Theorem Proving for Verification

the Early Days

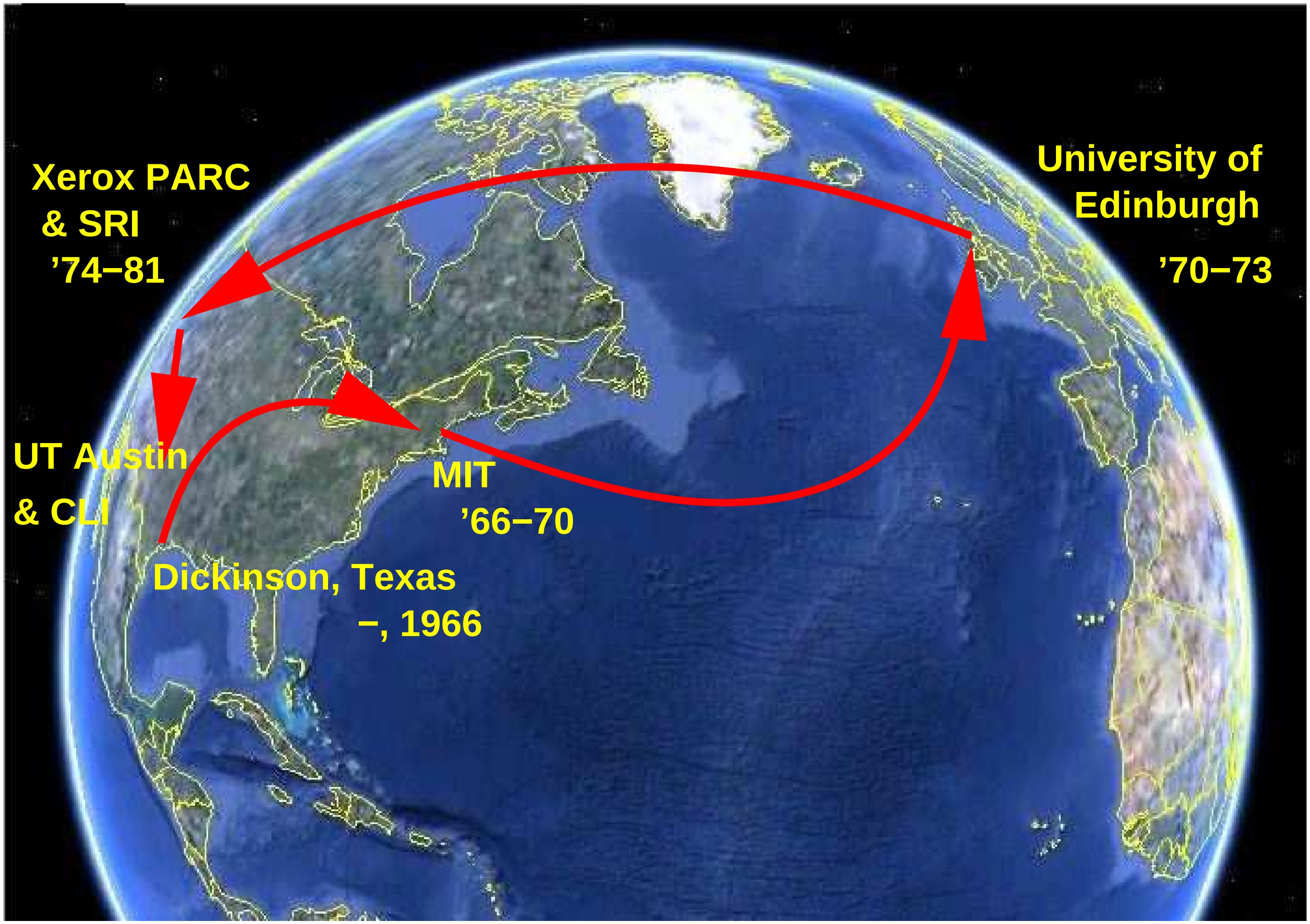
J Strother Moore
Department of Computer Sciences
University of Texas at Austin

Prologue

When I look at the state of our science today, I am amazed and proud of how far we've come and what is routinely possible today with mechanized verification.

But how did we get here?

A personal perspective on the journey



Hope Park Square, Edinburgh, 1970

Hope Park Square, Edinburgh, 1970

Rod Burstall
Donald Michie
Bernard Meltzer
Bob Kowalski
Pat Hayes

Hope Park Square, Edinburgh, 1970

Rod Burstall
Donald Michie
Bernard Meltzer
Bob Kowalski
Pat Hayes

Gordon Plotkin 1968

Hope Park Square, Edinburgh, 1970

Rod Burstall
Donald Michie
Bernard Meltzer
Bob Kowalski
Pat Hayes

Gordon Plotkin 1968
Mike Gordon 1970
J Moore 1970

Hope Park Square, Edinburgh, 1970

Rod Burstall
Donald Michie
Bernard Meltzer
Bob Kowalski
Pat Hayes

Gordon Plotkin 1968
Mike Gordon 1970
J Moore 1970

Bob Boyer 1971
Alan Bundy 1971

Hope Park Square, Edinburgh, 1970

Rod Burstall
Donald Michie
Bernard Meltzer
Bob Kowalski
Pat Hayes

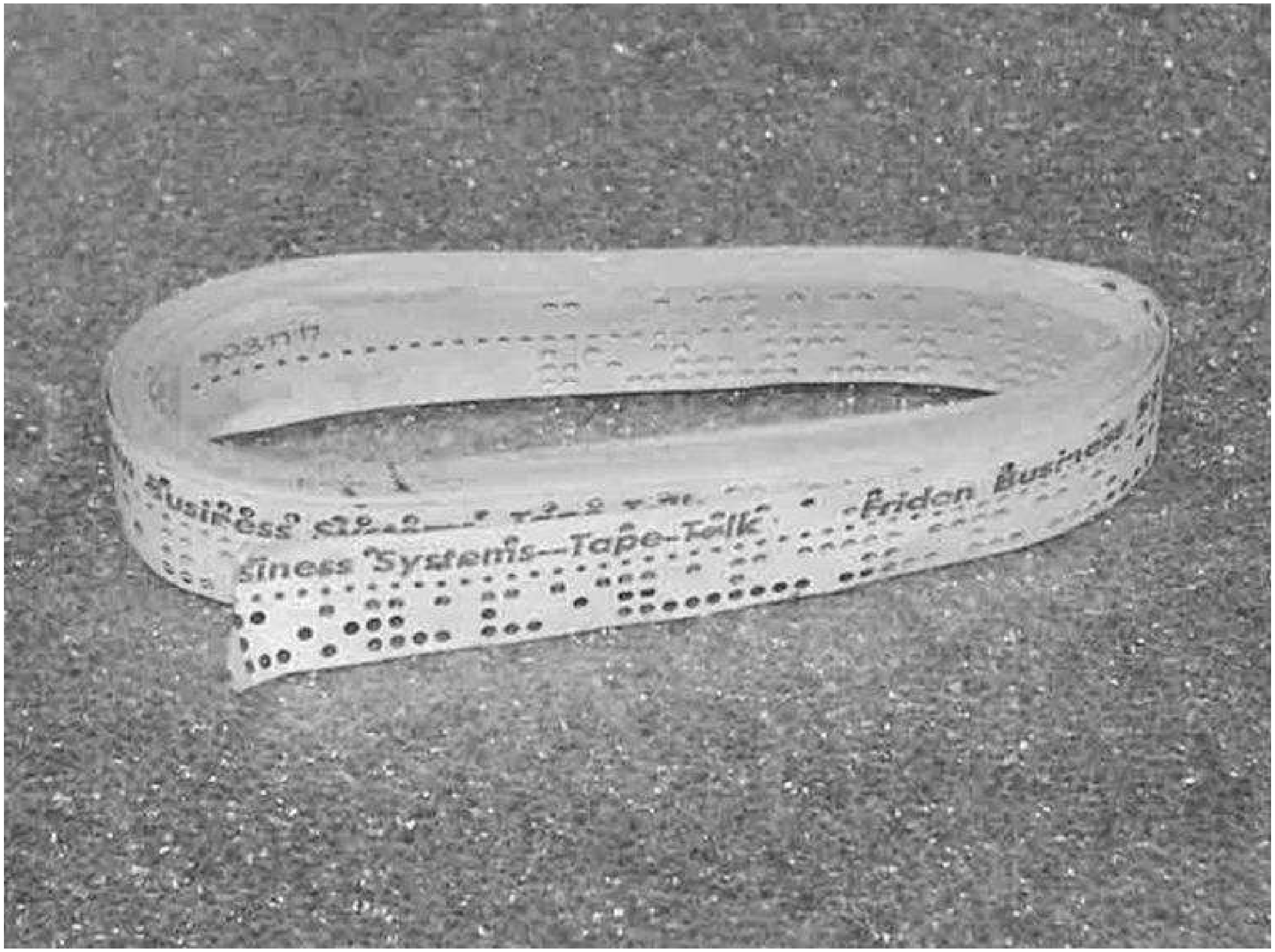
Gordon Plotkin 1968
Mike Gordon 1970
J Moore 1970

Bob Boyer 1971
Alan Bundy 1971

Robin Milner 1973

Our Computing Resources

64KB of RAM, paper tape input

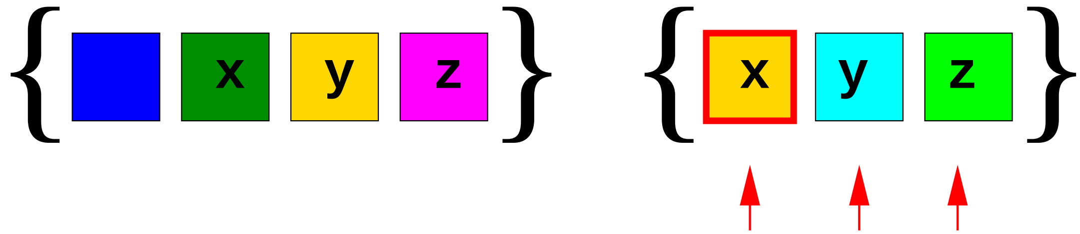

Friden Business
siness Systems--Tape-Talk

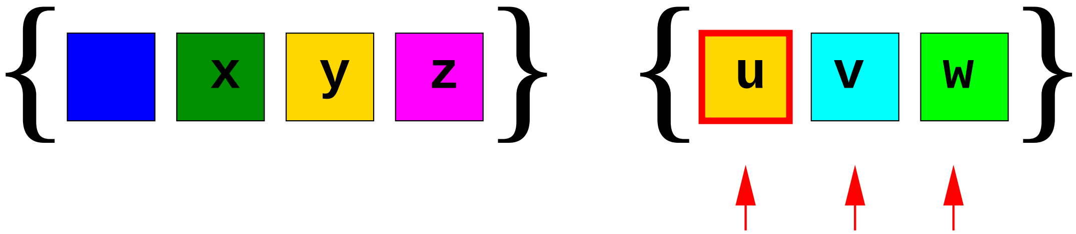
Instead of debugging a program, one should prove that it meets its specifications, and this proof should be checked by a computer program.

