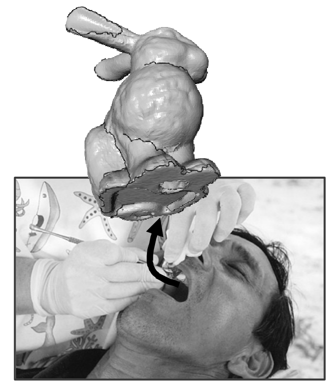
Spin Tutorial Part II
Model Extraction
with FeaVer/Modex



gerard@spinroot.com

LARS

NASA/JPL Laboratory
for Reliable Software

a simple example

to get started

here is a typical little, inscrutable, C program.
does it have bugs, and if so how many?

can we use model checking to find them?

how much work do we have to do to extract a model and see the result?

step 1: define a Modex test harness:

```
$   cat example.prx
%F example.c
%X -x
%L
Import    _all_    _all_
%%
```

from this file

extract all functions

and import all data

```
$   cat example.c
#define N 5
#define M 4

int
main(void)
{   int i, x = 12, y = 34, z, w;
    int *p,*q,*h,*j,*k;
    int *****a, *b[N][M], c[M];

    for (i = 0; i < N; i++)
          b[i][3] = &c[i];

    p = &z;
    q = &x;
    x = 56;
    *p = *q;
    h = p;
    p = &y;
    ****a = h;
    j = ****a;
    h = q;
}
```

run modex

run modex

```
$   ls –l
-rwxrwxrwx  1 g None  302 Aug 10 10:15 example.c
-rwxrwxrwx  1 g None   44 Aug 10 10:08 example.prx
$   modex example.c
$   ls –l
-rw-rw-rw-  1 g None  130 Aug 10 11:39 _modex_.cln
-rw-rw-rw-  1 g None   47 Aug 10 11:39 _modex_.drv
-rw-rw-rw-  1 g None  348 Aug 10 11:39 _modex_.h
-rw-rw-rw-  1 g None   79 Aug 10 11:39 _modex_.lut
-rw-rw-rw-  1 g None  285 Aug 10 11:39 _modex_.run
-rw-rw-rw-  1 g None 1489 Aug 10 11:39 _modex_all.spn
-rw-rw-rw-  1 g None  158 Aug 10 11:39 _modex_all_0.h
-rwxrwxrwx  1 g None  302 Aug 10 10:15 example.c
-rwxrwxrwx  1 g None   44 Aug 10 10:08 example.prx
$   cat _modex_.run
    cpp –E -P _modex_.drv > model
    sh _modex_.cln
    if spin -a model && cc -o pan pan.c
    then    ./pan
    else    exit 1
    fi
    exit 0
$   cpp –E –P _modex_.drv > model
$   cat model
```

view model

the extracted model:

```
$ cat model
c_state "int  ***** a"          "Local main"
c_state "int  * b[5][4]"        "Local main"
c_state "int  * p"              "Local main"
c_state "int  * q"              "Local main"
c_state "int  * h"              "Local main"
c_state "int  * j"              "Local main"
c_state "int  * k"              "Local main"

active proctype main( )
            int c[4];
            int i;
            int x = 12;
            int y = 34;
            int z, w;

            c_code { Pmain->i=0; };

            do
            :: c_expr { (Pmain->i<5) };
                        c_code [(Pmain->i < 5)]
                             { Pmain->b[Pmain->i][3]=&(Pmain->c[Pmain->i]); };
                        c_code { Pmain->i++; }

            :: else ->
                        break
            od;

            c_code { Pmain->p = &(Pmain->z); };
            c_code { Pmain->q = &(Pmain->x); };
            c_code { Pmain->x = 56; };
            c_code [Pmain->q && Pmain->p]  { (*Pmain->p) = (*Pmain->q); };
            c_code { Pmain->h = Pmain->p; };
            c_code { Pmain->p = &(Pmain->y); };
            c_code [Pmain->a&&(*Pmain->a)&&(*((*Pmain->a)))&&(*((*((*Pmain->a)))))]
                 { (*((*((*((*Pmain->a)))))))=Pmain->h; };
            c_code [Pmain->a&&(*Pmain->a)&&(*((*Pmain->a)))&&(*((*((*Pmain->a)))))]
                 { Pmain->j = (*((*((*((*Pmain->a))))))); };
            c_code { Pmain->h = Pmain->q; };
}
```

some variables could be represented as Promela data object; others are imported with c_state declarations

