What Can Formal Methods Offer to Digital Flight Control Systems Design?

Formal Methods Workshop
NASA Langley Research Center
Hampton, VA.

August 20-23, 1990

Donald I. Good
Computational Logic, Inc.

Abstract

Formal methods research is beginning to produce methods which will enable mathematical modeling of the physical behavior of digital hardware and software systems. The development of these methods directly supports the NASA mission of increasing the scope and effectiveness of flight system modeling capabilities.

The conventional, continuous mathematics that is used extensively in modeling flight systems is not adequate for accurate modeling of digital systems. Therefore, the current practice of digital flight control system design has not had the benefits of extensive mathematical modeling which are common in other parts of flight system engineering.

Formal methods research is showing that by using discrete mathematics, very accurate modeling of digital systems is possible. These discrete modeling methods are still in an embryonic stage. But when they are fully developed, they will bring the traditional benefits of modeling to digital hardware and software design. Sound reasoning about accurate mathematical models of flight control systems can be an important part of reducing the risks of unsafe flight control.

What Can Formal Methods Offer to Digital Flight Control Systems Design?

Donald I. Good

Computational Logic, Inc.
1717 West Sixth, Suite 290
Austin, Texas 78703

512-322-9951

good@cli.com

"Formal Methods" Enable

Mathematical Modeling

of

Digital Systems

(Hardware and Software)

NASA Mission Objective: Increase the scope and effectiveness of flight system modeling capabilities. -- Lee Holcomb, NASA HQ, 1990.

Why Model?

For either design of a new system or operation of an old one, modeling provides...

Benefits: early error detection

- Saves time
- Saves money
- Saves operational disruption
- Saves operational mishaps

Risks: model misrepresents system

- Inaccurate
- Incomplete

Kinds of models: physical, analog, schematic, mathematical.

Blanchard and Fabrycky. Systems Engineering and Analysis, Prentice Hall, 1990.

Why a Mathematical Model?

- High abstraction
- High precision
- Simulate by manipulating symbols
- Represent large classes of system states
- Use mathematical deduction

Get a lot of system simulation for a little symbol manipulation.

Operational Safety

Operating a system safely requires

- accurate predictions

of how it will behave.

Accurate predictions can be obtained from

- sound deductions about
- accurate mathematical models

of system behavior.

A Classic Model

Free Fall Distance:

```
f(b,t) = [g(b) * t**2] / 2

g(b) = if b="earth" then 32
  else if b="moon"  then ...

t is time (sec)

f(b,t) is distance (ft)
```

Simulation:

```
f("earth", .7) = [32 * .7**2] / 2

               =  16 * .49

               =  7.84 ft
```

Power of Mathematical Deduction

```
Suppose 0 le t0 le t1.

        t in [t0..t1]

f("earth", t) in (32 * [t0..t1]**2) / 2

f("earth", t) in  16 * [t0..t1]**2

f("earth", t) in  16 * [t0**2..t1**2]

                        (** is monotonic)
```

Physical simulation of this result is impossible because `[t0..t1]` contains an infinite number of values.

Validating a Model

- Ultimately, the accuracy of a model of a physical system must be validated by testing it against measured, observed behavior of the actual physical system.
- One cannot construct a mathematical proof that a model is an accurate representation of a physical system.
- Typically, one iterates through a process of
  - stating a mathematical model
  - testing it against physical observations
  - adjusting the model

