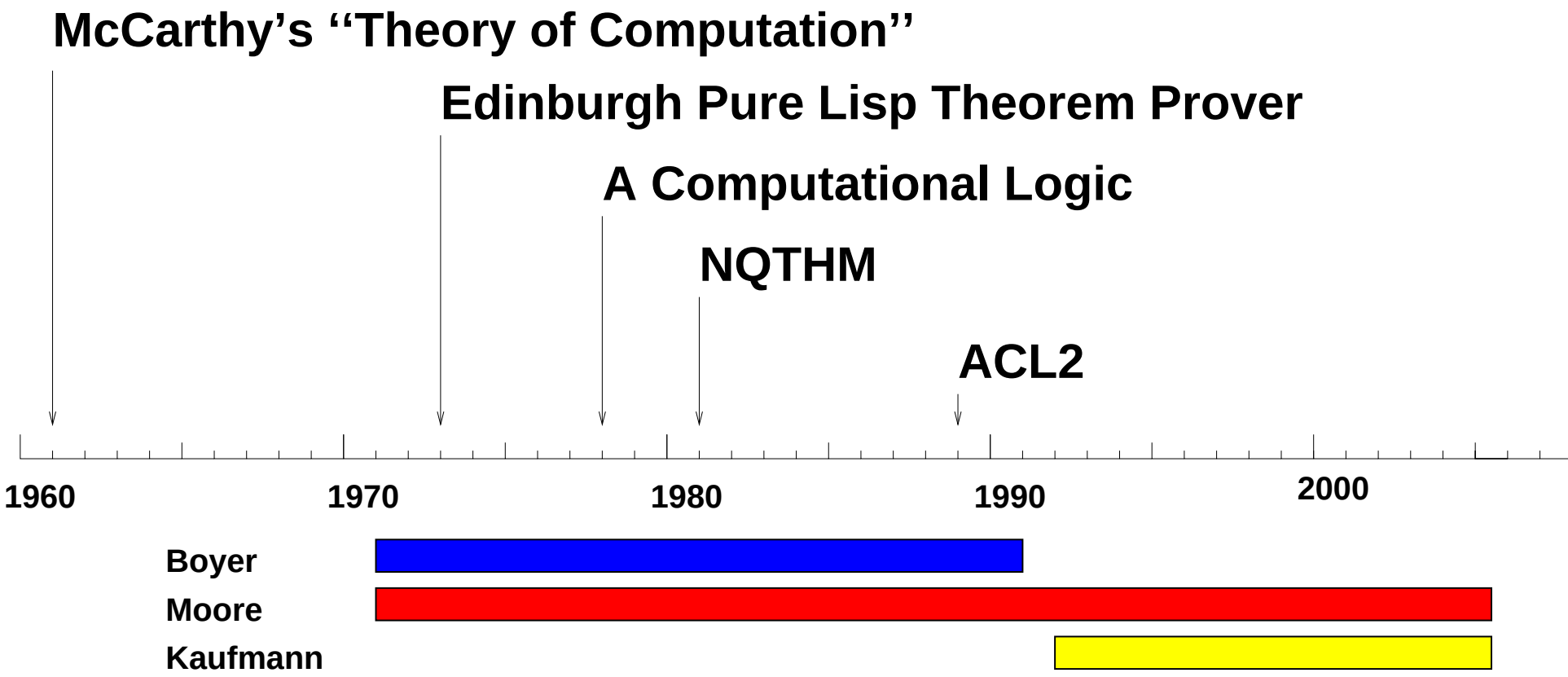
At 3 miles/hour, 8 hours per day, 5 days per week, for the duration of a typical research grant, you travel 12,500 miles.

In 40 years, you travel 250,000 miles.

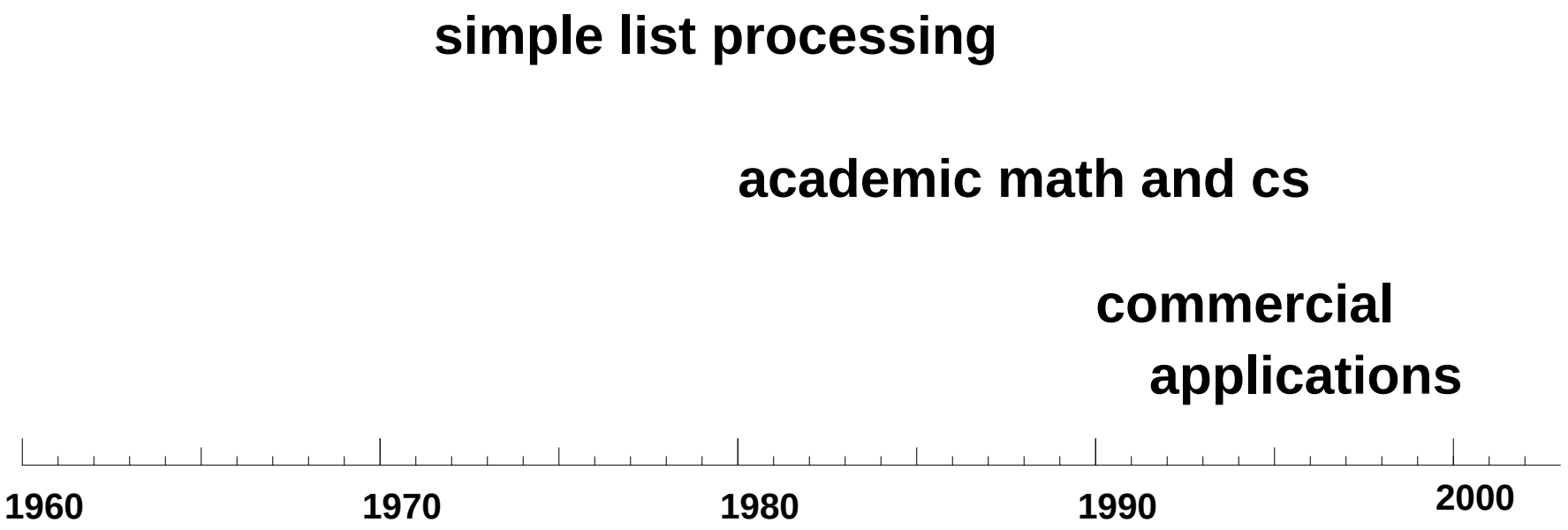




Boyer-Moore Project



Theorems Proved



Theorems Proved: 1970s

- ap is associative:

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

$\forall a \forall b \forall c$

ap(ap(a,b),c)

$=$

ap(a,ap(b,c)).