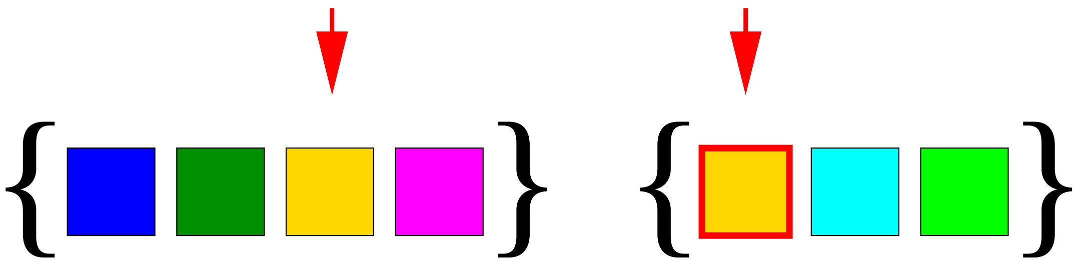
— *John McCarthy, "A Basis for a Mathematical Theory of Computation," 1961*

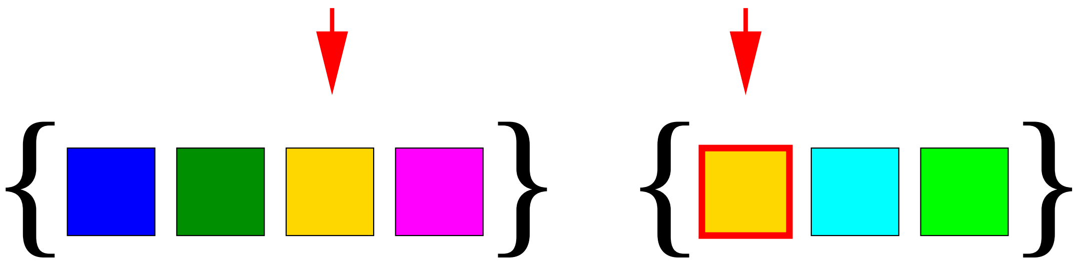
Theorem Proving in 1970

resolution: a complete, first-order, uniform proof-procedure based on unification and cut