Hardware Model Observables

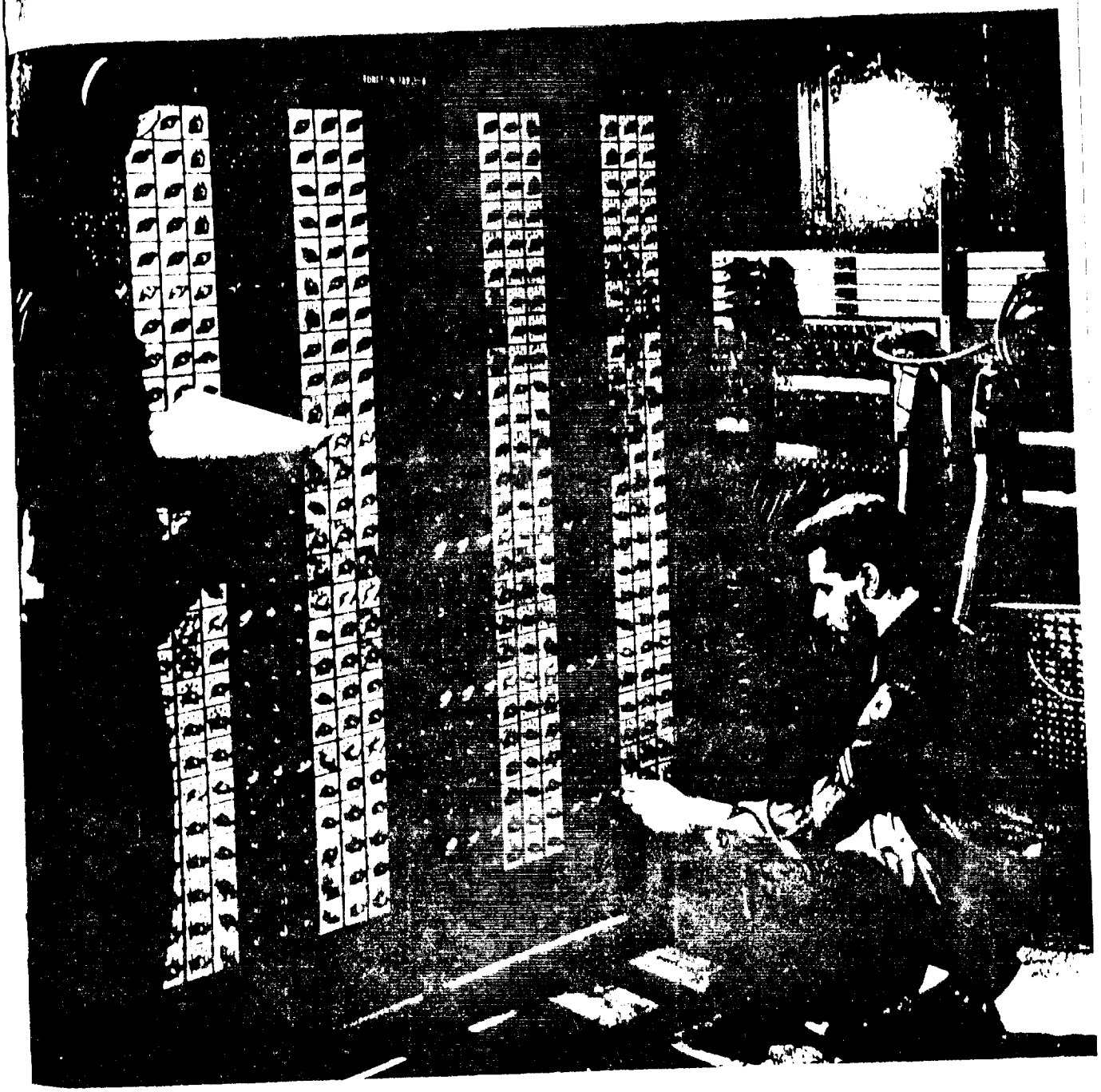
A hardware system

is composed

of physical switches.

Nancy Stern. From ENIAC to UNIVAC: An Appraisal of the Eckert-Mauchly Computers. Digital Equipment Corporation, 1981.

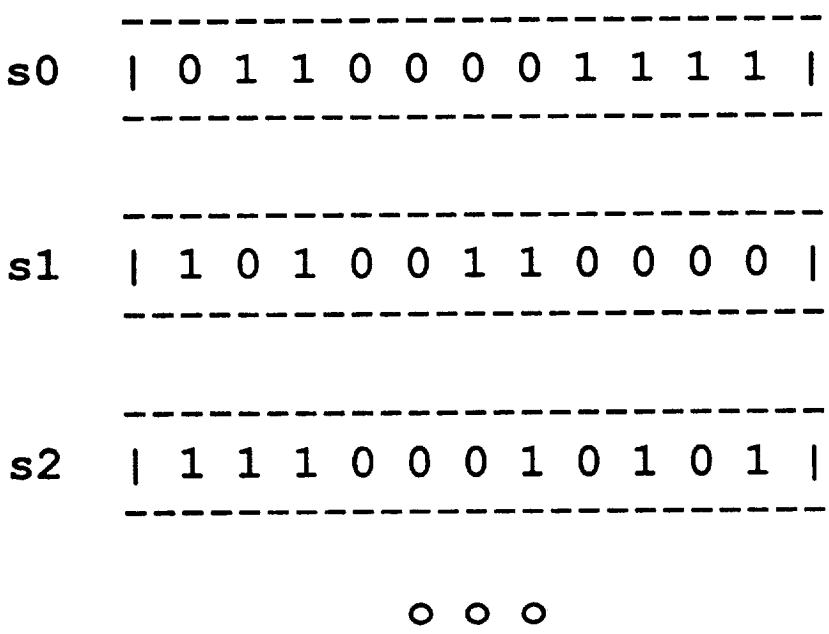
Next page.



The switches of Function Table A are being set by Pfc. Homer Spence and Cpl. Irwin Goldstein. Three manually set function tables served as read-only memory units. Courtesy Moore School of Electrical Engineering, University of Pennsylvania.