A Few Axioms

$\texttt{t} \neq \texttt{nil}$

$\texttt{x} = \texttt{nil} \rightarrow (\texttt{if x y z}) = \texttt{z}$

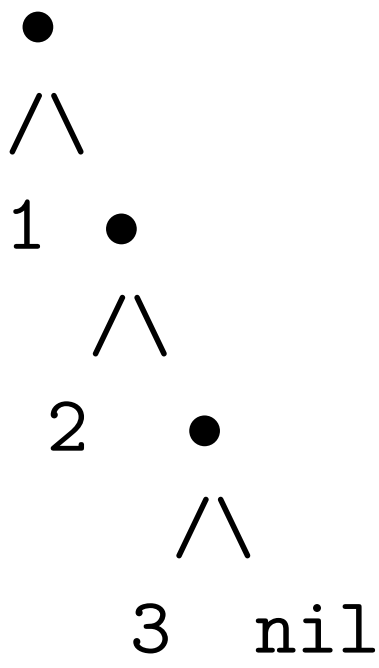
$\texttt{x} \neq \texttt{nil} \rightarrow (\texttt{if x y z}) = \texttt{y}$

$(\texttt{car } (\texttt{cons x y})) = \texttt{x}$

$(\texttt{cdr } (\texttt{cons x y})) = \texttt{y}$

```
(endp nil) = t

(endp (cons x y)) = nil
```

Ordered Pairs as Lists




```
(cons 1 (cons 2 (cons 3 nil))) = '(1 2 3)
```

```
(defun ap (x y)
  (if (endp x)
      y
      (cons (car x)
            (ap (cdr x) y))))

(ap '(1 2 3) '(4 5 6))
= '(1 2 3 4 5 6)
```

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

Proof: by induction on a.

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

Proof: by induction on a.

Base Case: (endp a).

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

Proof: by induction on a.

Base Case: (endp a).

```
(equal (ap b c)
       (ap a (ap b c)))
```

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

Proof: by induction on a.

Base Case: (endp a).

```
(equal (ap b c)
       (ap a (ap b c)))
```

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

Proof: by induction on a.

Base Case: (endp a).

```
(equal (ap b c)
       (ap b c))
```

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

Proof: by induction on a.

Base Case: (endp a).

```
(equal (ap b c)
       (ap b c))
```

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

Proof: by induction on a.

Base Case: (endp a).

T

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

Proof: by induction on a.

Induction Step: (not (endp a)).

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

```
(equal (ap (ap (cdr a) b) c)        ; Ind Hyp
       (ap (cdr a) (ap b c)))
```

Proof: by induction on a.

Induction Step: (not (endp a)).

```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

```
(equal (ap (ap (cdr a) b) c)      ; Ind Hyp
       (ap (cdr a) (ap b c)))
```

Proof: by induction on a.

Induction Step: (not (endp a)).

```
(equal (ap (cons (car a)
                 (ap (cdr a) b)) c)
       (ap a (ap b c)))
```

```
(equal (ap (ap (cdr a) b) c)      ; Ind Hyp
       (ap (cdr a) (ap b c)))
```

Proof: by induction on a.

Induction Step: (not (endp a)).

```
(equal (ap (cons (car a)
                 (ap (cdr a) b)) c)
       (ap a (ap b c)))
```

```
(equal (ap (ap (cdr a) b) c)        ; Ind Hyp
       (ap (cdr a) (ap b c)))
```

Proof: by induction on a.

Induction Step: (not (endp a)).

```
(equal (cons (car a)
             (ap (ap (cdr a) b) c))
       (ap a (ap b c)))
```

```
(equal (ap (ap (cdr a) b) c)        ; Ind Hyp
       (ap (cdr a) (ap b c)))
```

Proof: by induction on a.

Induction Step: (not (endp a)).

```
(equal (cons (car a)
             (ap (ap (cdr a) b) c))
       (ap a (ap b c)))
```

```
(equal (ap (ap (cdr a) b) c)        ; Ind Hyp
       (ap (cdr a) (ap b c)))
```

Proof: by induction on a.

Induction Step: (not (endp a)).

```
(equal (cons (car a)
             (ap (ap (cdr a) b) c))
       (cons (car a)
             (ap (cdr a) (ap b c))))
```

```
(equal (ap (ap (cdr a) b) c)        ; Ind Hyp
       (ap (cdr a) (ap b c)))
```

Proof: by induction on a.

Induction Step: (not (endp a)).

```
(equal (cons (car a)
             (ap (ap (cdr a) b) c))
       (cons (car a)
             (ap (cdr a) (ap b c))))
```

```
(equal (ap (ap (cdr a) b) c)        ; Ind Hyp
       (ap (cdr a) (ap b c)))
```

Proof: by induction on a.

Induction Step: (not (endp a)).

```
(equal
        (ap (ap (cdr a) b) c)

        (ap (cdr a) (ap b c)))
```

```
(equal (ap (ap (cdr a) b) c)        ; Ind Hyp
       (ap (cdr a) (ap b c)))
```

Proof: by induction on a.

Induction Step: (not (endp a)).