{ x y z } { x y z }

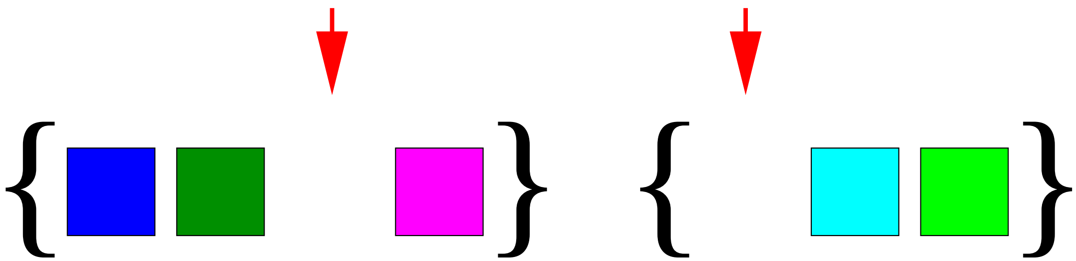

x
y
z
u
v
w





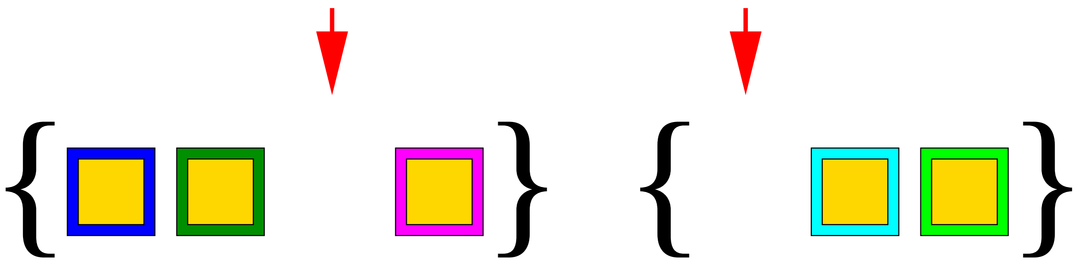
```
{x ← (F u (G v)),
 u ← (H z)}
```

Most General Unifier



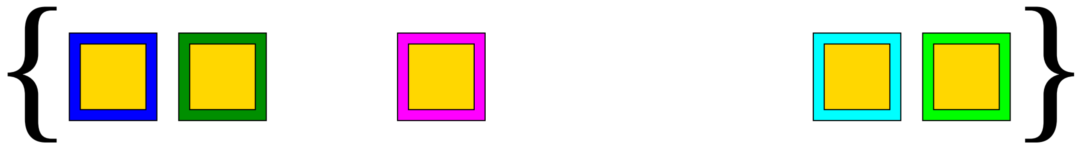
```
{x ← (F u (G v)),
 u ← (H z)}
```

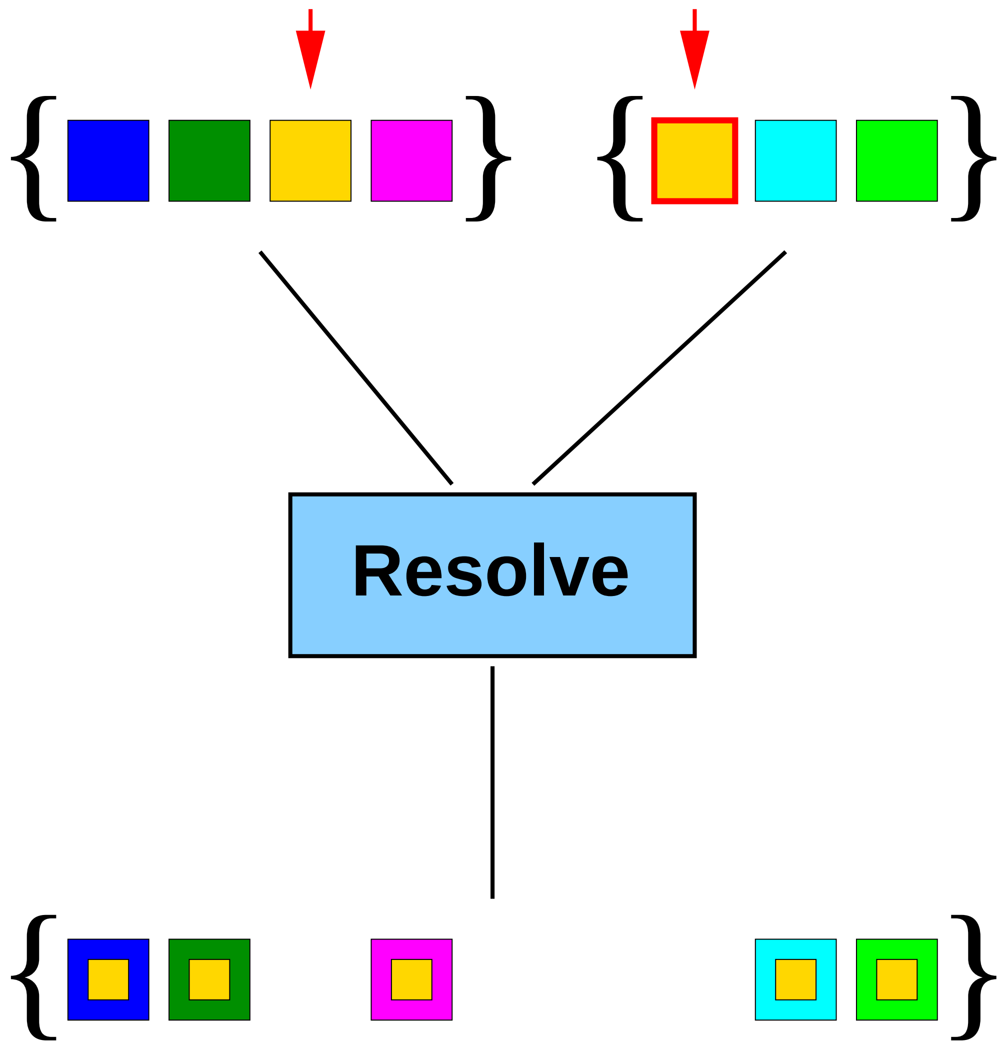
Most General Unifier

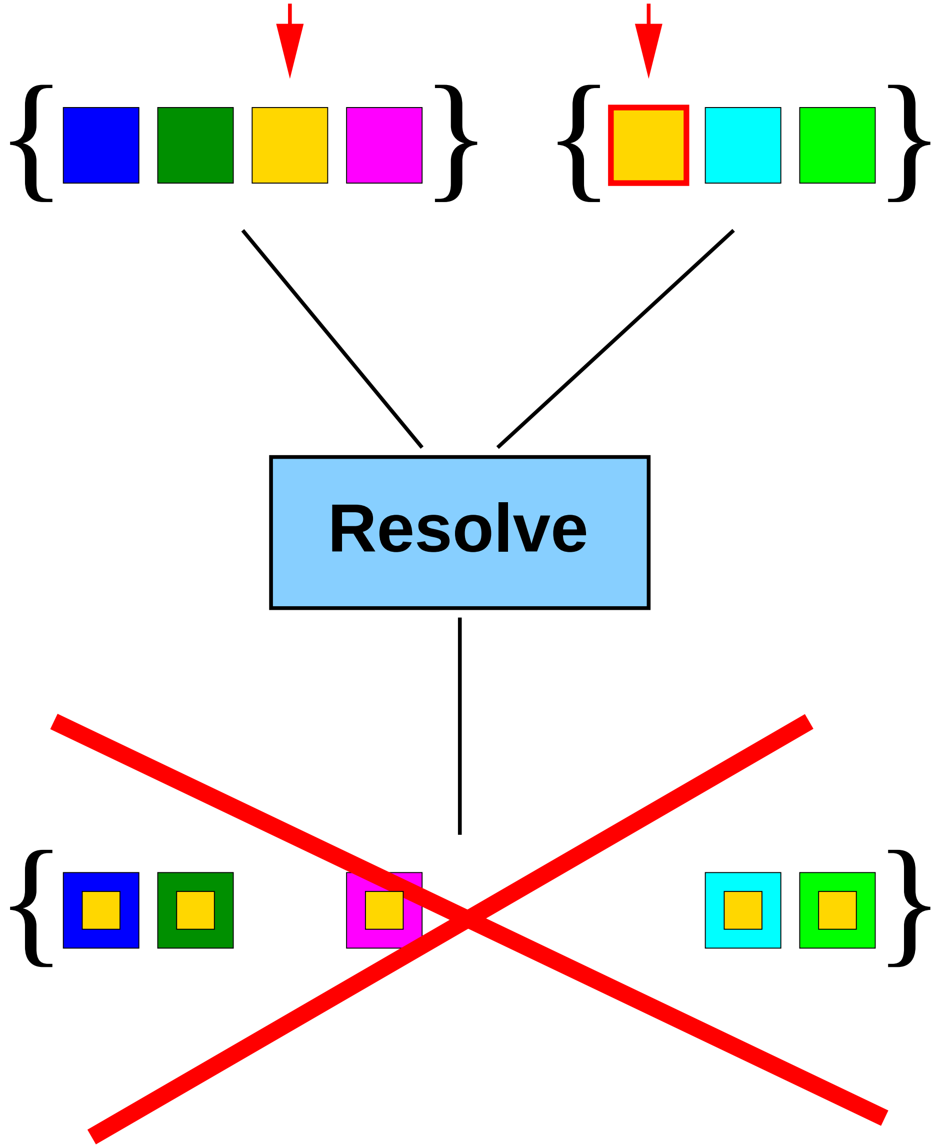


```
{x ←(F u (G v)),
 u ←(H z)}
```

Most General Unifier

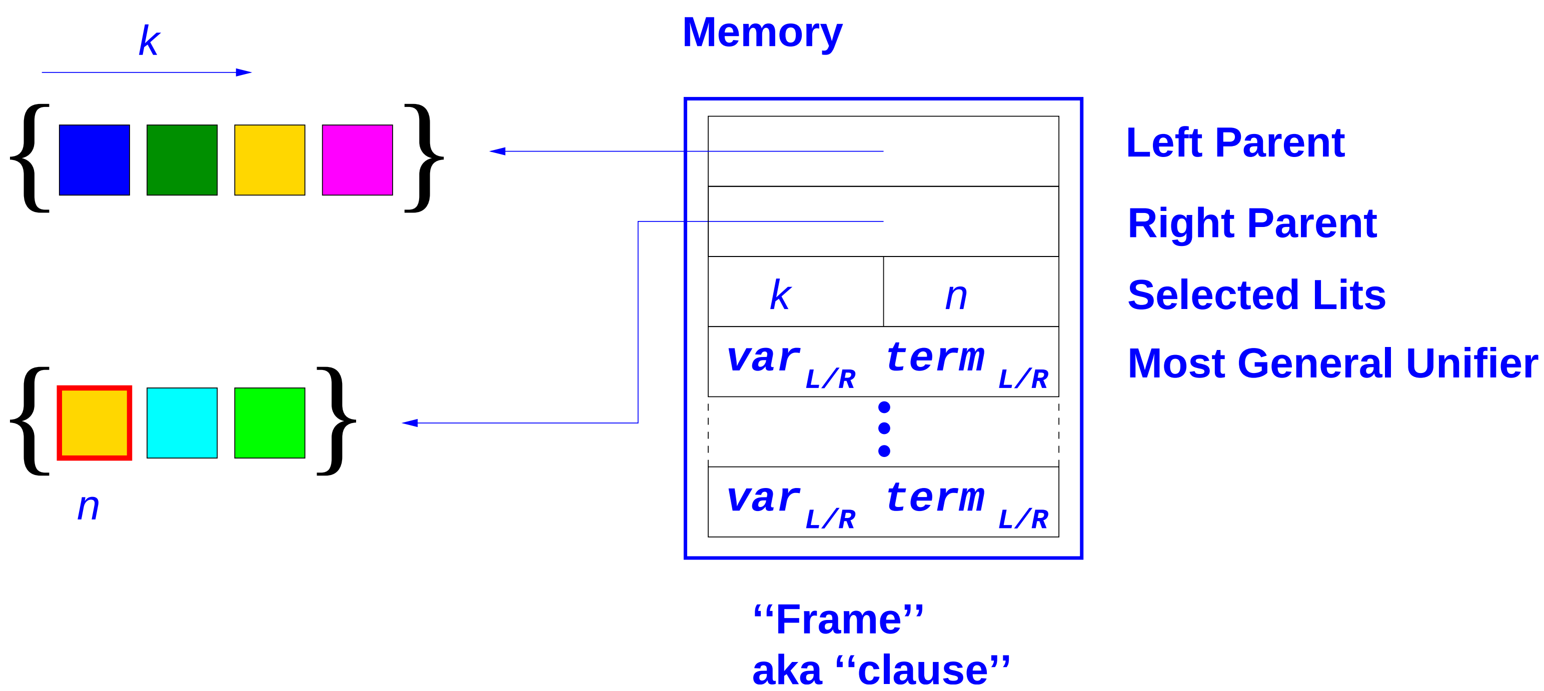



Resolve


Resolve

Structure Sharing

clause: a record of the two parents and binding environment



Structure Sharing

clauses are their own derivations

standardizing apart is implicit (free)

linear resolution can be done on a stack of frames

resolvents cost fixed space plus a “binding environment” all terms are specific instances of original ones

unifiers can be preprocessed

easy to attach pragmas (and other metadata) to variables and clauses

Such observations encouraged in Edinburgh the view that predicate calculus could be viewed as a programming language

But Boyer and I were interested in *computational logic*:

- a logic convenient for talking about computation

- a logic designed for computationally assisted proofs

So we invented a programming language that was integrated into this resolution framework

BAROQUE[1]

```
LEN1: (LENGTH NIL) -> 0;

LEN2: (LENGTH (CONS X Y)) -> Z
      WHERE
      (LENGTH Y) -> U;
      (ADD U 1) -> Z;
      END;
```

This language is called BAROQUE. It has several properties not found in traditional programming languages. Among these are: pattern directed invocation and return, backtracking, and the ability to run functions "backwards" (from results to arguments).

[1] *Computational Logic: Structure Sharing and Proof of Program Properties*, PhD Thesis, Moore, 1973. "Baroque" was named after a bizarre chess-like game taught to us by **Steve Crocker** at the Firbush Workshop 1972.

```
APP: (APP X Y) -> U
     WHERE
     (IF X
         (CONS (CAR X)
               (APP (CDR X) Y))
         Y) -> U;
     END;
```

We could prove such things as:

$\exists$ `X : (LENGTH (APP X NIL)) = 2`

```
(APP NIL X) = X
```

```
(MEMBER E (APP (CONS E A) B))
```

But we could not prove

```
(APP (APP A B) C) = (APP A (APP B C))

(LENGTH (APP A B)) = (+ (LENGTH A) (LENGTH B))
```

To prove these theorems the underlying mathematical logic must support

- recursion
- induction
- rewriting

Users lacking support for these techniques often added (inconsistent) axioms

Verification work in the 1970s was focused on programming language semantics

But to prove anything interesting about the *data* manipulated by programs, you need recursion, induction, and equality in the logic

We therefore abandoned resolution and set out to build a theorem prover specifically for a computational logic

6.3 Design Philosophy of the Program[2]

The program was designed to behave properly on simple functions. The overriding consideration was that it should be automatically able to prove theorems about simple LISP function[s] in the straightforward way we prove them.

[2] *Computational Logic: Structure Sharing and Proof of Program Properties*, PhD Thesis, Moore, 1973.

A Few Axioms

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

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

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

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

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

$\texttt{(endp nil)} = \texttt{t}$

$\texttt{(endp (cons x y))} = \texttt{nil}$

```
(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)
```

Proper Treatement of Definitions, 1972

To specify programs one needs to extend the logical theory by the introduction of new functions and predicates

But this should be done via conservative extension mechanisms, not the assumption of arbitrary axioms

Symbolic Evaluation, 1972

`(length (ap (cons e a) b))`