Use Discrete Mathematics to Model Hardware

- Switches by binary digits
- Operation by recursive functions



An MC68020 Machine Model

```
MC68020(s,n) =
  if   haltp(s) or n=0
  then s
  else MC68020(NEXT(s), n-1)

NEXT(s) =
  if   evenp(pc(s))
  then if   pc_readp(mem(s),pc(s))
       then EXECUTE(FETCH(pc(s),s),
                    update_pc(s,...))
       else halt(s, pc_signal)
  else halt(s,pc_odd_signal)

EXECUTE(ins,s) =
... [50 pages for 90% user ins.] ...
```

Provides a mathematically precise and consistent machine language reference manual.

Yuan Yu. PhD Thesis (in progress). University of Texas.

The VIPER Machine

A 32-bit microprocessor "whose functions are totally predictable."

- Accumulator
- 2 index registers
- Program counter
- Comparison register
- 16 instructions

Avra Cohn. A Proof of Correctness of the VIPER Microprocessor: The First Level. Technical Report 104, University of Cambridge Computer Laboratory, January, 1987.

W. J. Cullyer. Implementing High Integrity Systems: The VIPER Microprocessor. In Computer Assurance, COMPASS 88. IEEE, June, 1988.

A VIPER Machine Model

```
NEXT(ram,p,a,x,y,b,stop) =
  if   stop
  then (ram,p,a,x,y,b,stop)
  else (noinc \/ illegaladdr) \/
       if       (illegalcl  \/ illegalsp)
            \/ (illegalonp \/ illegalwr)
       then (ram,newp,a,x,y,b,T)
       else ... [about 7 pages] ...

where

ram  - a memory of 32-bit words
p    - 20-bit program counter
a    - 32-bit accumulator
x,y  - 32-bit index registers
b    - 1 bit compare result register
stop - stop flag
```

The FM8502 Machine

A 32-bit microprocessor.

- 2 address architecture
- 4 addressing modes
- 8 general purpose registers
- 2^{19} 20-bit instructions

Warren A. Hunt, Jr. FM8501: A Verified Microprocessor, Ph.D. Thesis, The University of Texas at Austin, 1985.

-----, Microprocessor Design Verification. Journal of Automated Reasoning. Vol. 5, No. 4, Dec 1989.

An FM8502 Machine Model

```
FM8502(ms,mn) =
  if   not(listp(mn))
  then ms
  else FM8502(NEXT(ms),
                rest(mn))

NEXT(ms) =
  list(next_memory          (ms),
       next_register_file(ms),
       next_carry_flag    (ms),
       next_overflow_flag(ms),
       next_zero_flag     (ms),
       next_negative_flag(ms) )


... [about 10 pages] ...
```

An FM8502
Register Transfer Model

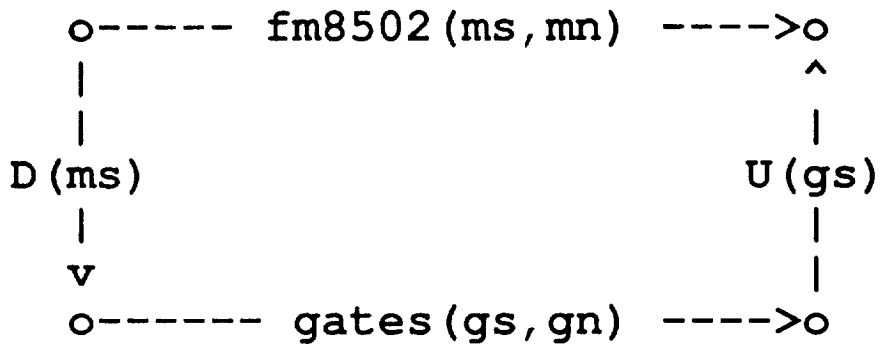
```
GATES(gs,gn) =
  if   not(listp(gn))
  then gs
  else GATES(COMB_LOGIC(gs,car(gn)),
             cdr(gn))

COMB_LOGIC(gs,gn) =
... [on bit operators, e.g., b_xor] ...
```

where

gs	- [regs, flags, mem, int-regs]
regs	- 8 32-bit vectors
flags	- 4 Booleans
mem	- 2^{32} 32-bit vectors
int-regs	- 32-bit vectors for internal registers, flags, latches

Connecting the Models




```
Theorem: H(ms,mn) ->
           fm8502(ms,mn) =
           U(gates(D(ms),Kg(ms,mn,md)))
```

Under the conditions `H`,

- **the `fm8502` model is just as <u>accurate</u> as `gates`**
- **but with some <u>details</u> <u>suppressed</u> by `U`.**