modex generated correctness assertions

original C code:

```
$   cat example.c
#define N   5
#define M   4

int
main(void)
{   int i, x = 12, y = 34, z, w;
    int *p,*q,*h,*j,*k;
    int *****a, *b[N][M], c[M];

    for (i = 0; i < N; i++)
           b[i][3] = &c[i];

    p = &z;
    q = &x;
    x = 56;
    *p = *q;
    h = p;
    p = &y;
    ****a = h;
    j = ****a;
    h = q;
}
```

verification of the model with Spin

```
$   spin –a model
$   cc –o pan pan.c
$   ./pan
pan: (Pmain->i < 4) (at depth 15)
pan: wrote model.trail
(Spin Version 4.2.5 -- 2 April 2005)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 160 byte, depth reached 14, errors: 1
      15 states, stored
       0 states, matched
      15 transitions (= stored+matched)
       0 atomic steps
hash conflicts: 0 (resolved)

2.622   memory usage (Mbyte)
$   ./pan –r
```

one of the generated preconditions fails

the counter-example

the counter-example is generated by pan, not spin, because we use embedded C code, which Spin cannot simulate

```
$ ./pan -r
  1:    proc  0 (main) line  25 "pan_in" (state 1) [ Pmain->i=0; ]
  2:    proc  0 (main) line  27 "pan_in" (state 7) [( (Pmain->i<5) )]
  3:    proc  0 (main) line  29 "pan_in" (state 3) [ Pmain->b[Pmain->i][3]=&(Pmain->c[Pmain->i]); ]
  4:    proc  0 (main) line  30 "pan_in" (state 4) [ Pmain->i++; ]
  5:    proc  0 (main) line  27 "pan_in" (state 7) [( (Pmain->i<5) )]
  6:    proc  0 (main) line  29 "pan_in" (state 3) [ Pmain->b[Pmain->i][3]=&(Pmain->c[Pmain->i]); ]
  7:    proc  0 (main) line  30 "pan_in" (state 4) [ Pmain->i++; ]
  8:    proc  0 (main) line  27 "pan_in" (state 7) [( (Pmain->i<5) )]
  9:    proc  0 (main) line  29 "pan_in" (state 3) [ Pmain->b[Pmain->i][3]=&(Pmain->c[Pmain->i]); ]
 10:    proc  0 (main) line  30 "pan_in" (state 4) [ Pmain->i++; ]
 11:    proc  0 (main) line  27 "pan_in" (state 7) [( (Pmain->i<5) )]
 12:    proc  0 (main) line  29 "pan_in" (state 3) [ Pmain->b[Pmain->i][3]=&(Pmain->c[Pmain->i]); ]
 13:    proc  0 (main) line  30 "pan_in" (state 4) [ Pmain->i++; ]
 14:    proc  0 (main) line  27 "pan_in" (state 7) [( (Pmain->i<5) )]
pan: precondition false: (Pmain->i < 4)
 15:    proc  0 (main) line  29 "pan_in" (state 3) [ Pmain->b[Pmain->i][3]=&(Pmain->c[Pmain->i]); ]
spin: trail ends after 15 steps
local vars proc 0 (main):
        int    c[0]:    0
        int    c[1]:    0
        ...
$
```

example2.c

NOTE: we did not define an example2.prx file ! the contents of example.prx matches the default settings of modex, so we can omit it too

```
$   sed 's/c\[M\]/c[N]/' example.c > example2.c   # correct the array size of c[]
$   modex example2.c
$   cpp –E –P _modex_.drv > model
$   spin –a model
$   cc –o pan pan.c
$   ./pan
pan: Pmain->a && (*Pmain->a) && (*((*Pmain->a))) && (*((*((*Pmain->a))))) (at depth 24)
pan: wrote model.trail
(Spin Version 4.2.5 -- 2 April 2005)
Warning: Search not completed
        + Partial Order Reduction
Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +
State-vector 164 byte, depth reached 23, errors: 1
      24 states, stored
       0 states, matched
      24 transitions (= stored+matched)
       0 atomic steps
...
```