The key "proof technique" would be *rewriting* via symbolic evaluation

Symbolic Evaluation, 1972

```
(length (if (endp (cons e a))
            b
            (cons (car (cons e a))
                  (ap (cdr (cons e a)) b))))
```

Symbolic Evaluation, 1972

```
(length (if (endp (cons e a))
            b
            (cons (car (cons e a))
                  (ap (cdr (cons e a)) b))))
```

Symbolic Evaluation, 1972

```
(length (if NIL
            b
            (cons (car (cons e a))
                  (ap (cdr (cons e a)) b))))
```

Symbolic Evaluation, 1972

```
(length (if NIL
            b
            (cons (car (cons e a))
                  (ap (cdr (cons e a)) b))))
```

Symbolic Evaluation, 1972

```
(length (cons (car (cons e a))
              (ap (cdr (cons e a)) b)))
```

Symbolic Evaluation, 1972

```
(length (cons (car (cons e a))
              (ap (cdr (cons e a)) b)))
```

Symbolic Evaluation, 1972

```
(length (cons e
              (ap (cdr (cons e a)) b)))
```

Symbolic Evaluation, 1972

```
(length (cons e
               (ap (cdr (cons e a)) b)))
```

Symbolic Evaluation, 1972

```
(length (cons e
              (ap a b)))
```

Symbolic Evaluation, 1972

```
(length (cons e
                (ap a b)))
```

Symbolic Evaluation, 1972

```
(+ 1 (length (ap a b)))
```

Symbolic Evaluation, 1972

conditional rewriting (with recursive definitions and axioms)

`IF` as the main propositional connective

typing as theorem proving mechanism

Controlling Recursive Functions, 1972

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

Controlling Recursive Functions, 1972

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

Controlling Recursive Functions, 1972

```
(ap (if (endp a)
        b
        (cons (car a)
              (ap (cdr a) b)))
    c)
```

If `(cdr a)` is already in the problem, keep the expansion. Otherwise...

Recursion and Induction, 1972

`(ap (ap a b) c)`

Recursion and Induction, 1972

`(ap (ap a b) c)`

Consider induction on `a` by `(cdr a)`

The recursive definitions suggest plausible induction schemes

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

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

Proof: induct on a by (cdr a).

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

Proof: induct on a by (cdr 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: induct on a by (cdr 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: induct on a by (cdr 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: induct on a by (cdr a).

Base Case: (endp a).

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

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

```
Proof: induct on a by (cdr a).
```

```
Base Case:  (endp a).
(equal (ap b c)
       (ap b c))
```

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

Proof: induct on a by (cdr a).

Base Case: (endp a).

T

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

Proof: induct on a by (cdr 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: induct on a by (cdr 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: induct on a by (cdr 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: induct on a by (cdr 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: induct on a by (cdr 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: induct on a by (cdr 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: induct on a by (cdr 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: induct on a by (cdr 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: induct on a by (cdr 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: induct on a by (cdr 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)
       (ap (cdr a) (ap b c)))
```

$\{Ind\ Hyp\}$

```
Proof: induct on a by (cdr 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: induct on a by (cdr a).

Induction Step: (not (endp a)).

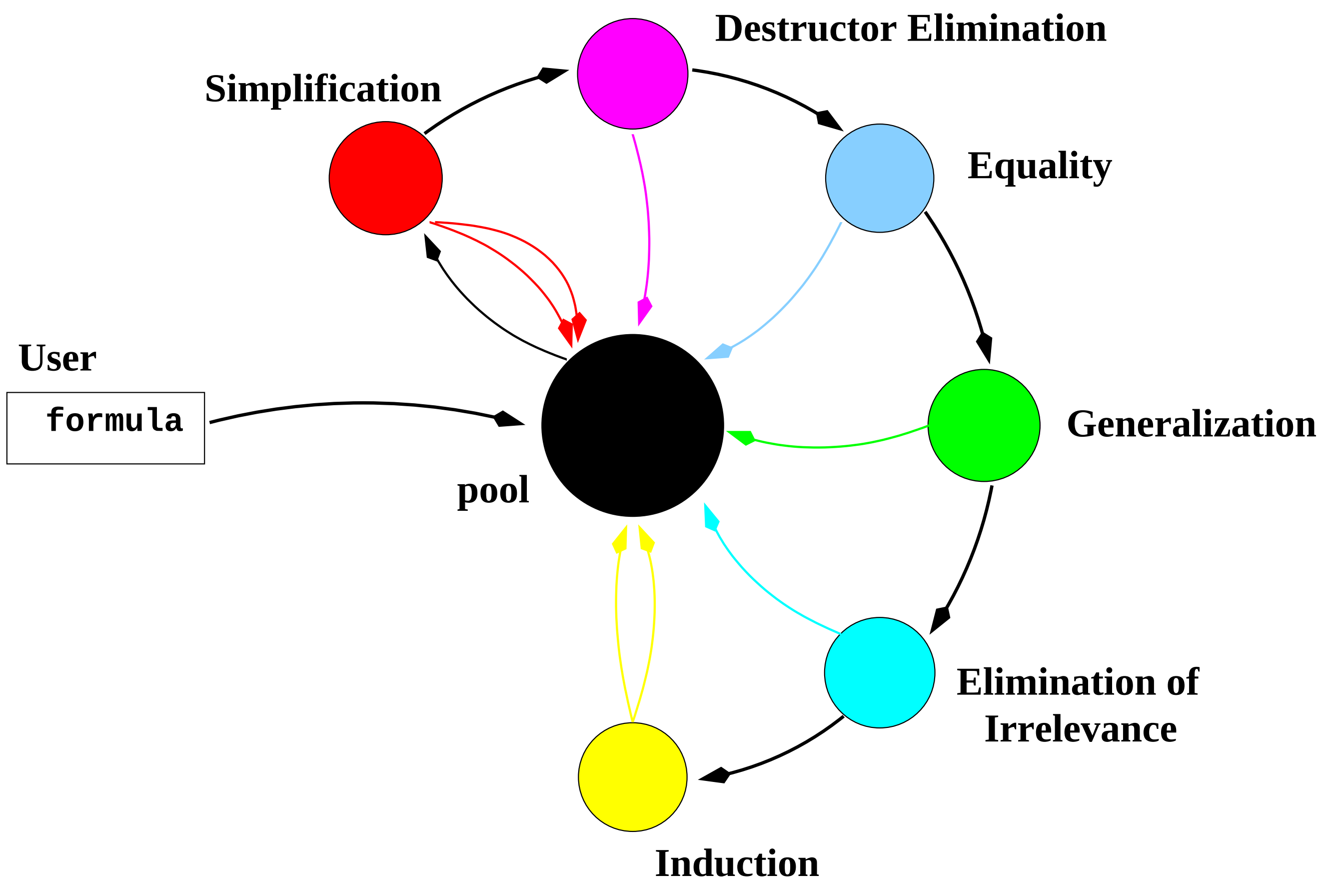
T

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

Proof: induct on a by (cdr a).