Software Model Observables

Programming languages provide

a wide variety of ways

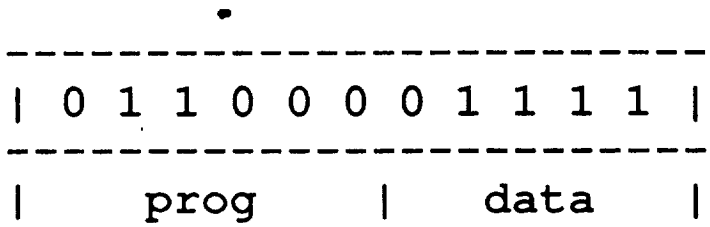
of describing them, but

the observables are still switches,

and so are programs!

Models of Programmed Machines

- A machine is programmed by setting the switches which it will interpret as instructions during its operation. (Before stored-program machines, this process was called "setting up" the machine.)



- These switches are the program. They control the subsequent operation of the machine.
- A computer program is a physical control mechanism.
- The bit string "011000" is a mathematical description of the control mechanism.

A Model of a Programmed Machine

A model of machine `M` operating on initial state `s0` for `k(s0)` steps under the control of the program described by `p0` is given by

```
M(s0,k(s0))
```

where

```
s0        - a machine state such that
            prog(s0)=p0

prog(s)   - a function that extracts the
            program description from s
```

Operating Requirements

A model of a machine programmed to satisfy an operating requirement `R(s0,sk)` is given by

```
R(s0, M(s0,k(s0)))
```

A Program Description, p0

```
088B 000D 0002 088B 000E 0003 004B 0003 00BF 000E 0CD0 004D 0002 0009 0041 0002
000F 10CB 0002 0000 31CB 0002 0000 12CB 0002 000D 13CB 0002 000E 0CCB 0002 0004
0DCB 0002 0005 0ECB 0002 0006 0FCB 0002 0007 0041 0002 0008 50CB 0002 0000 104B
0002 000D 104B 0003 000E 0000 084B 0003 0002 004D 0002 0009 0041 0002 000F 004D
0003 0002 0041 0003 009F 00DA 0003 01DE 0003 084B 0003 0002 0041 0003 0008 188B
0000 0002 398B 0000 0002 1A8B 0000 0003 F84B 0007 0002 D84B 0006 0002 B84B 0005
0002 984B 0004 0002 784B 0003 0002 584B 0002 0002 0000 000E 09F3 004B 0003 00BF
000E 0CD0 000E 0CCA 000E 0CA8 0002 0C86 000E 09F3 004B 0003 00BF 000E 0CD0 000E
0CCA 004D 0002 0002 0041 0002 00D3 00CB 0002 0001 01CB 0002 0000 0002 0C86 000E
09F3 0089 0008 0004 0083 0008 0000 000A 0A98 0083 0008 0001 000A 0AFD 0083 0008
0002 000A 0B6A 0002 0BA5 084B 0006 0002 0049 0006 0010 084B 0007 0003 004B 0003
00BF 000E 0CD0 088B 000C 0002 084B 0004 0006 004D 0004 0008 084B 0003 0002 004D
0003 0080 0841 0003 0004 0041 0003 01F3 000E 0CE1 000A 0AE7 084B 0002 0007 000E
0CA8 084B 0003 0006 004D 0003 0002 0041 0003 00D3 00C3 0003 0003 0006 0C96 11C3
0003 000C 0006 0C96 00CB 0003 0000 01CB 0003 0000 084B 0002 0006 004B 0003 00BF
000E 0CA8 0002 0C96 004B 0003 00BF 000E 0CCA 104B 0003 000C 004D 0003 0002 0041
0003 00D3 00CB 0003 0002 09CB 0003 0006 0002 0C86 084B 0006 0002 0049 0006 0010
004B 0003 00BF 000E 0CD0 088B 000C 0002 084B 0004 0002 004D 0004 0008 084B 0003
0006 004D 0003 0080 0841 0003 0004 0041 0003 01F3 000E 0CDD 000A 0B54 000E 0CD0
000E 0CCA 104B 0003 000C 004D 0003 0009 0041 0003 000F 0BCB 0003 0002 084B 0003
0006 004D 0003 0002 0041 0003 00D3 00C3 0003 0002 0006 0C96 11C3 0003 000C 0006
0C96 00CB 0003 0000 01CB 0003 0000 084B 0002 0006 004B 0003 00BF 000E 0CA8 0002
0C96 004B 0003 00BF 000E 0CCA 104B 0003 000C 004D 0003 0002 0041 0003 00D3 00CB
0003 0003 09CB 0003 0006 0002 0C86 084B 0007 0003 004B 0003 00BF 000E 0CD0 088B
000C 0002 084B 0003 0002 004D 0003 0008 0041 0003 0173 000E 0CE1 000A 0B8F 084B
0002 0007 000E 0CA8 00B6 000C 0006 0C96 00A6 000C 0002 0C96 004B 0003 00BF 000E
0CCA 104B 0003 000C 004D 0003 0002 0041 0003 00D3 00CB 0003 0004 01CB 0003 0000
0002 0C86 004B 0003 00BF 000E 0CD0 088B 000C 0002 084B 0003 0002 004D 0003 0008
0041 0003 00F3 000E 0CDD 000A 0BCC 000E 0CD0 000E 0CCA 104B 0003 000C 004D 0003
0009 0041 0003 000F 0BCB 0003 0002 0002 0C96 004B 0003 00BF 000E 0CCA 104B 0003
000C 004D 0003 0002 0041 0003 00D3 00CB 0003 0005 01CB 0003 0000 0002 0C86 008B
000A 0C86 088B 000E 0003 004B 0003 00BF 000E 0CDD 000A 0BF7 008B 000A 0C9F 104B
0003 000E 000E 09F3 104B 0005 0008 004D 0005 0002 0041 0005 00D3 00C3 0005 0005
0006 0C13 104B 0002 0008 004B 0003 00BF 000E 0CA8 00CB 0005 0000 01CB 0005 0000
104B 0003 0008 004D 0003 0008 0041 0003 00F3 000E 0CE1 0006 0C2A 104B 0002 0009
0041 0002 0100 000E 0CBA 0082 000A 00B2 0008 0006 0C38 104B 0002 0009 0041 0002
0100 000E 0CA8 0082 000A 104B 0002 0009 000E 0CA8 0082 000A 008B 000A 0C86 088B
000E 0003 004B 0003 00BF 000E 0CDD 000A 0C54 008B 000A 0C9F 104B 0003 000E 000E
09F3 104B 0005 0009 004D 0005 0002 0041 0005 00D3 00C3 0005 0004 0006 0C70 104B
0002 0009 004B 0003 00BF 000E 0CA8 00CB 0005 0000 01CB 0005 0000 104B 0003 0009
004D 0003 0008 0041 0003 0173 000E 0CDD 008A 000A 000E 0CD0 088F 0009 0002 000E
0CCA 0082 000A 004B 0003 00BF 000E 0CDD 000A 0C95 000E 0CD0 000E 0A29 00BA 000B
00A2 0000 0004 004B 0003 00BF 000E 0CD0 000E 0A29 00AE 0000 004B 0003 00BF 000E
0CD0 000E 0A29 00A2 0000 084B 0004 0003 0041 0004 0004 3841 0004 0003 08CB 0004
0002 02D6 0003 79C7 0003 0003 0000 384B 0004 0003 7845 0004 0003 0841 0004 0003
0041 0004 0004 08CB 0004 0002 0000 02D2 0003 78C7 0003 0003 0000 084B 0002 0003
0041 0002 0004 1841 0002 0003 184B 0002 0002 0000 02C3 0003 0000 0000 7AC3 0003
```