```
(equal (ap (ap (cdr a) b) c)
       (ap (cdr a) (ap b c)))
```

```
(equal (ap (ap (cdr a) b) c)        ; Ind Hyp
       (ap (cdr a) (ap b c)))
```

Proof: by induction on a.

Induction Step: (not (endp a)).

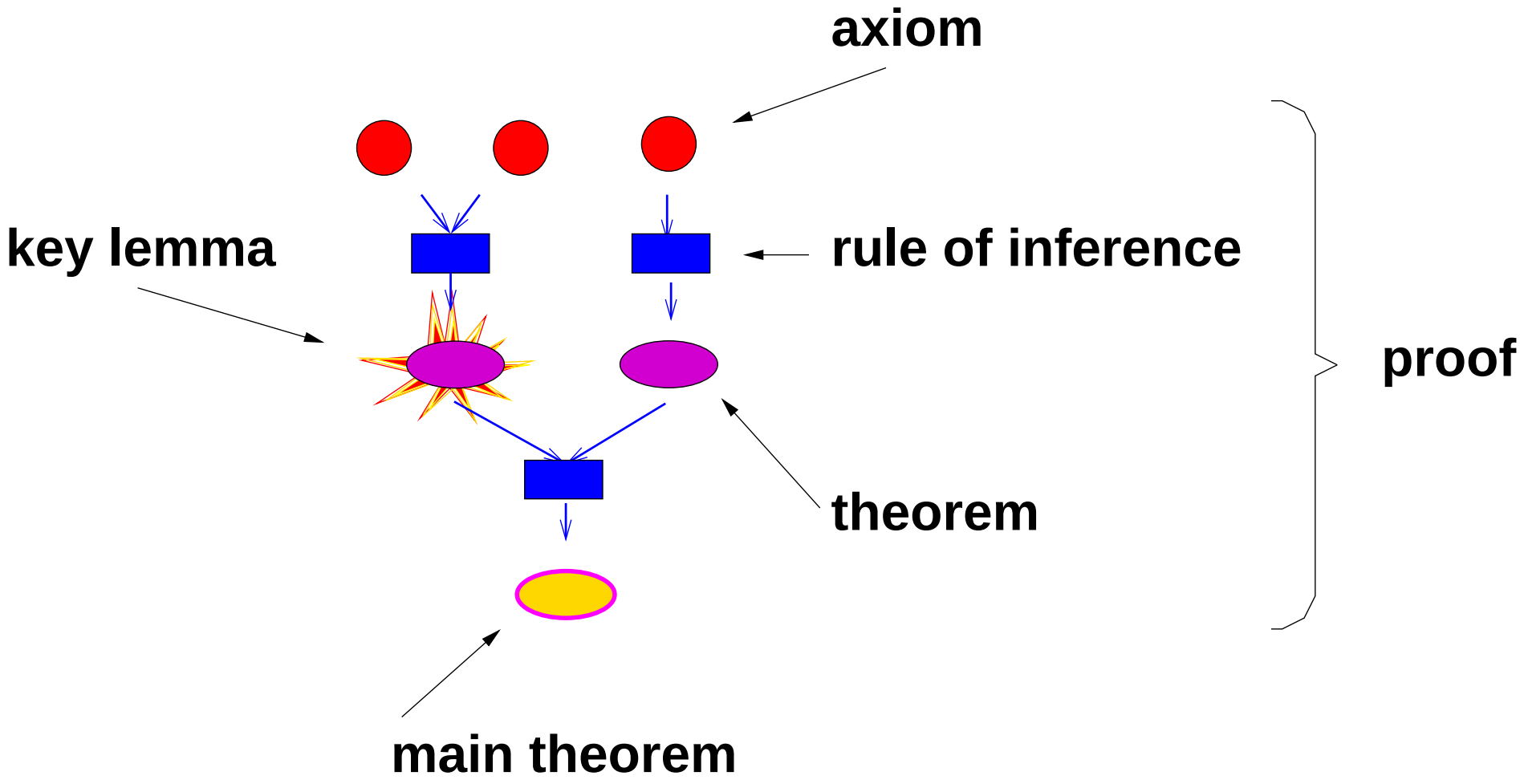
```
(equal (ap (ap (cdr a) b) c)
       (ap (cdr a) (ap b c)))
```

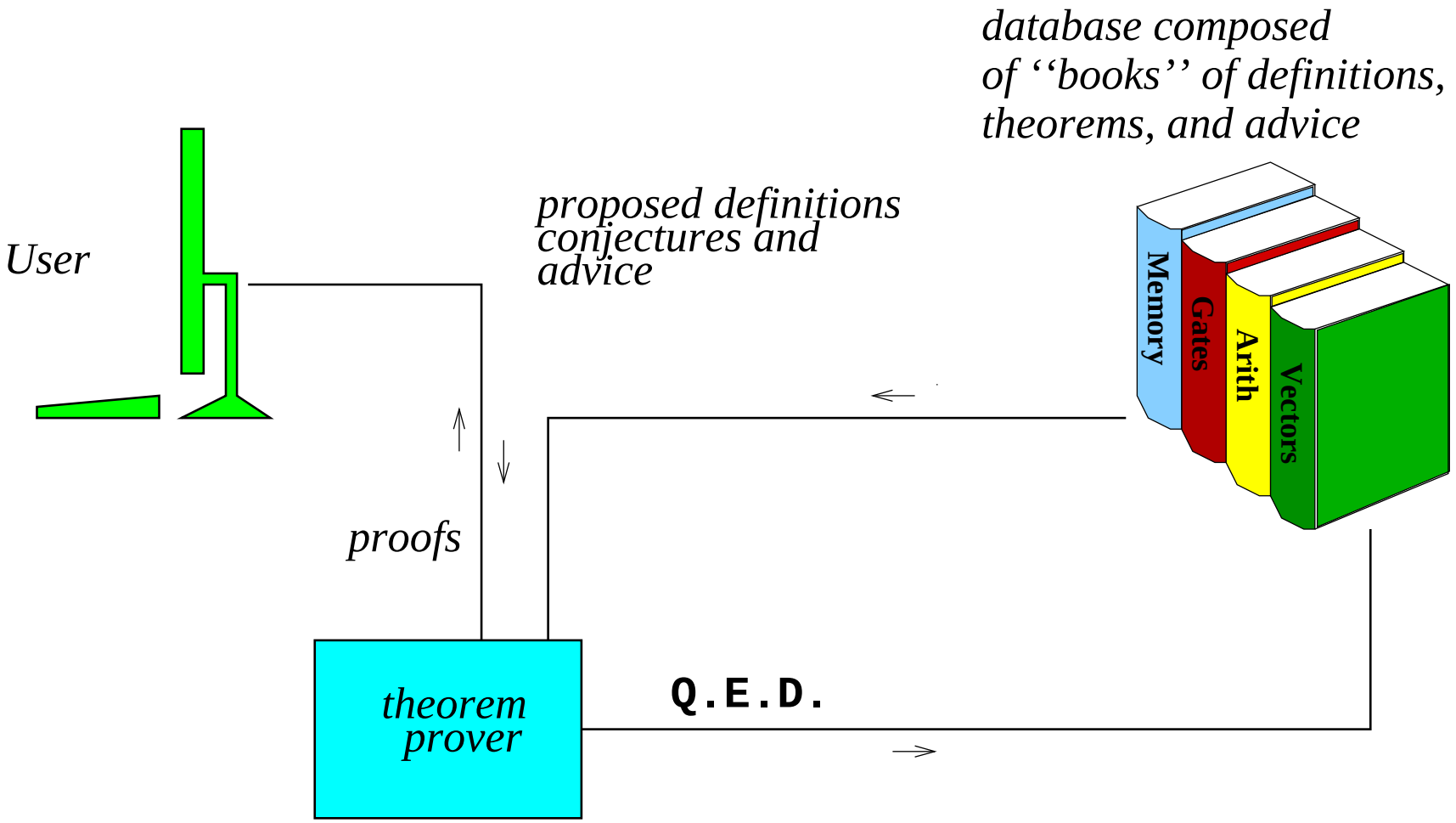
```
(equal (ap (ap a b) c)
       (ap a (ap b c)))

Proof: by induction on a.