Q.E.D.

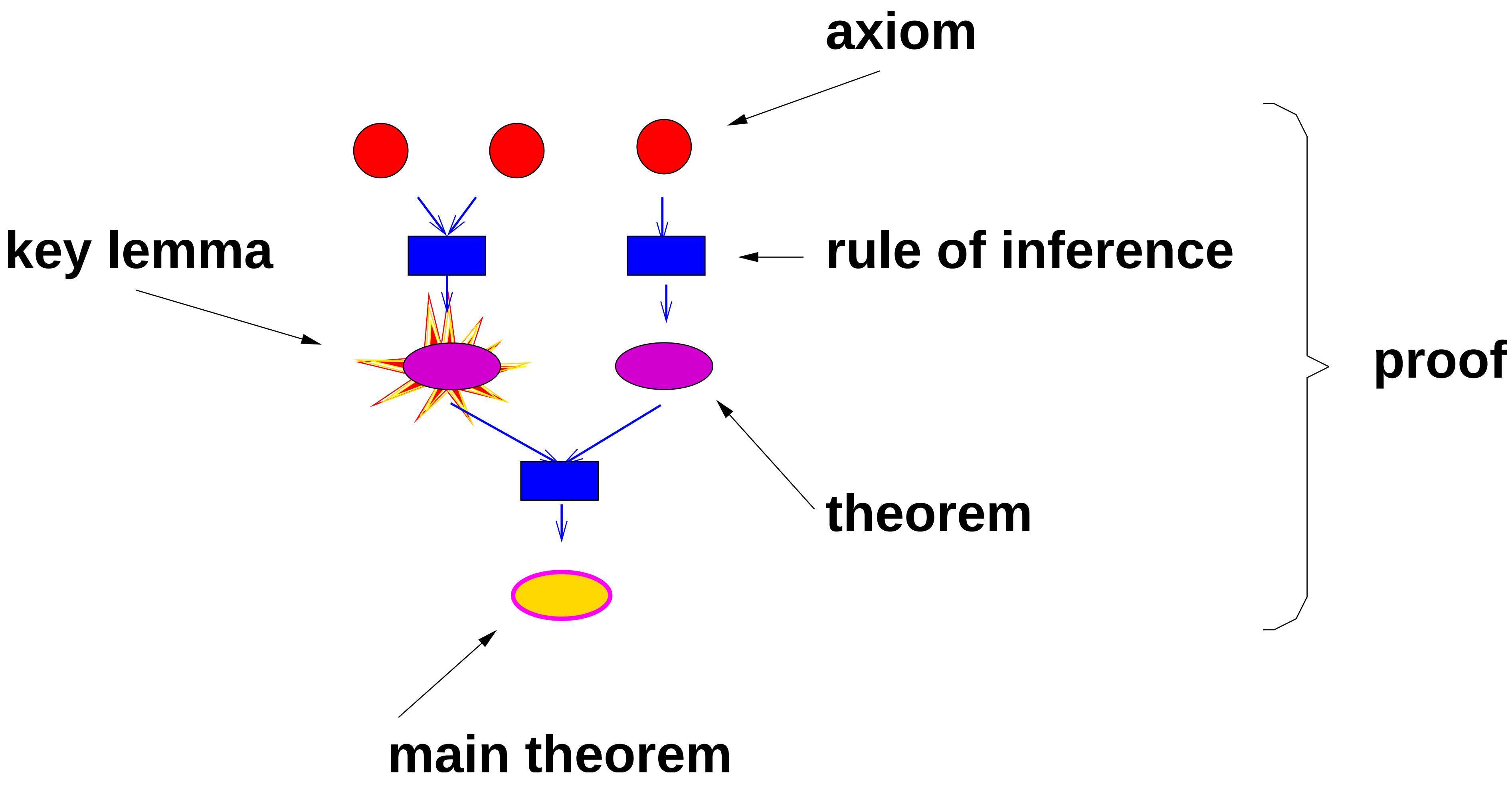
Heterogenous Proof Techniques, 1972

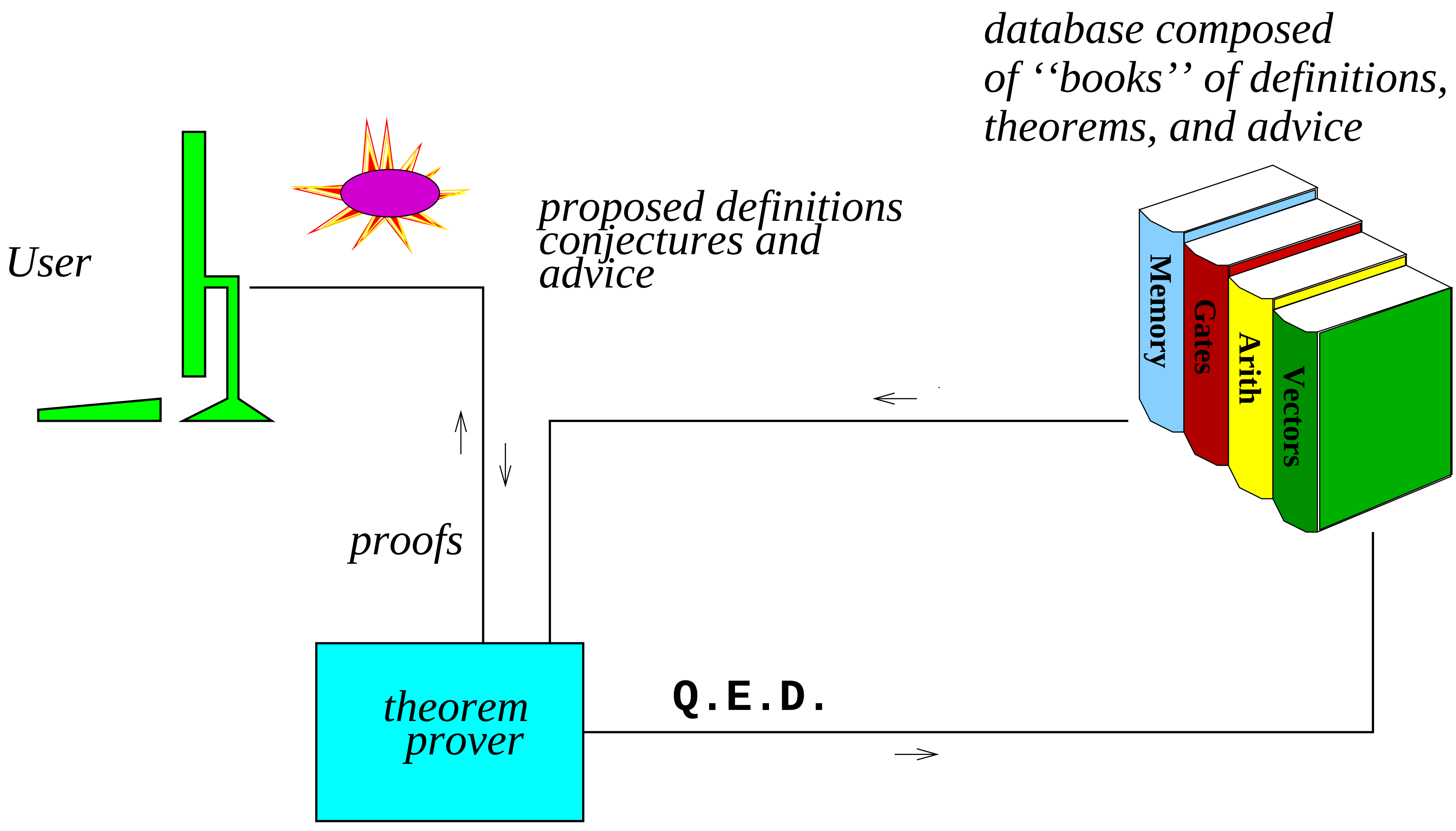


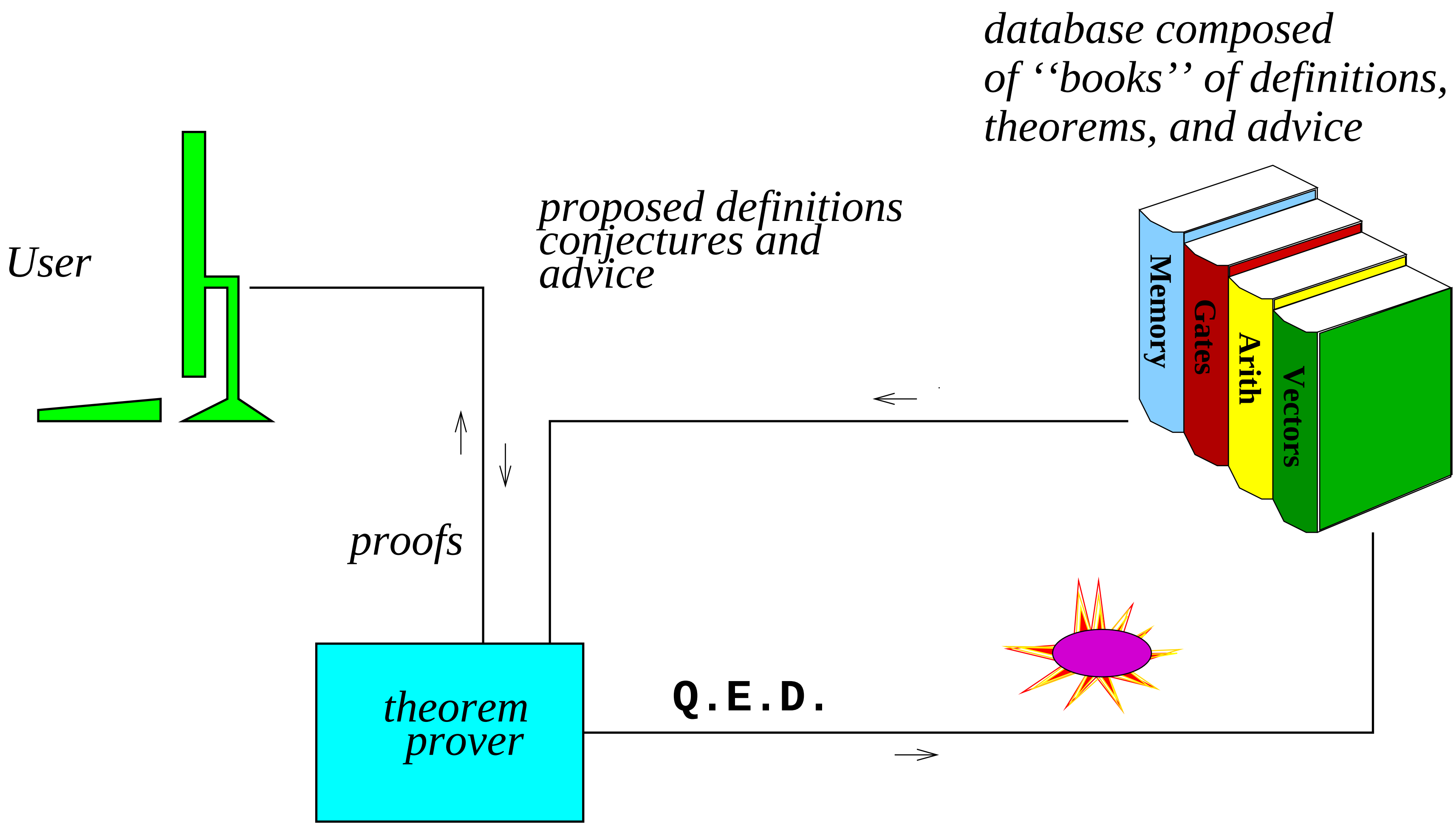
Lemmas, 1975

Allow the user to guide the proof by suggesting lemmas to prove first (interactive theorem proving above the proof-checker level)

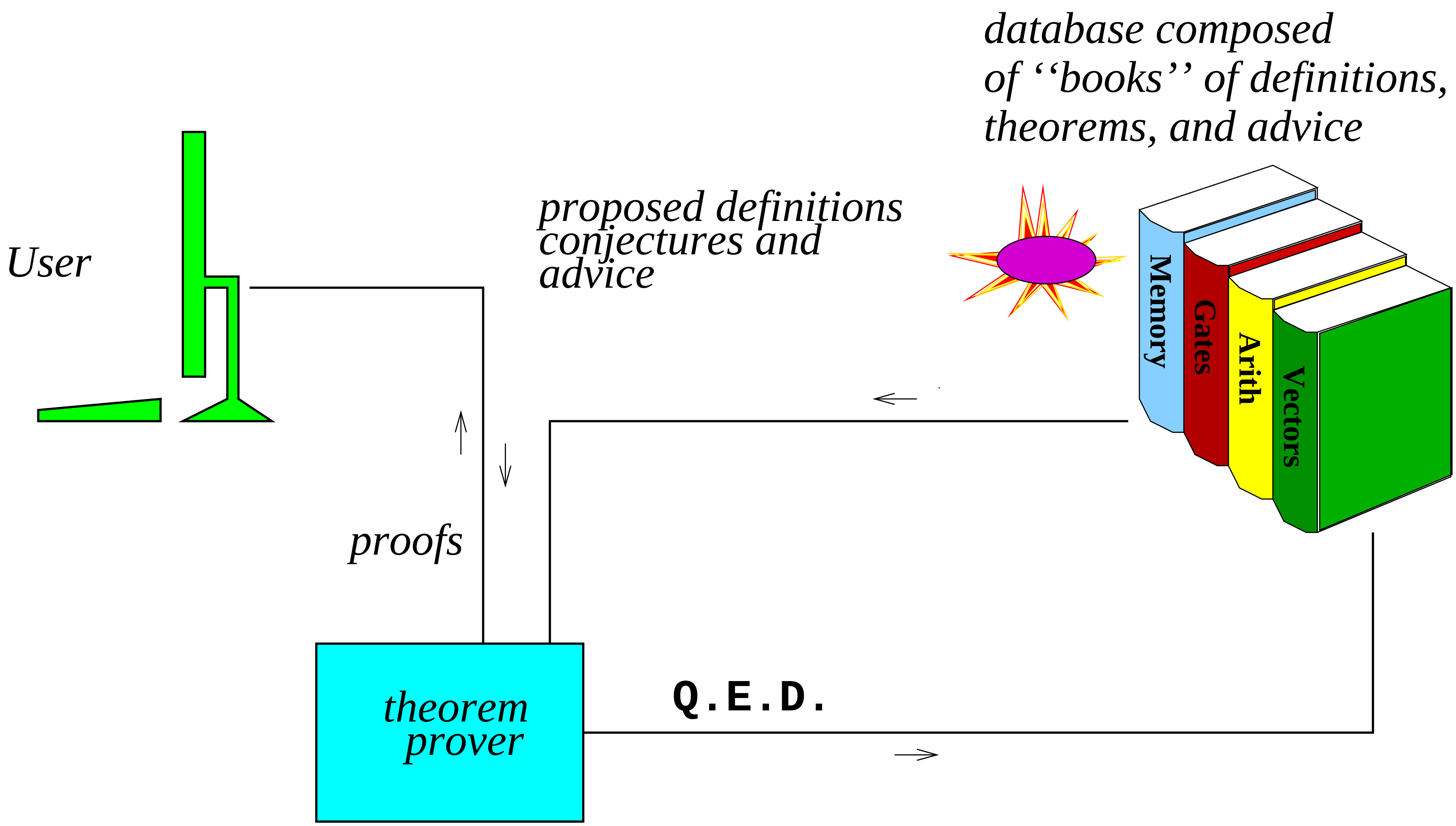
Mathematical facts are transformed into rules affecting the operation of the system and used automatically

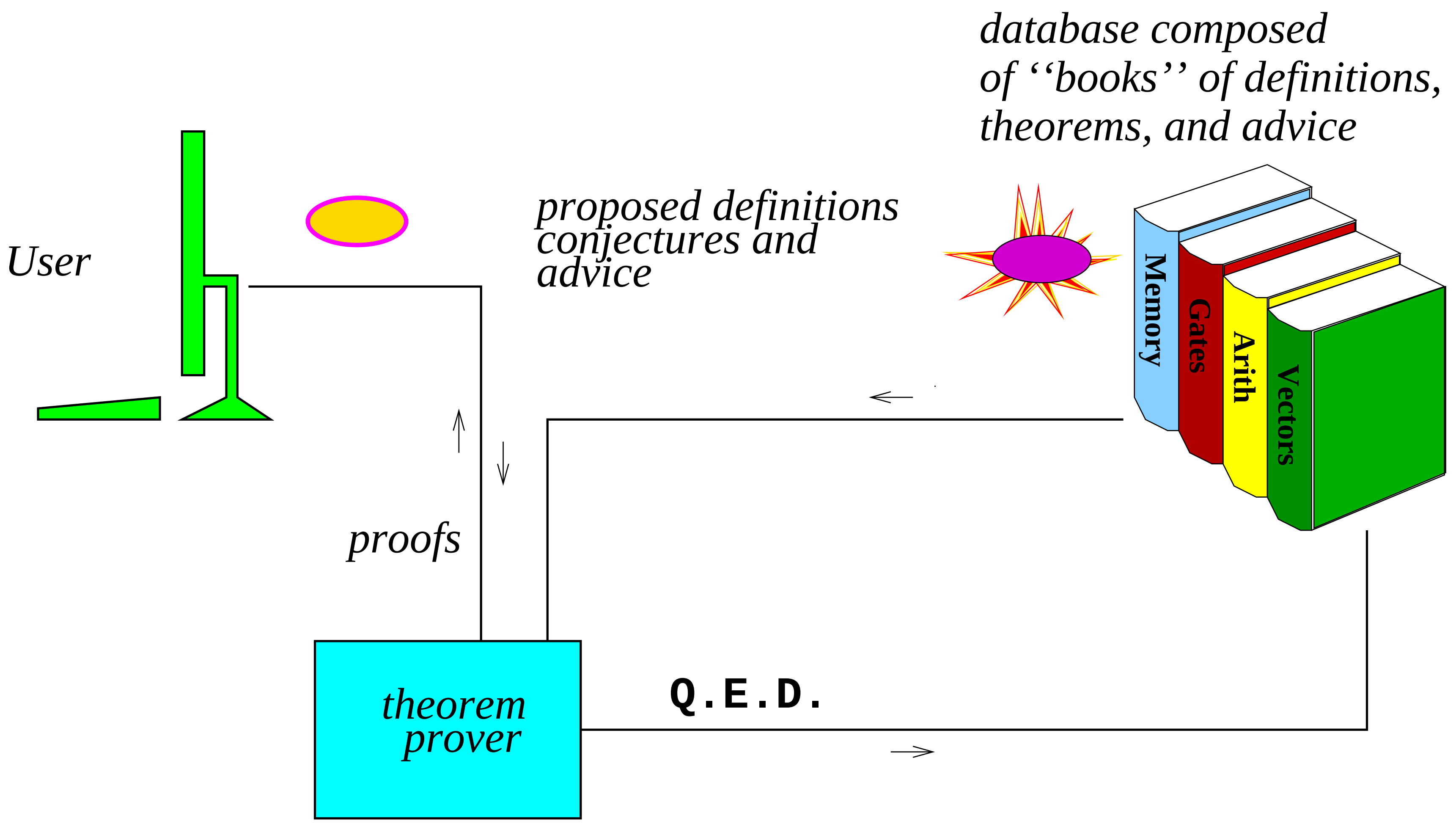

axiom
key lemma
rule of inference
proof
theorem
main theorem


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
proposed definitions
conjectures and
advice
User
Memory
Gates
Arith
Vectors
proofs
theorem
prover
Q.E.D.

Efficient Representation of Constants and Calculation, 1978

```
(CINT (PUSHF 15)           ; SIFT Dispatcher
      (PUSHM 1 13)         ; BDX 930 Assembler
      (PUSHM 0 0)
      (LOAD 0 ACLK)
SCHG  (TRA 1 15)
      (LDM 15 15 STACK)
      (PUSHM 0 1)
      (JSS* ASCHE)
      (TRA 15 12)
      (POPM 0 0)
      (POPM 1 13)
      (POPF 15)
      (CONT ES)
      (RET 0))
```