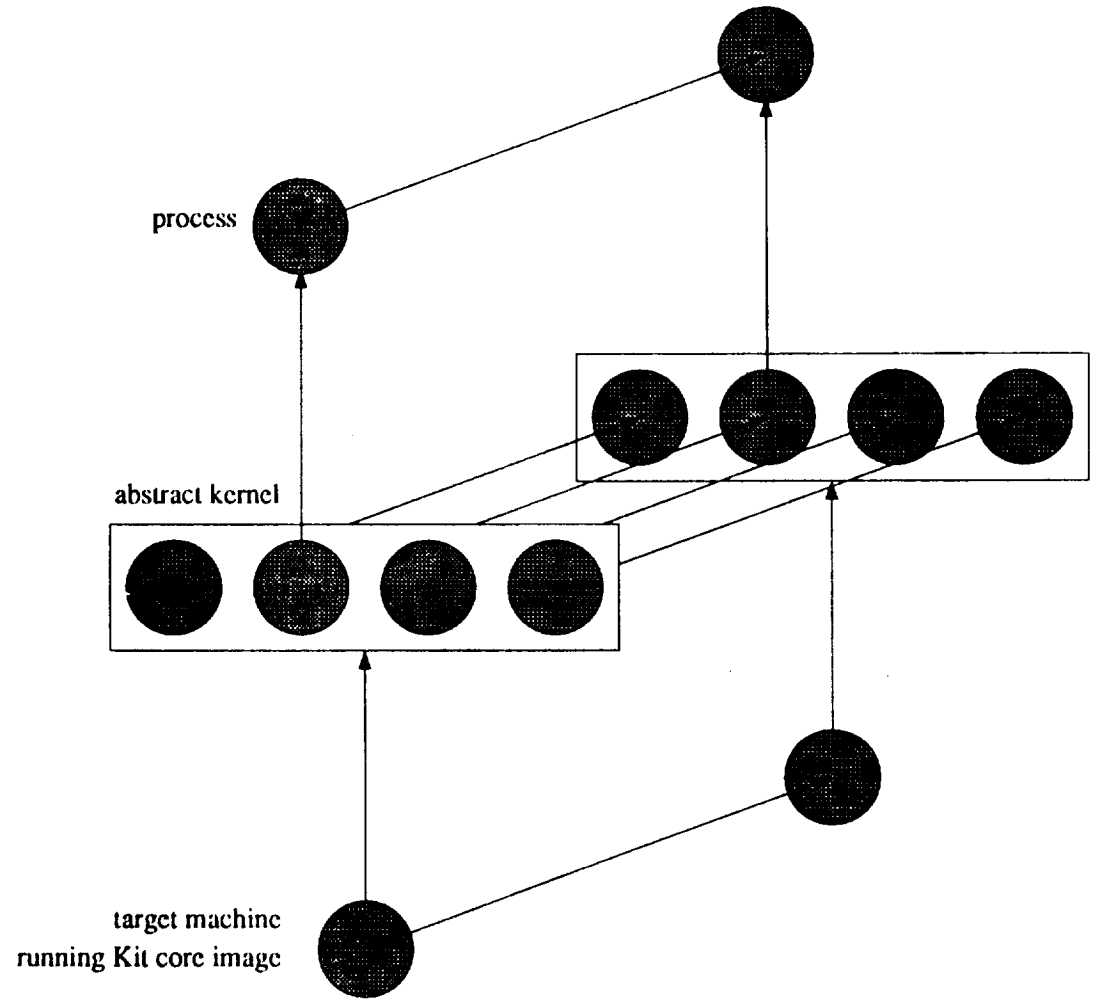
[752 16-bit words]