Induction Step: (not (endp a)).
T
```

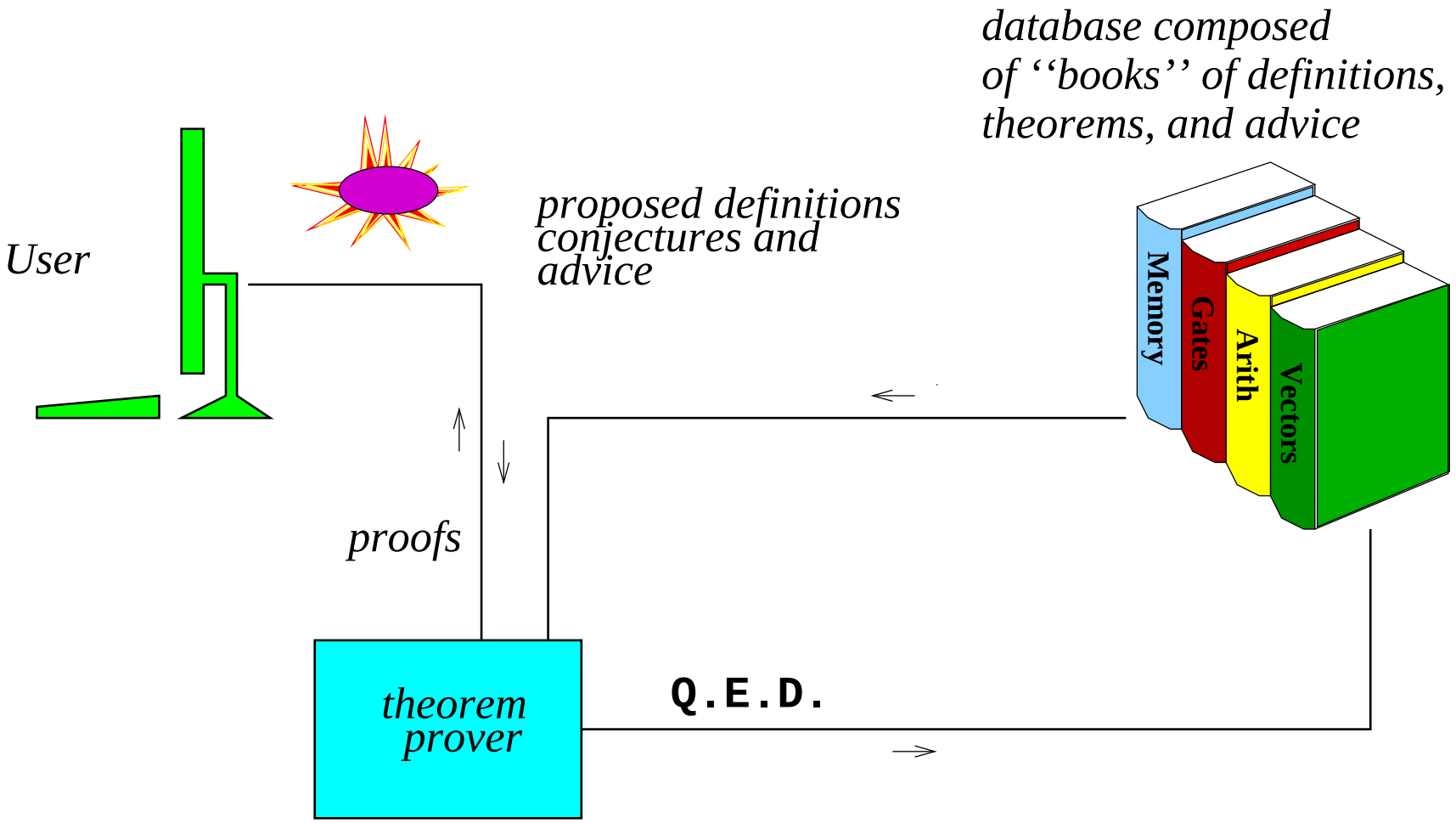
```
(equal (ap (ap a b) c)
       (ap a (ap b c)))
```

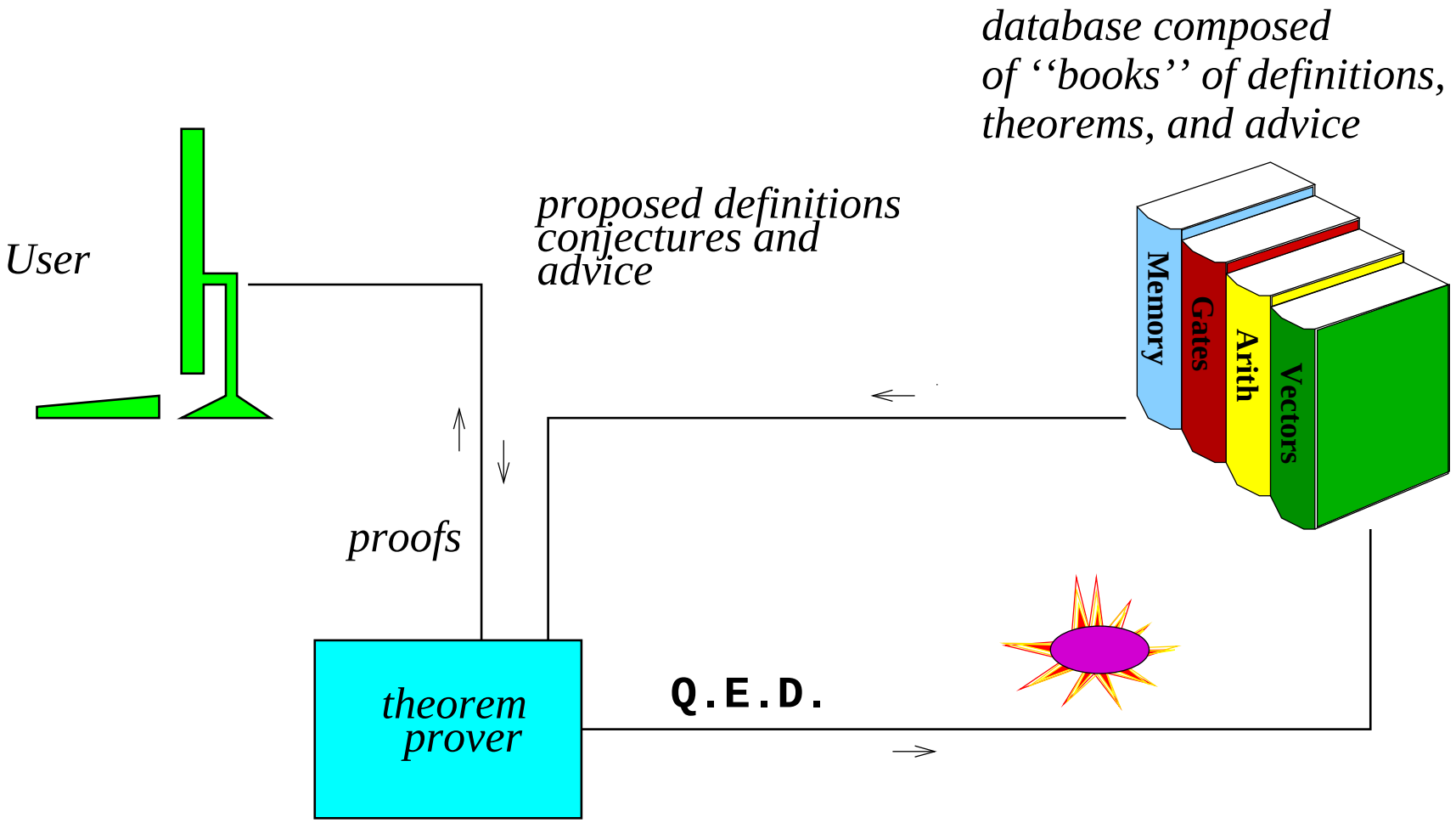
Proof: by induction on a.

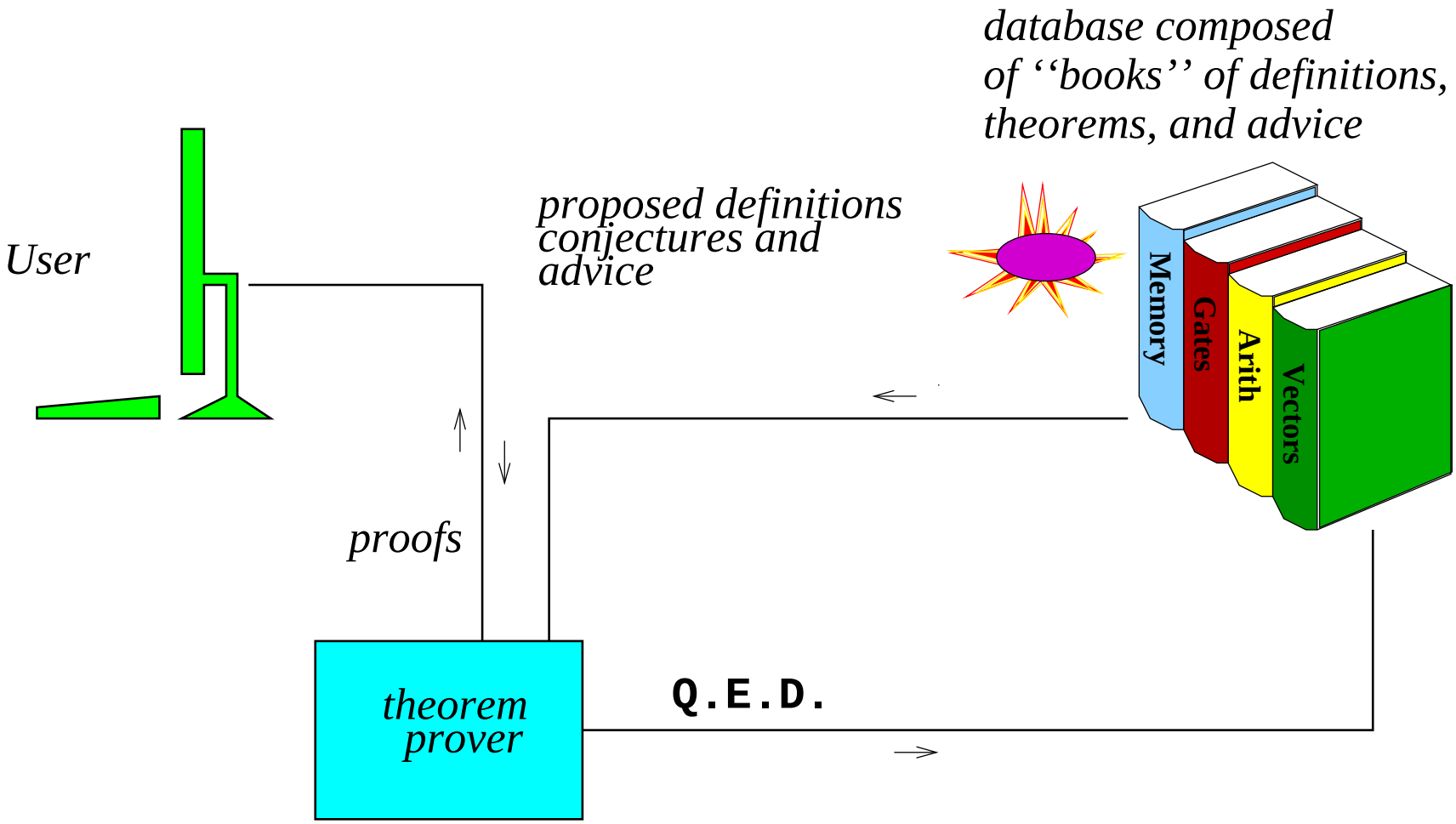
Q.E.D.


axiom
key lemma
rule of inference
proof
theorem
main theorem


database composed
of ‘‘books’’ of definitions,
theorems, and advice
User
proposed definitions
conjectures and
advice
Memory
Gates
Arith
Vectors
proofs
theorem
prover
Q.E.D.


database composed
of ‘‘books’’ of definitions,
theorems, and advice
User
proposed definitions
conjectures and
advice
Memory
Gates
Arith
Vectors
proofs
theorem
prover
Q.E.D.


database composed
of ‘‘books’’ of definitions,
theorems, and advice
User
proposed definitions
conjectures and
advice
Memory
Gates
Arith
Vectors
proofs
theorem
prover
Q.E.D.

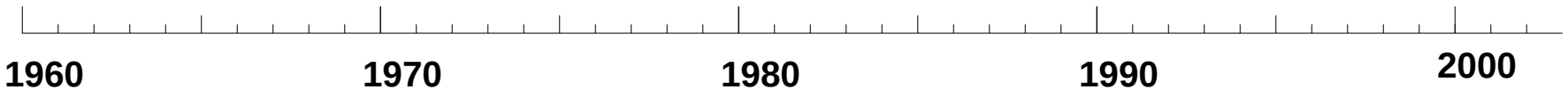

database composed
of ‘‘books’’ of definitions,
theorems, and advice
User
proposed definitions
conjectures and
advice
Memory
Gates
Arith
Vectors
proofs
theorem
prover
Q.E.D.

ACL2 Demo 1

simple list processing

academic math and cs

commercial applications

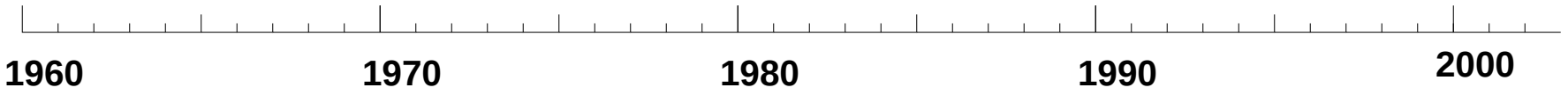


Theorems Proved

simple list processing