Operational Semantics, 1978

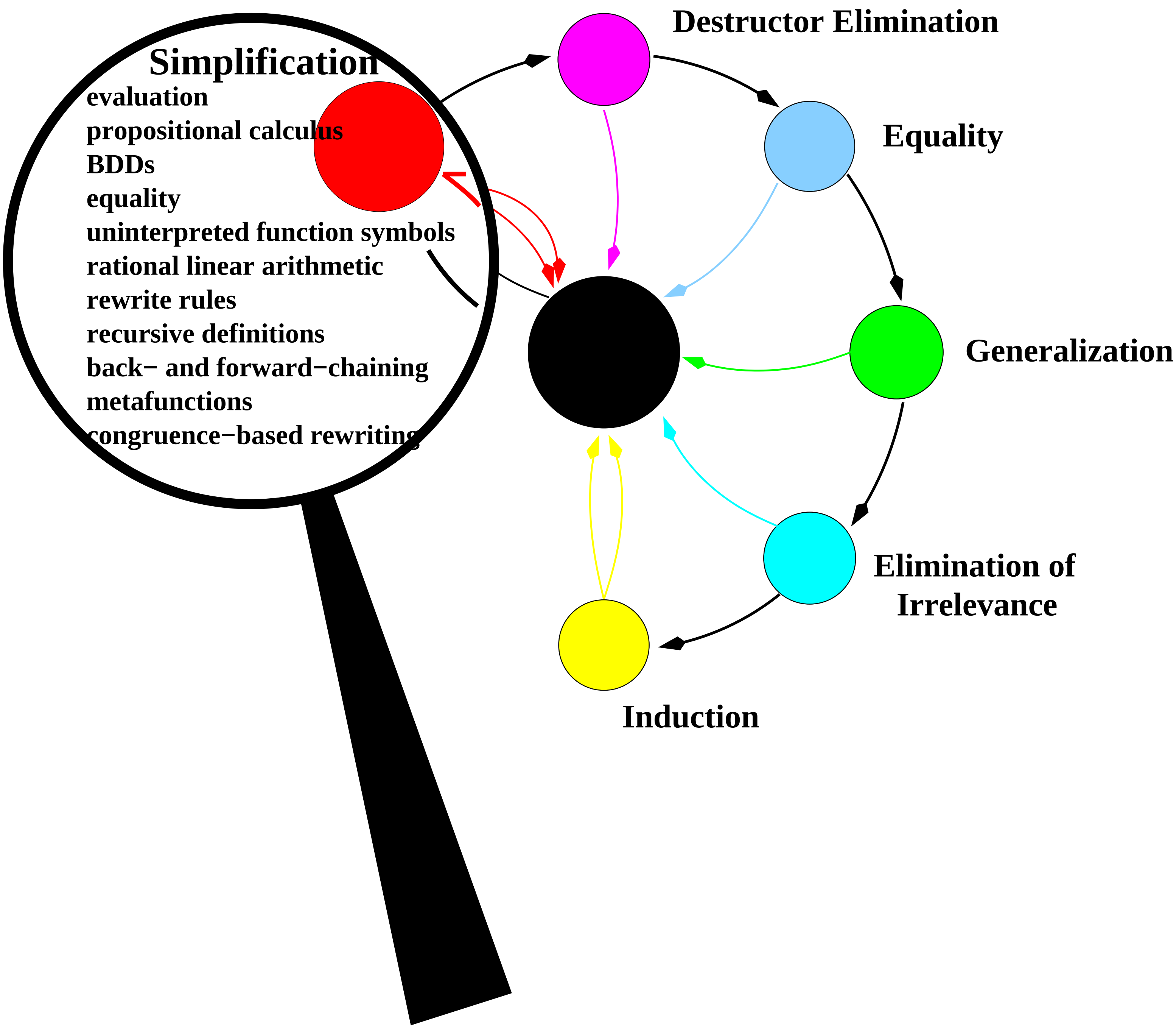
To capture the semantics of the instruction set, we encoded in our logic a recursive function that describes the state changes induced by each instruction. Thirty pages are required ... (in terms of certain still undefined bit-level functions such as the 8-bit signed addition function). We encountered difficulty getting the mechanical theorem prover to process such a large definition. However, the system was improved and the function was eventually admitted. We still anticipate great difficulty proving anything about the function because of its large size.

– *On why it is impossible to prove that the BDX90 dispatcher implements a time-sharing system*, Boyer and Moore, 1983

Integrated Decision Procedures, 1978

Decision procedures should be integrated into the rewriter

- `IF`-based normalization as a decision procedure for propositional calculus, 1972
- typing, 1973–...
- equality, 1978
- linear arithmetic, 1978–...

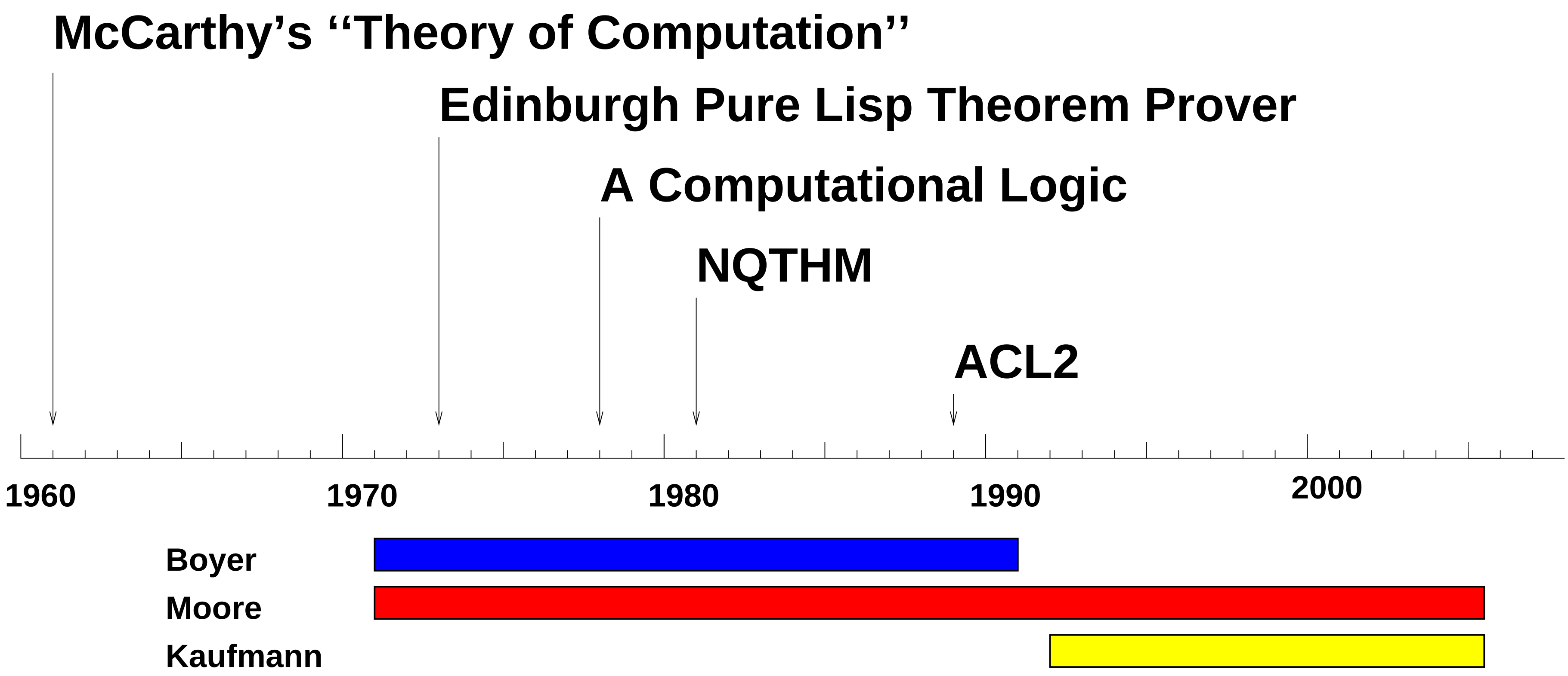

Simplification
evaluation
propositional calculus
BDDs
equality
uninterpreted function symbols
rational linear arithmetic
rewrite rules
recursive definitions
back− and forward−chaining
metafunctions
congruence−based rewriting
Destructor Elimination
Equality
Generalization
Elimination of Irrelevance
Induction

Meta-Theoretic Extensibility, 1979

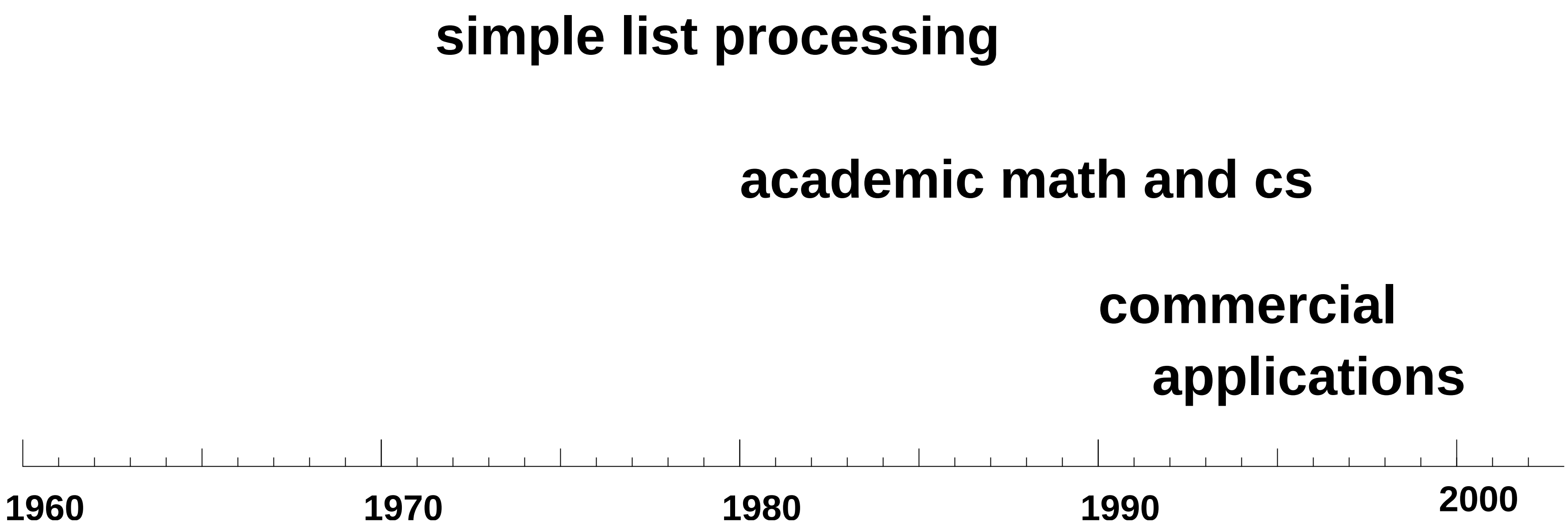
Theorem provers are written in Lisp

The logic is Lisp

Allow the user to code, verify, and use new techniques


McCarthy's ''Theory of Computation''
Edinburgh Pure Lisp Theorem Prover
A Computational Logic
NQTHM
ACL2
1960
1970
1980
1990
2000
Boyer
Moore
Kaufmann