The Kit Separation Kernel

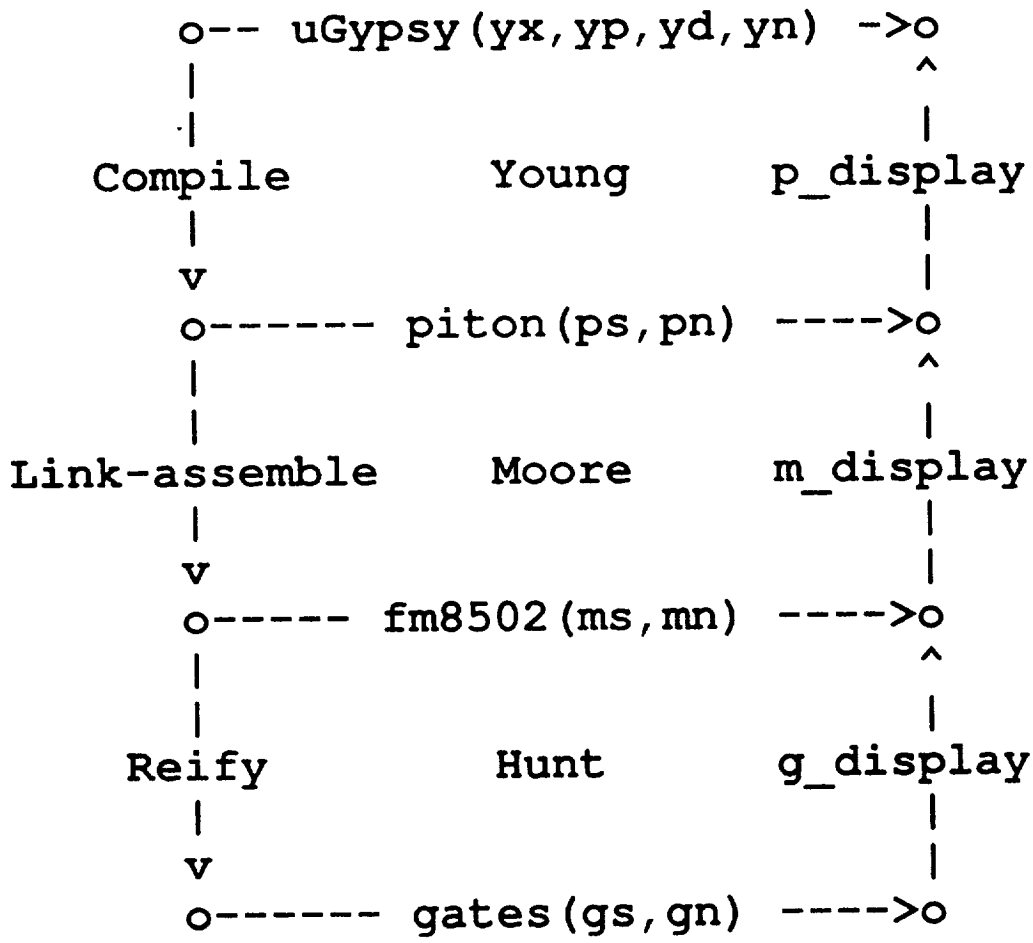
- Uses a modified `FM8501(ms,mn)` machine
- Interrupts for timer and I/O
- Process management
  - fixed number of processes
  - process scheduling (round robin)
  - process communication (message passing)
  - response to error conditions
- Device management for character I/O to asynchronous devices
- Memory management uses hardware protection