academic math and cs

commercial applications



1980s Academic Math

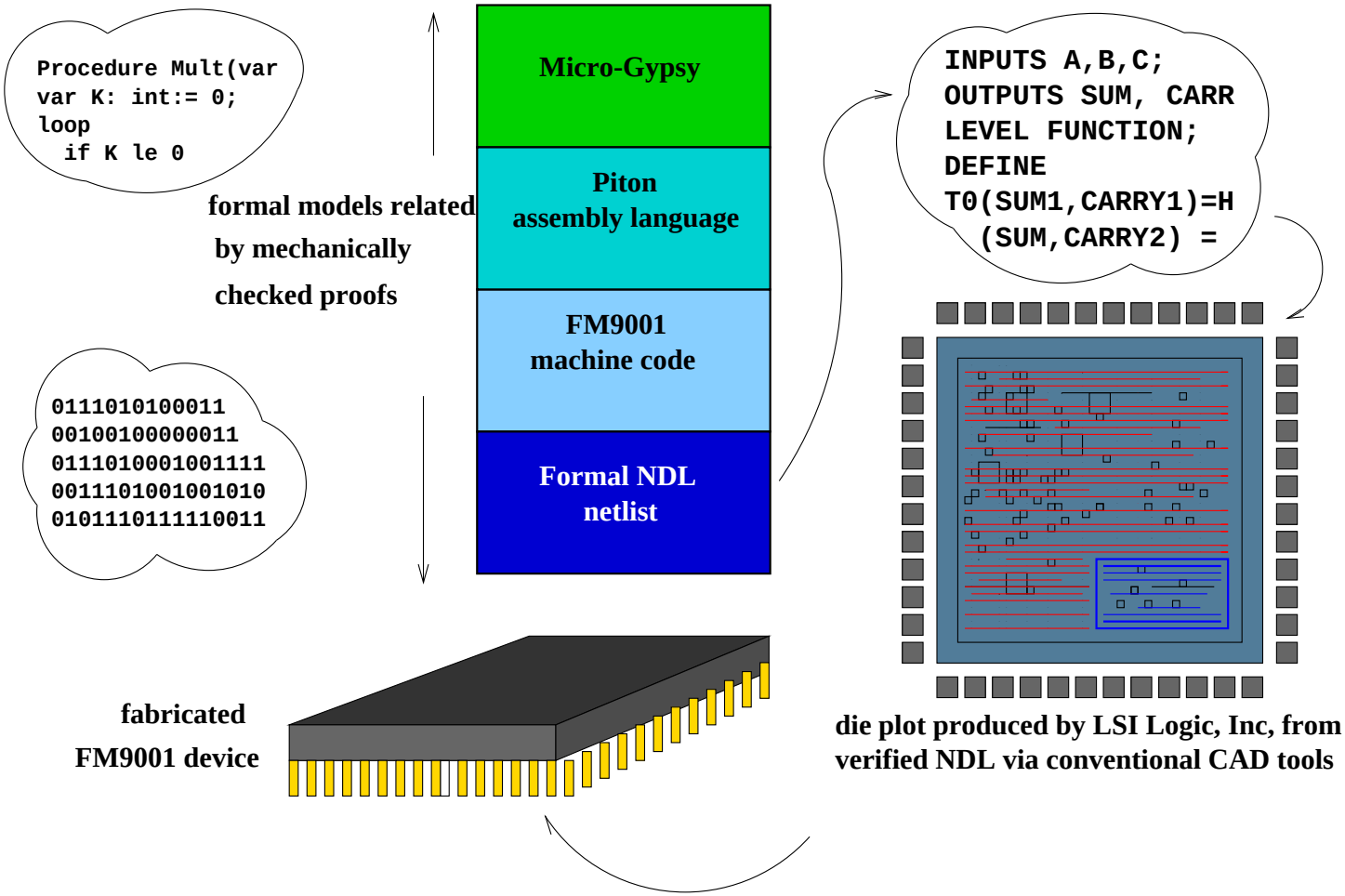
- undecidability of the halting problem (18 lemmas)
- invertibility of RSA encryption (172 lemmas)

- Gauss' law of quadratic reciprocity
  [Russinoff]
  (348 lemmas)

- Gödel's First Incompleteness Theorem
  [Shankar]
  (1741 lemmas)

1980s Academic CS

- The CLInc Verified Stack:
  - microprocessor: gates to machine code [Hunt]
  - assembler-linker-loader (3326 lemmas)
  - compilers [Young, Flatau]
  - operating system [Bevier]

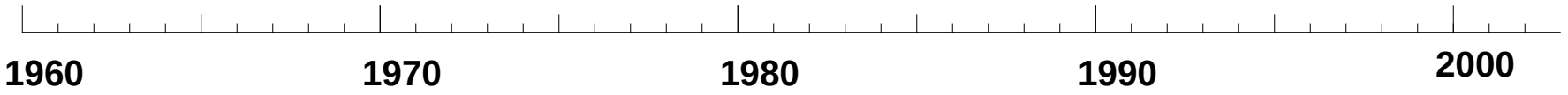

Procedure Mult(var
var K: int:= 0;
loop
if K le 0
formal models related
by mechanically
checked proofs
Micro-Gypsy
Piton
assembly language
FM9001
machine code
Formal NDL
netlist
0111010100011
00100100000011
0111010001001111
0011101001001010
0101110111110011
INPUTS A,B,C;
OUTPUTS SUM, CARR
LEVEL FUNCTION;
DEFINE
T0(SUM1,CARRY1)=H
(SUM,CARRY2) =
fabricated
FM9001 device
die plot produced by LSI Logic, Inc, from
verified NDL via conventional CAD tools

Theorems Proved

simple list processing

academic math and cs

commercial applications



1990s

- Motorola 68020 and Berkeley C String Library (Yu)
- Motorola CAP DSP (Brock)