Theorems Proved



1980s Academic Math

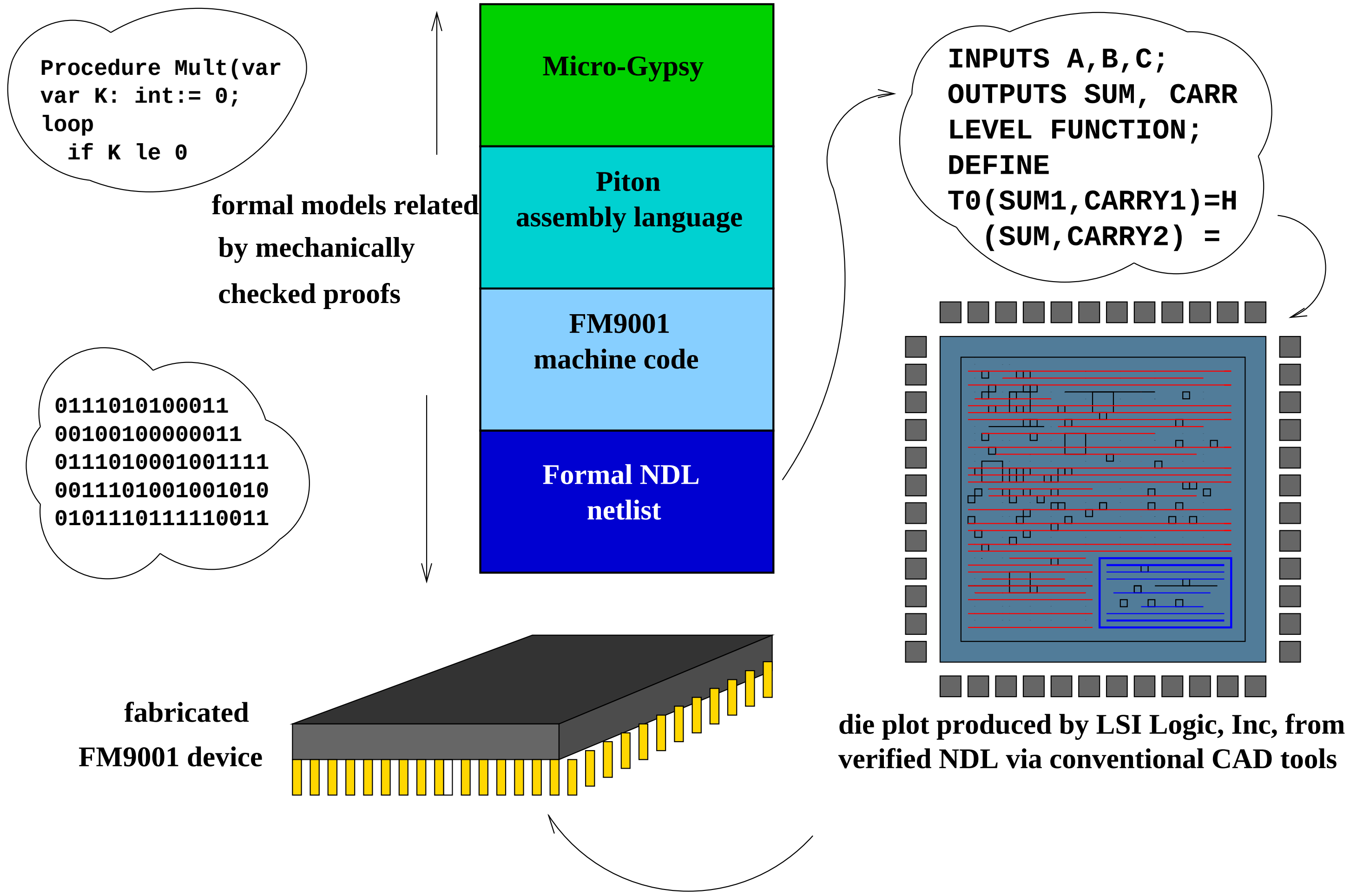
- undecidability of the halting problem (18 lemmas)
- invertibility of RSA encryption (172 lemmas)

- 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
Micro-Gypsy
Piton
assembly language
formal models related
by mechanically
checked proofs
FM9001
machine code
0111010100011
00100100000011
0111010001001111
0011101001001010
0101110111110011
Formal NDL
netlist
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

1990s

- FDIV on AMD K5
  [Moore-Kaufmann-Lynch]

- AMD Athlon floating point
  [Russinoff-Flatau]

- AMD process: all FPUs are to be
  mechanically verified

1990s

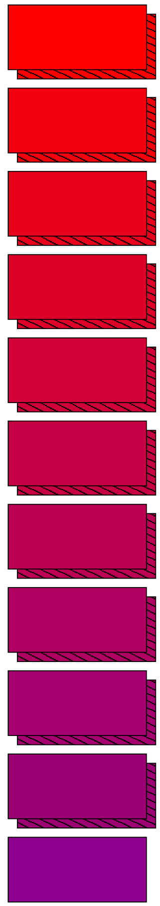
- Motorola 68020 and Berkeley C String Library [Yu]
- Motorola CAP DSP [Brock-Hunt]
- Rockwell Collins microarchitectural equivalence [Hardin-Greve-Wilding]

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]

- Sun Microsystems JVM [Liu]
- Centaur Technology (VIA) Media Unit [Hunt, Swords]
- Milawa: a Verified Stack of Theorem Provers [Davis]

Milawa Stack



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

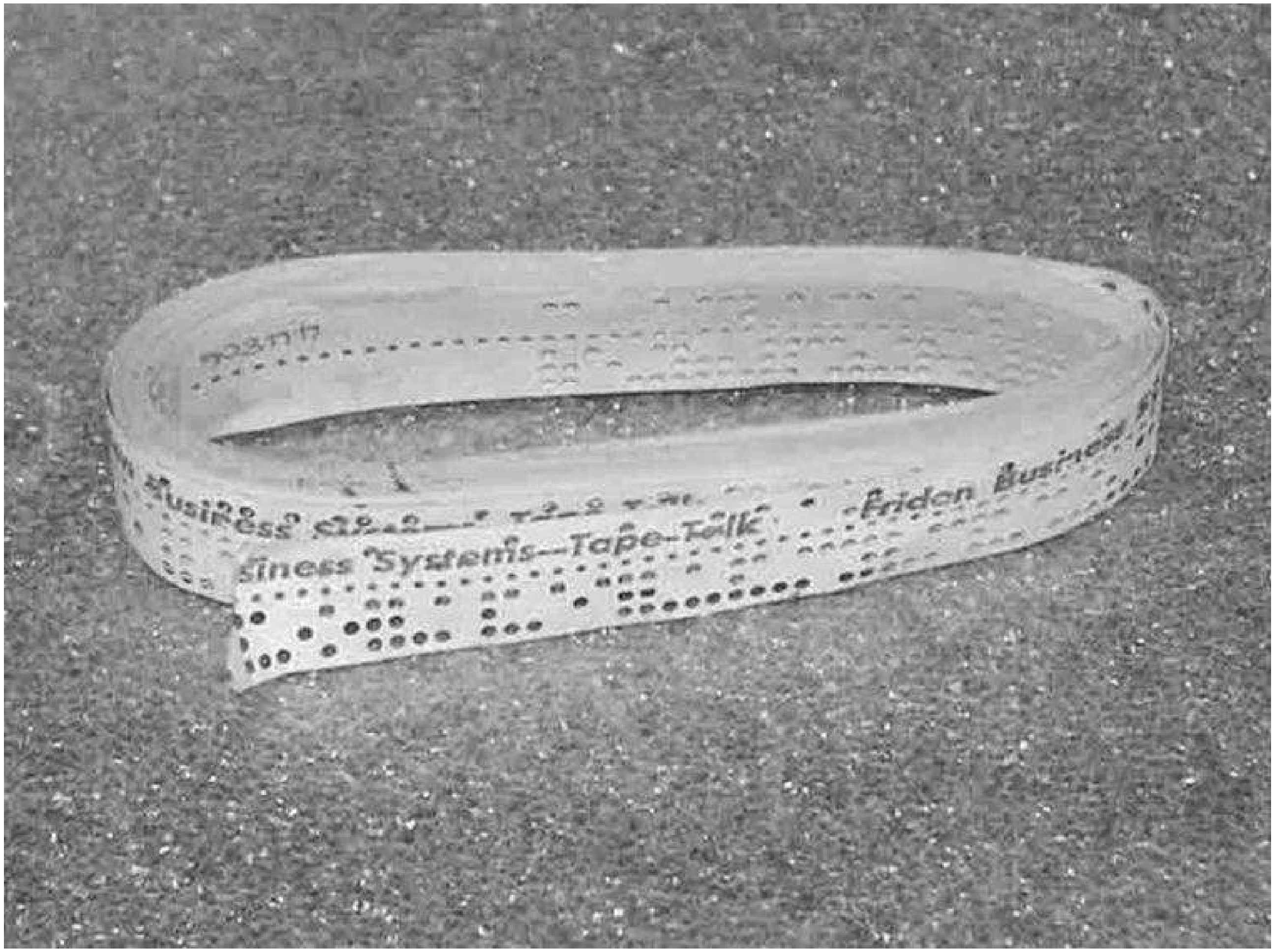
2 Propositional reasoning

1 Primitive proof checker

Proof Sizes (Gigabytes*)

Level	*Defs*	*Thms*	*Max Sz*	*Sum Sz*
1	201	2,015	2.8	51.4
2	87	514	2.7	72.3
3	230	815	4.9	63.9
4	168	991	9.2	152.9
5	192	1,071	3.7	74.6
6	55	402	6.0	26.2
7	83	749	3.5	7.5
8	184	1,059	5.6	54.4
9	427	2,475	1.5	12.3
10	82	616	1,934.3	2,713.9
11	233	1,157	0.2	21.4

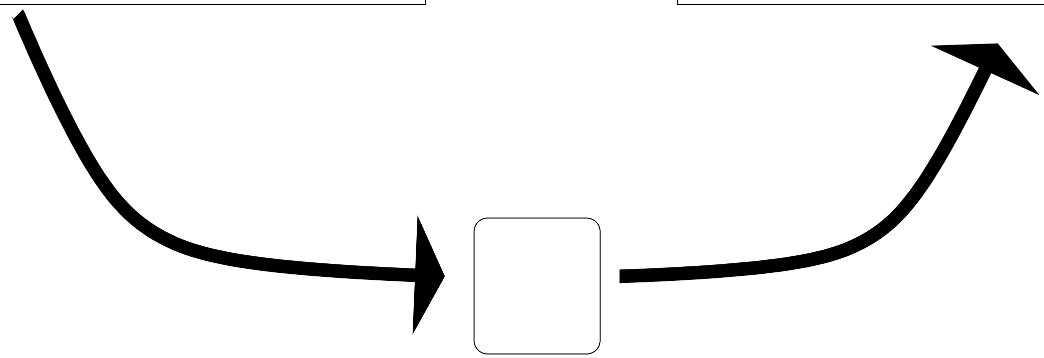
* 1 `cons` = 8 bytes


Friden Business
siness Systems--Tape-Talk

Editing

Input File ***Output File***

```
FUNCTION RESOLVE CL1 I CL2
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
END;

FUNCTION ...
```



<Command> ***<one character buffer>***

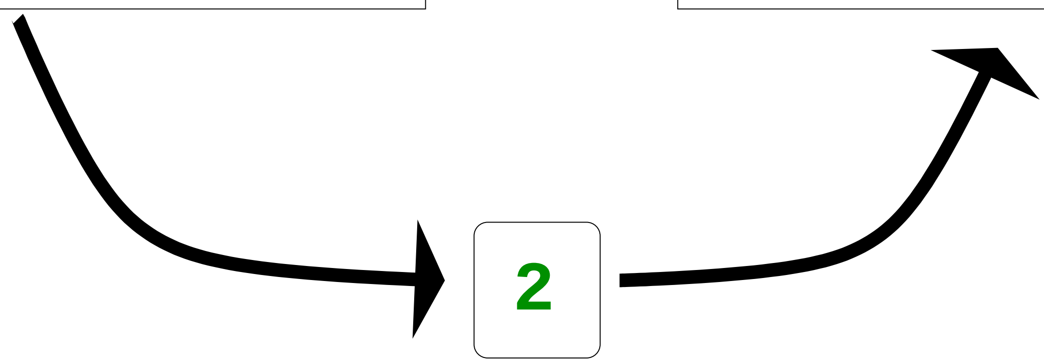
Editing

Input File

```
FUNCTION RESOLVE CL1 I CL2
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
END;

FUNCTION ...
```