William R. Bevier. Kit: A Study in Operating System Verification. IEEE Transactions on Software Engineering. November 1989.

Kit Operating Requirement, R



The CLInc Stack



Warren A. Hunt, J Strother Moore II, William D. Young. Journal of Automated Reasoning. Vol. 5, No. 4, Dec 1989.

The Piton Language

The Piton language has

- execute-only program space
- read/write global arrays
- recursive subroutine calls
- formal parameters
- user-visible stack
- stack-based instructions
- flow-of-control instructions.

The cross assembler produces an FM8502 binary core image.

The Micro Gypsy Language

The Micro Gypsy subset of Gypsy has

- types integer, boolean, character
- one dimensional arrays
- procedure calls with pass by reference parameters
- sequential control structures if, loop,
- condition handling signal..when.

The compiler produces Piton.

The Stack Theorem

```
Theorem: H'(yx,yp,yd,yn) ->
           uGypsy(yx,yp,yd,yn) =
           U'(gates(D'(yx,yp,yd),
                    Kg'(yx,yp,yd,yn,md)))

Proof: Mechanically checked.
```

Under the conditions `H'`,

- the `uGypsy` model is just as accurate as `gates`
- but with many details suppressed by `U'`.

Boyer-Moore Logic

Robert S. Boyer, J Strother Moore II. A Computational Logic Handbook, Academic Press, 1988.

Matt Kaufmann. A User's Manual for an Interactive Enhancement to the Boyer-Moore Theorem Prover. TR 19, Computational Logic, Inc., 1988.

A Hierarchy of Models of a Programmed Machine

```
      R(yx0,yp0,yd0,  ydk)
uGypsy(yx0,yp0,yd0,  yk(yx0,yp0,yd0))
 piton(ps0,          pk(ps0))
fm8502(ms0,          mk(ms0))
 gates(gs0,          gk(gs0))
```

Corresponding to these is a hierarchy of program descriptions....

Operating Requirement

```
procedure mult(var ans:fm8502_int;
                   i,j:fm8502_int) =
begin
ENTRY j ge 0;
EXIT  ans = NTIMES(i,j);
  pending;
end;

type fm8502_int =
     integer[-(2**31)..(2**31)-1];


{A Simple Problem Domain Theory}

function NTIMES(x,y:integer):integer =
begin
exit (assume result =
      if y = 0 then 0
      else if y = 1 then x
           else x + NTIMES(x,y-1)
      fi   fi);
end;
```

Gypsy Program Description

```
procedure mult(var ans:fm8502_int;
                   i,j:fm8502_int) =
begin
ENTRY j ge 0;
EXIT  ans = NTIMES(i,j);
  var k:fm8502_int := 0;
    k := j;
  ans := 0;
  loop
    ASSERT j ge 0 & k in [0..j]
         & ans = NTIMES(i,j-k);
    if k le 0 then leave end;
    ans := ans + i;
      k := k - 1;
  end;
end;
```

Piton Program Description

```
(MG-MULT
 (K ZERO ONE B ANS I J)   ;formals
 NIL                      ;locals
 (PUSH-LOCAL ANS)         ;ans := 0;
 (PUSH-CONSTANT (INT 0))
 (CALL MG-SIMPLE-CONSTANT-ASSIGNMENT)
 (PUSH-LOCAL K)           ;k := j;
 (PUSH-LOCAL J)
 (CALL MG-SIMPLE-VARIABLE-ASSIGNMENT)
 (DL L-1 NIL (NO-OP))     ;loop
 (PUSH-LOCAL B)           ;  b := k le 0
 (PUSH-LOCAL K)
 (PUSH-LOCAL ZERO)
 (CALL MG-INTEGER-LE)
 (PUSH-LOCAL B)           ;  if b then leave
 (FETCH-TEMP-STK)
 (TEST-BOOL-AND-JUMP FALSE L-3)
 (PUSH-CONSTANT (NAT 0))
 (POP-GLOBAL C-C)
 (JUMP L-2)
 (JUMP L-4)
 (DL L-3 NIL (NO-OP))
 (DL L-4 NIL (NO-OP))
 (PUSH-LOCAL ANS)         ;     ans := ans + i;
 (PUSH-LOCAL ANS)
 (PUSH-LOCAL I)
 (CALL MG-INTEGER-ADD)
 (PUSH-GLOBAL C-C)
 ... [14 more support routines] ...
```