- FDIV on AMD K5 (Moore-Kaufmann-Lynch)

An elusive circuitry error is causing a chip used in millions of computers to generate inaccurate results

— *NY Times, "Circuit Flaw Causes Pentium Chip to Miscalculate, Intel Admits," Nov 11, 1994*

Intel Corp. last week took a $475 million write-off to cover costs associated with the divide bug in the Pentium microprocessor's floating-point unit — *EE Times, Jan 23, 1995*

AMD K5 Algorithm FDIV$(p, d, mode)$

1.	sd_0	$= \text{lookup}(d)$	`[exact 17 8]`
2.	d_r	$= d$	`[away 17 32]`
3.	sdd_0	$= sd_0 \times d_r$	`[away 17 32]`
4.	sd_1	$= sd_0 \times \text{comp}(sdd_0, 32)$	`[trunc 17 32]`
5.	sdd_1	$= sd_1 \times d_r$	`[away 17 32]`
6.	sd_2	$= sd_1 \times \text{comp}(sdd_1, 32)$	`[trunc 17 32]`
...	...	= ...	...
29.	q_3	$= sd_2 \times ph_3$	`[trunc 17 24]`
30.	qq_2	$= q_2 + q_3$	`[sticky 17 64]`
31.	qq_1	$= qq_2 + q_1$	`[sticky 17 64]`
32.	$fdiv$	$= qq_1 + q_0$	$mode$

The Futility of Testing

If AMD builds this, will it work?

A bug in this design could cost AMD hundreds of millions of dollars.

To test all possible combinations on the fastest machine in the world today would take over 500 billion billion billion (556×10^{27}) years!

The Formal Model of the Code