Output File

```
FUNCTION RESOLVE CL1 I CL
```



Search: 2

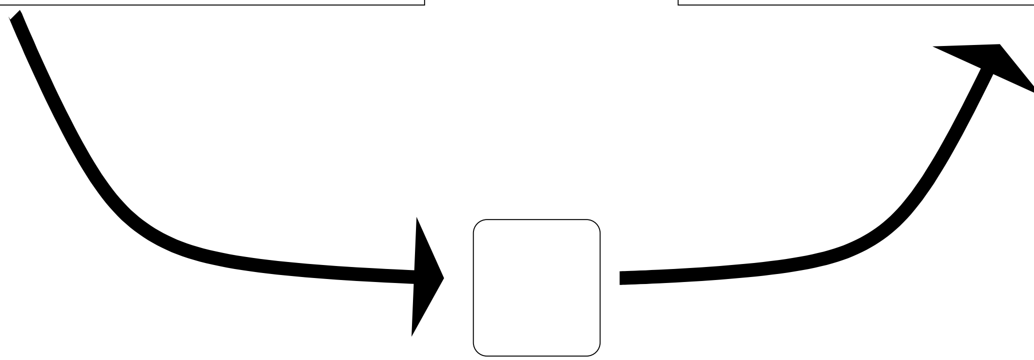
Editing

Input File

```
FUNCTION RESOLVE CL1 I CL2
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
END;

FUNCTION ...
```

Output File

```
FUNCTION RESOLVE CL1 I CL
```



Insert 2 J;

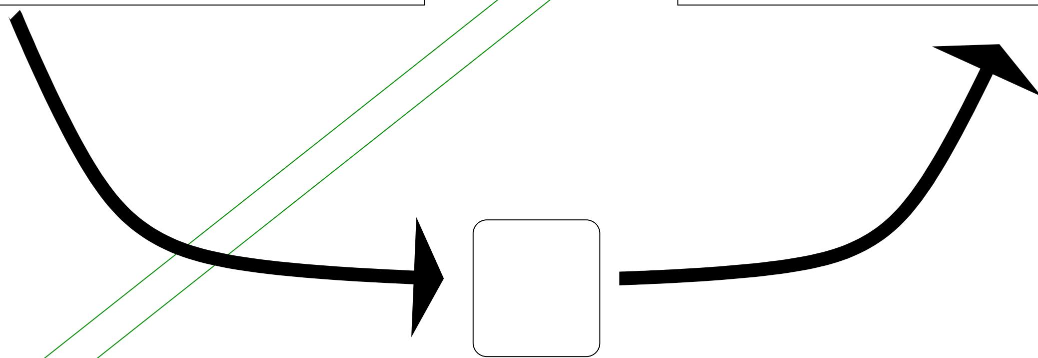
Editing

Input File

```
FUNCTION RESOLVE CL1 I CL2
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
END;

FUNCTION ...
```

Output File

```
FUNCTION RESOLVE CL1 I CL2 J;
```



Insert 2 J;

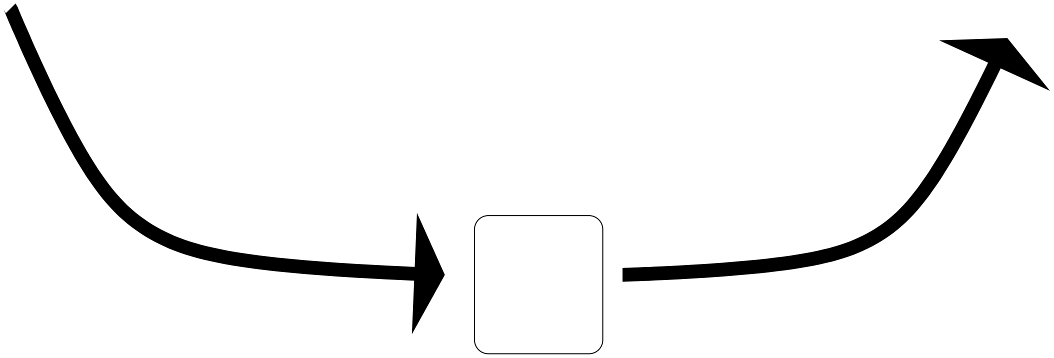
Editing

Input File

```
FUNCTION RESOLVE CL1 I CL2
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
END;

FUNCTION ...
```

Output File

```
FUNCTION RESOLVE CL1 I CL2 J;
```



Search FUNCTION

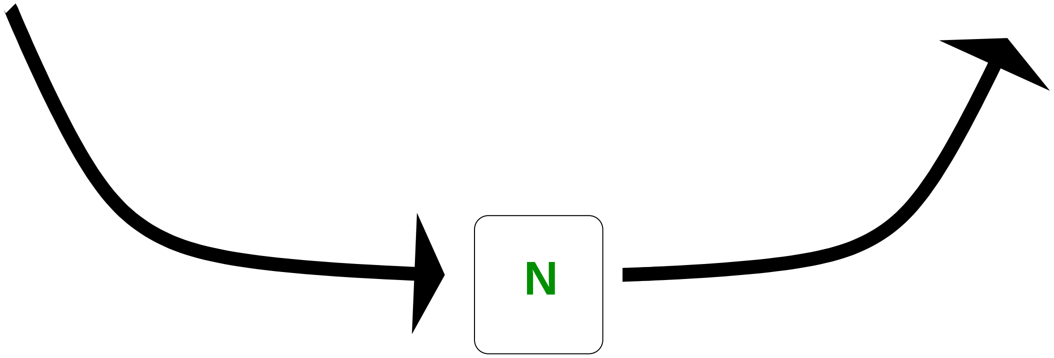
Editing

Input File

```
FUNCTION RESOLVE CL1 I CL2
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
END;

FUNCTION ...
```

Output File

```
FUNCTION RESOLVE CL1 I CL2 J;
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
E
```



Search FUNCTION

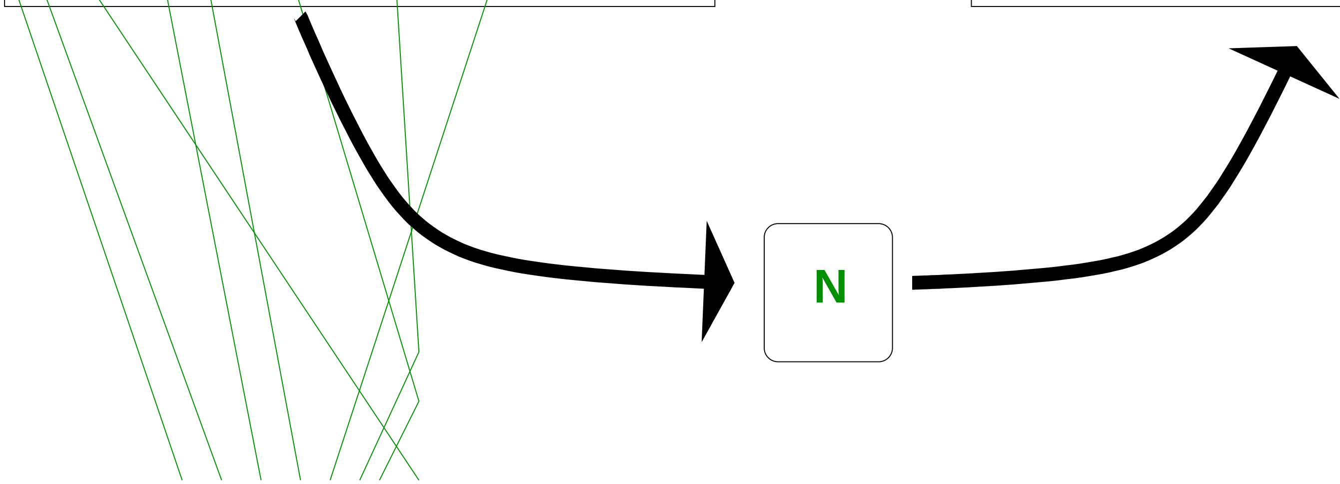
Editing

Input File

```
FUNCTION RESOLVE CL1 I CL2
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
END;

FUNCTION ...
```

Output File

```
FUNCTION RESOLVE CL1 I CL2 J;
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
E
```



Search FUNCTION

A Better Search Facility

Clearly, we needed a better string searching algorithm, but that is another story...

Of interest now is a better text editor!

How can we represent the document with a small memory footprint?

A Better Search Facility

Clearly, we needed a better string searching algorithm, but that is another story...

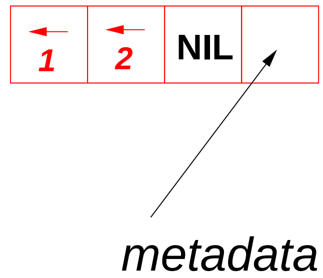
Of interest now is a better text editor!

How can we represent the document with a small memory footprint?

structure sharing!

Piece Table

```
FUNCTION RESOLVE CL1 I CL2
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
END;

FUNCTION ...
```



1 3

```
FUNCTION RESOLVE CL1 I CL2
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
END;

FUNCTION ...
```

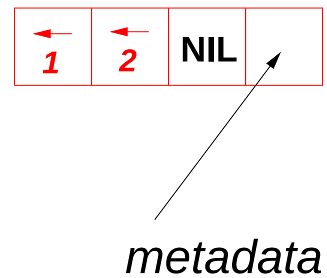
2

4

```
" J; @This function produces
the resolvent of CL1 (lit I)
with CL2 (lit J)@"
```

5

Piece Table



Piece Table

1 3

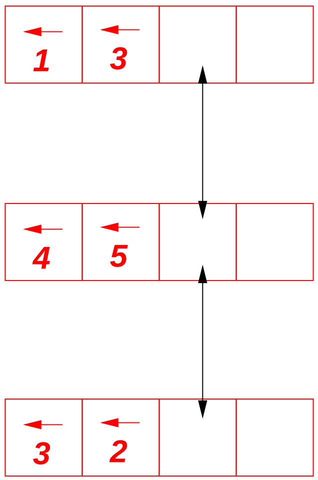
```
FUNCTION RESOLVE CL1 I CL2
VARS LEFTTERM LEFTI RIGHTTERM RIGHTI;
GETLIT(I,CL1) -> LEFTTERM -> LEFTI;
GETLIT(J,CL2) -> RIGHTTERM -> RIGHTI;
CONSCLAUSE(CL1,I,CL2,J,
           LITCNT(CL1)+LITCNT(CL2)-2,
           MAXINDEX(CL1)+MAXINDEX(CL2),
           NIL) -> BNDEV;
IF HD(LEFTTERM) /= HD(RIGHTTERM) AND
   UNIFY(HD(TL(LEFTERM)),LEFTI,
         HD(TL(RIGHTTERM),RIGHTI+MAXINDEX(CL1))
   THEN BNDEV; TRUE;
   ELSE FALSE; CLOSE;
END;

FUNCTION ...
```

2

4

```
" J; @This function produces
the resolvent of CL1 (lit I)
with CL2 (lit J)@"
```

5



The Piece Table

small memory footprint

easy undoing

provision for metadata

The Piece Table

When I moved to Xerox PARC, I explained the Piece Table to Charles Simonyi and Butler Lampson

Lampson had independently discovered it

They subsequently used it in the Bravo text editor

It migrated to Microsoft Word

It is still the representation used in Word

Lessons

- heuristics and some user guidance can put intractable problems within reach
- apply your methods to problems at the largest scale you can – and absorb the lessons

- understand the value of demonstrating what is *possible* – but don't think your work ends there (it has taken decades to get into the tool flow of microprocessor design)

- believe in your dreams – and act on them

Acknowledgements

This personal retrospective has ingored the many other theorem prover communities where great work is also being done

The "Boyer-Moore community" has grown too numerous to list all the key players, but I'd like to especially thank Bob Boyer, Matt Kaufmann, and Warren Hunt.

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