FM8502 Program Description

```
(M-STATE
'(B000000000000000000000001011000000 B000000000000000000000001111100011
  B000000000000000000000001111100000 B000000000000000000000010001000111
  B000000000000000000000000000000000 B000000000000000000000000000000000
  B000000000000000000000000000000000 B000000000000000000000000000000000)
F F F F
'(B000000000000011111000001001000001 B000000000000011111000000000100010
  B000000000000011111000001001011011 B000000000000011111000001001011011
  B000000000000011111000000010011000 B000000000000000000000000000000001
  B000000000000001100000000010000010 B000000000000011111000001001101100
  B000000000000011111000000010111011 B000000000000000010000000010100101
  B000000000000011111000000010011000 B000000000000000000000010001001101
  B000000000000011111000000010001100 B000000000000001110000000010000101
  B000000000000011111000001001101100 B000000000000011111000000010011000
  B000000000000000000000000000000000 B000000000000000110000000010000010
  B000000000000011111000001001101100 B000000000000011111000000010111011
  B000000000000001000000010100101 B000000000000011111000000010011000
  B000000000000000000010001001101 B000000000000011111000000010001100
  B000000000000011100000000010000101 B000000000000011111000000110011011
  B000000000000011111000000000001000 B000000000000000000000000000011100
  B000000000000011111000000000001000001 B000000000000011111000000000111010
  B000000000000011111000000000011010 B000000000000011111000001001000001
  B000000000000011111000000000100010 B000000000000011111000001001011011
  B000000000000011111000001001011011 B000000000000011111000000010011000
  B000000000000000000000000000000000 B000000000000000110000000010000010
  B000000000000011111000001001101100 B000000000000011111000000010111011
  B000000000000001000000010100101 B000000000000011111000000010011000
  B000000000000000000010001001101 B000000000000011111000000010001100
  B000000000000011111000000010001100 B000000000000011111000001001011011
  B000000000000011111000001001101100 B000000000000000000000000000000001
  B000000000000000000000000000000010 B000000000000011111000001001101100
  B000000000000011111000000010111011 B000000000000000010000000010100101
  B000000000000011111000000010011000 B000000000000000000000010001001101
  B000000000000011111000000010001100 B000000000000001110000000010000101
  B000000000000011111000001001101100 B000000000000011111000000010011000
  B000000000000000000000000000000010 B000000000000000110000000010000010
  B000000000000011111000001001101100 B000000000000011111000000010111011
  B000000000000000010000000010100101 B000000000000011111000000010011000
  B000000000000000000010001001101101 B000000000000011111000000010001100
  B000000000000011100000000010000101 B000000000000011111000001001101100
  B000000000000011111000000010011011 B000000000000010110000010101100100
  B000000000000010110000000101101011 B000000000000000101100000010111000
  B000000000000000000000000000000001 B000000000000011111000000010011000
  B000000000000000000000000000000000 B000000000000000110000000010000010
  B000000000000011111000001001101100 B000000000000011111000000010111011

  ... [10 more pages] ... ))
```

Mathematical Requirements

- Unambiguous: Requirements have a well-defined interpretation that tells exactly what they do say.
- Analyzable: Do the requirements say the "right" thing?

```
R(x,y) -> good_thing(x,y)
```

- Consistency: Requirements contain no contradictions.
- Enable modeling a program component before building it (and thereby save the time and cost of designing a poor program.)

To get these benefits, the requirements notation must have a rigorous mathematical foundation (semantics).

Design >> Requirements

- There is more to designing a digital system than just stating and refining mathematical requirements.
- One must still construct a program for some machine.
- Mathematical models of commonly used languages and machines are still very scarce.

Summary

For either design of a new system or operation of an old one, mathematical modeling of digital flight control systems offers

Benefits: early error detection

- Saves time
- Saves money
- Saves operational disruption
- Saves operational mishaps

Risks: model misrepresents system

- Inaccurate
- Incomplete

Conventional Non-Wisdom

Use "formal methods" (mathematical modeling)

- only after a system is built to certify it
- only before a system is built to design it
- to guarantee perfect system behavior
- to eliminate the need for testing