```
(defun FDIV (p d mode)
 (let*
   ((sd0  (eround (lookup d)              '(exact  17  8)))
    (dr   (eround d                       '(away   17 32)))
    (sdd0 (eround (* sd0 dr)              '(away   17 32)))
    (sd1  (eround (* sd0 (comp sdd0 32)) '(trunc  17 32)))
    (sdd1 (eround (* sd1 dr)              '(away   17 32)))
    (sd2  (eround (* sd1 (comp sdd1 32)) '(trunc  17 32)))
    ...
    (qq2  (eround (+ q2 q3)               '(sticky 17 64)))
    (qq1  (eround (+ qq2 q1)              '(sticky 17 64)))
    (fdiv (round   (+ qq1 q0)             mode)))
   (or (first-error sd0 dr sdd0 sd1 sdd1 ... fdiv)
       fdiv)))
```

The K5 FDIV Theorem (1200 lemmas)

```
(defthm FDIV-divides
  (implies (and (floating-point-numberp p 15 64)
                (floating-point-numberp d 15 64)
                (not (equal d 0))
                (rounding-modep mode))
           (equal (FDIV p d mode)
                  (round  (/ p d) mode))))
```

(by Moore, Lynch and Kaufmann, in 1995, *before the K5 was fabricated*)

AMD Athlon 1997

All elementary floating-point operations,
`FADD`, `FSUB`, `FMUL`, `FDIV`, and `FSQRT`,
on the AMD Athlon were

- specified in ACL2 to be IEEE compliant,
- proved to meet their specifications, and
- the proofs were checked mechanically.

1990s

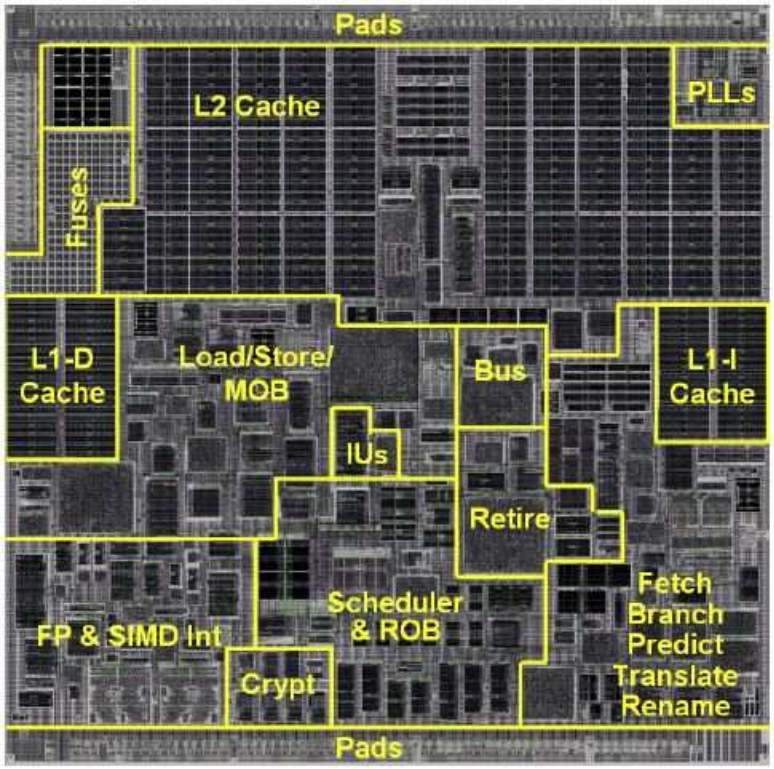
- ...
- AMD Athlon floating point (Russinoff-Flatau)
- Rockwell Collins microarchitectural equivalence (Wilding, Greve)
- Rockwell Collins / aJile Systems JEM1 (Hardin-Greve-Wilding)

2000s

- IBM Power4 divide and square root (Sawada)
- Rockwell Collins AAMP7 Separation Kernel Microcode (Greve, et al)
- Rockwell Collins/Green Hills OS Kernel (Greve, et al)

- Centaur (VIA) media unit (Hunt and Swords)
- Sun Microsystems JVM (class loading and bytecode verification) (Liu)
- A Verified Theorem Prover
- ...

Centaur (VIA) Nano 64-bit x86 CPU



- $\sim$20 FP operations verified
- LOC: FADD is 33,700 lines of Verilog
- Proof Times: few secs – few hrs
- Memory: 1 – 80 GB
- design and (automatic) proofs change daily
- hard-to-discover bugs found (e.g., 4 tests out of 2^{160} would have failed)

JVM Operational Semantics

M6 is an ACL2 model of the Sun JVM.

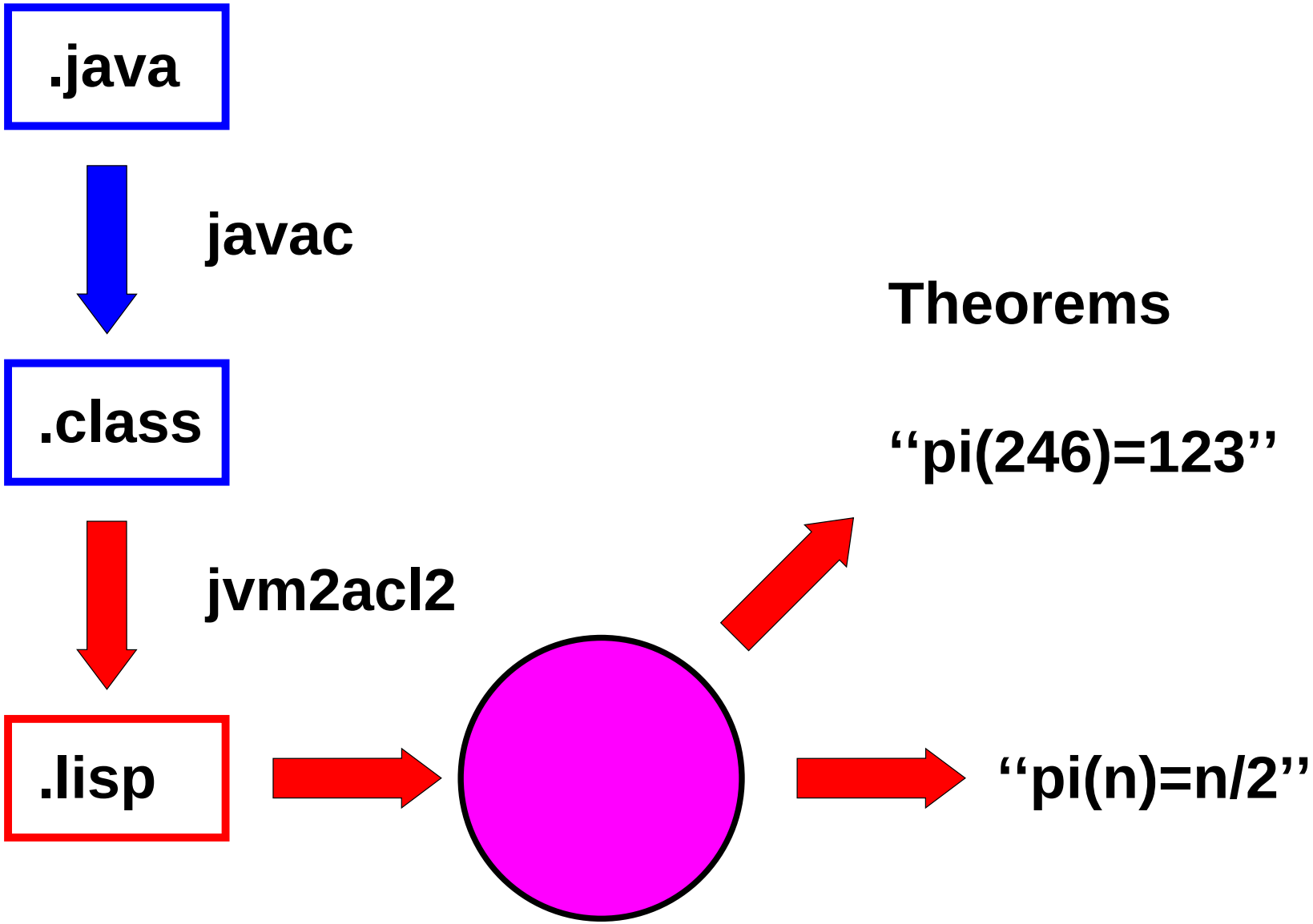
(M6 is a JVM bytecode interpreter written in pure Lisp.)

It executes most J2ME Java programs (except those with significant I/O or floating-point).

M6 was created by Hanbing Liu (now at AMD) with support from Sun Microsystems.

M6 supports

- all data types (except floats),
- multi-threading,
- dynamic class loading,
- class initialization, and
- synchronization via monitors.


.java
javac
.class
jvm2acl2
.lisp
Theorems
“pi(246)=123”
“pi(n)=n/2”

The state we model includes an "external class table" where classes reside until they are loaded.

We have translated the entire Sun CLDC API library implementation into our external representation (672 methods in 87 classes).

The model is 160 pages of ACL2.

ACL2 Demo 2

How Do We Know ACL2 is Sound?