repeat verification

new counter-example
a nil-pointer derefence error

```
$ ./pan -r
  1:    proc  0 (main) line  25 "pan_in" (state 1)      [ Pmain->i=0; ]
  2:    proc  0 (main) line  27 "pan_in" (state 7)      [( (Pmain->i<5) )]
  3:    proc  0 (main) line  29 "pan_in" (state 3)      [ Pmain->b[Pmain->i][3]=&(Pmain->c[Pmain->i]); ]
  4:    proc  0 (main) line  30 "pan_in" (state 4)      [ Pmain->i++; ]
  5:    proc  0 (main) line  27 "pan_in" (state 7)      [( (Pmain->i<5) )]
  6:    proc  0 (main) line  29 "pan_in" (state 3)      [ Pmain->b[Pmain->i][3]=&(Pmain->c[Pmain->i]); ]
  7:    proc  0 (main) line  30 "pan_in" (state 4)      [ Pmain->i++; ]
  8:    proc  0 (main) line  27 "pan_in" (state 7)      [( (Pmain->i<5) )]
  9:    proc  0 (main) line  29 "pan_in" (state 3)      [ Pmain->b[Pmain->i][3]=&(Pmain->c[Pmain->i]); ]
 10:    proc  0 (main) line  30 "pan_in" (state 4)      [ Pmain->i++; ]
 11:    proc  0 (main) line  27 "pan_in" (state 7)      [( (Pmain->i<5) )]
 12:    proc  0 (main) line  29 "pan_in" (state 3)      [ Pmain->b[Pmain->i][3]=&(Pmain->c[Pmain->i]); ]
 13:    proc  0 (main) line  30 "pan_in" (state 4)      [ Pmain->i++; ]
 14:    proc  0 (main) line  27 "pan_in" (state 7)      [( (Pmain->i<5) )]
 15:    proc  0 (main) line  29 "pan_in" (state 3)      [ Pmain->b[Pmain->i][3]=&(Pmain->c[Pmain->i]); ]
 16:    proc  0 (main) line  30 "pan_in" (state 4)      [ Pmain->i++; ]
 17:    proc  0 (main) line  27 "pan_in" (state 7)      [else]
 18:    proc  0 (main) line  34 "pan_in" (state 10)     [ Pmain->p=&(Pmain->z); ]
 19:    proc  0 (main) line  35 "pan_in" (state 11)     [ Pmain->q=&(Pmain->x); ]
 20:    proc  0 (main) line  36 "pan_in" (state 12)     [ Pmain->x=56; ]
 21:    proc  0 (main) line  37 "pan_in" (state 13)     [ (*Pmain->p)=(*Pmain->q); ]
 22:    proc  0 (main) line  38 "pan_in" (state 14)     [ Pmain->h=Pmain->p; ]
 23:    proc  0 (main) line  39 "pan_in" (state 15)     [ Pmain->p=&(Pmain->y); ]
pan: precondition false: Pmain->a && (*Pmain->a) && (*((*Pmain->a))) && (*((*((*Pmain->a)))))
 24:    proc  0 (main) line  40 "pan_in" (state 16)     [ (*((*((*((*Pmain->a)))))))=Pmain->h; ]
spin: trail ends after 24 steps
```

an example with concurrency

http://www.cs.cf.ac.uk/Dave/EXAMPLES/sw_race.c

```
int main(void)
{       thread_t thread_id, main_id;

        main_id = thr_self();
        thr_setconcurrency(2);
        thr_create(NULL, 0, thread_sub, (void *)main_id, THR_SUSPENDED, &thread_id);

        while(1) {
                printf("MAIN: continuing subroutine thread\n"); fflush(stdout);
                thr_continue(thread_id);
                printf("MAIN: suspending self\n"); fflush(stdout);
                thr_suspend(main_id);
        }
        return(0);
}
void *thread_sub(void *arg)
{       thread_t thread_id;
        thread_t main_id = (thread_t) arg;

        thread_id = thr_self();

        while(1) {
                printf("THREAD: continuing main thread\n"); fflush(stdout);
                thr_continue(main_id);
                printf("THREAD: suspending self\n"); fflush(stdout);
                thr_suspend(thread_id);
        }
        return((void *)0);
}
```

can this code deadlock?

yes

using a more specific test harness
to provide the correct Spin abstraction
for thread creation and
suspension and resumption

```
$ modex sw_race.c   # Extract Model
$ sh _modex_.run    # Compile and Run
pan: invalid end state (at depth 17)
pan: wrote model.trail
(Spin Version 4.2.5 -- 2 April 2005)
...
State-vector 16 byte, depth reached 18, errors: 1
...
$ pan –C            # Replay Error Trace
    THREAD: continuing main thread
    THREAD: suspending self
    MAIN: continuing subroutine thread
    THREAD: continuing main thread
    THREAD: suspending self
    MAIN: suspending self
18: main(0):[Suspend_main = 1]
spin: trail ends after 18 steps
#processes 2:
 18:    proc 0 (main)  line   5 (state  7) (invalid end state)
                 Printf("MAIN: continuing subroutine thread\n");
 18:    proc 1 (thread)  line  20 (state  7) (invalid end state)
                 Printf("THREAD: continuing main thread\n");
global vars:
        bit    Suspend_main:    1
        bit    Suspend_thread:  1
        bit    arg:     0
...
```