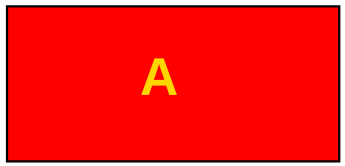
"Trust us!" – *Kaufmann and Moore*

Obviously, we would like to prove it correct.

But with what prover?

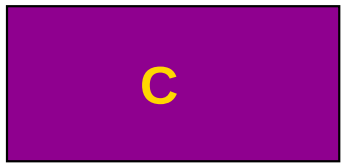
Jared Davis' Dissertation

ACL2−like theorem prover



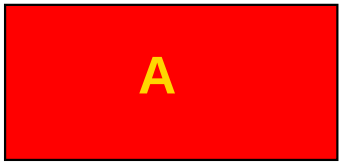
Small, Simple, Trusted

Proof Checker



Jared Davis' Dissertation

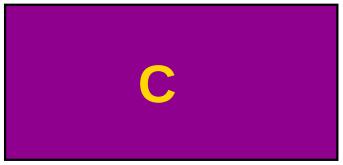
ACL2−like theorem prover



Correctness: If A proves Φ, then there is a proof of Φ that can be checked by C.

Small, Simple, Trusted

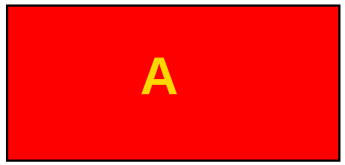
Proof Checker



Goal: Proof Correctness with C!

Jared Davis' Dissertation

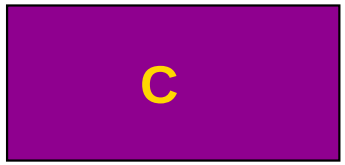
ACL2–like theorem prover



Correctness: If A proves Φ, then there is a proof of Φ that can be checked by C.

Small, Simple, Trusted

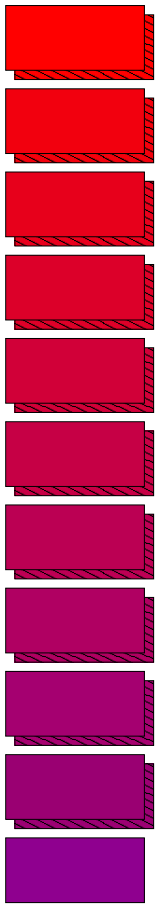
Proof Checker



Goal: Proof Correctness with C!

Idea: Use A to prove Correctness, recover the claimed proof, and check it with C.

Jared Davis' Stack "Milawa"



Level

11 Induction and other tactics

10 Conditional rewriting

9 Evaluation and unconditional rewriting

8 Audit trails (in prep for rewriting)

7 Case splitting

6 Factoring, splitting help

5 Assumptions and clauses

4 Miscellaneous ground work

3 Rules about primitive functions

2 Propositional reasoning

1 Primitive proof checker

Present - Why are we succeeding?

Reason 1: Our mathematical logic is an executable programming language.

- Many very efficient heavy-duty implementations
- Supported on many platforms

- Many independently provided programming/system development tools and environments.

Reason 2: We have invested 40 years

- supporting efficient execution,
- integrating a wide variety of proof techniques,
- engineering for industrial scale formulas, and
- developing reusable books.

Reason 3: We have chosen the right problems. In our applications, the models

- are bit- and cycle-accurate, not "toys",
- are useful as pre-fab simulation engines, and
- permit mathematical abstraction supported by proof.

Reason 4: Industry has no other alternative; their artifacts are too complicated to analyze accurately any other way.

Our Hypothesis

The "high cost" of formal methods

– to the extent the cost is high –

is a *historical anomaly* due to the fact that virtually every project formally recapitulates the past.

The use of mechanized formal methods will ultimately

- *decrease* time-to-market, and
- *increase* reliability.

Conclusion

Mechanical reasoning systems are changing the way complex digital artifacts are built.

Complexity not an argument *against* formal methods.

It is an argument *for* formal methods.

Next Time

How to Model Machines and Prove Theorems about Code

How to Drive ACL2

References

Computer-Aided Reasoning: An Approach, Kaufmann, Manolios, Moore, Kluwer Academic Publishers, 2000.

Computer-Aided Reasoning: ACL2 Case Studies, Kaufmann, Manolios, Moore (eds.), Kluwer Academic Publishers, 2000.

http://www.cs.utexas.edu/users/moore/acl2