test harness definition for sw_race

```
$ cat sw_race.prx
%F sw_race.c
%X -n main
%X -n thread_sub
%L
thr_setconcurrency(...          hide
thr_create(...                  hide

thr_suspend(main_id)            Suspend_main = 1
thr_continue(main_id)           Suspend_main = 0

thr_suspend(thread_id)          Suspend_thread = 1
thr_continue(thread_id)         Suspend_thread = 0

main_id=...                     hide
thread_id=...                   hide
%%
%P
#define thread_t        int

bool Suspend_main, Suspend_thread, arg;

active proctype main() provided (!Suspend_main) {
#include "_modex_main.spn"
}

active proctype thread() provided (!Suspend_thread) {
#include "_modex_thread_sub.spn"
}
%%
```

special keyword that can appear on the rhs in the lookup table (defining the mapping/abstractions for the model extraction)

optionally define an explicit Promela context for the extracted models for each function in a %P segment in test harness

special keywords that can appear on the rhs in a modex lookup table definition

Keyword	Meaning
comment	Includes the statement as a comment in the model
hide	Omits the statement from the test model.
skip	Like hide, but replaces statement with a null-step.
keep	Copies the statement without syntax conversions.
print	Includes the statement text in a print statement.
warn	Issues a warning if the left-hand side is encountered.
true	define an expression to always evaluate to true.
false	define an expression to always evaluate to false.
C_code	Encapsulates the statement as is inside the model.
C_expr	Encapsulates and treats as an expression.

examples:

```
printf(...        keep
exit(...          warn
```

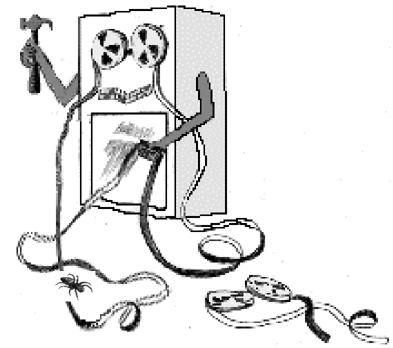
list of the 10 possible test harness commands

Symbol	Single- or Multi- Line	Meaning
%F	single	Set target filename for ANSI C-source.
%X	single	Defines C source procedures to be extracted.
%G	multi	Sets parameters used by the FeaVer GUI, feaver_gui.
%L	multi	Defines a filter (i.e., mapping table) to be used in model extractions via one or more %X commands.
%H	single/ multi	Defines optional header information for generated code.
%D	single/ multi	Declares C data types used in the generated code.
%C	single/ multi	Defines C data declarations used in the generated code.
%P	single/ multi	Defines process infrastructure in Spin modeling lan- guage.
%O	single	Defines extra directives and linked external C-files for final compilation of the generated model checker
%Q	single	Defines extra directives for preprocessing source files from which models are to be extracted.

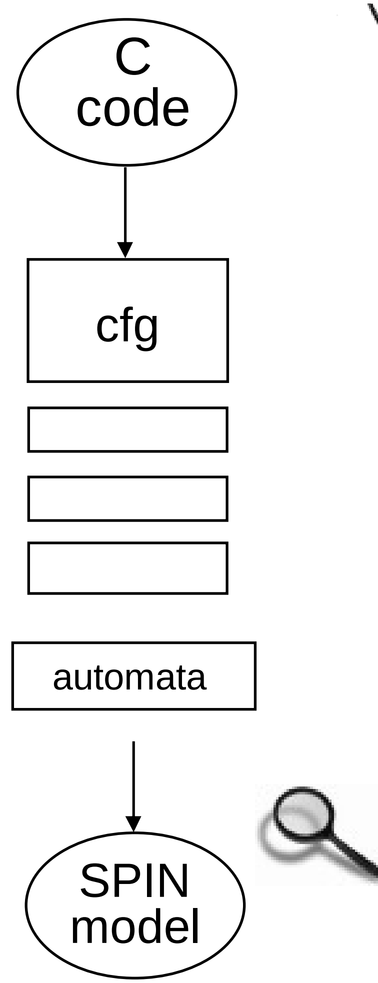
valid parameters to an %X command

Parameter	Meaning
-x	Extract all procedures from the target C-source file.
-a pname	Extract procedure pname as an active proctype.
-i pname	Extract procedure pname as an inline definition.
-p pname	Extract procedure pname as a passive proctype.
-e pname	Extract procedure as an extended active proctype (instru-mented to support a procedure call mechanism).
-E pname	Extract procedure as an extended passive proctype.
-n pname	Extract procedure pname without proctype encapsulation.
-L map-name	Define a map to be used in subsequent model extractions.

the main steps in the model extraction process

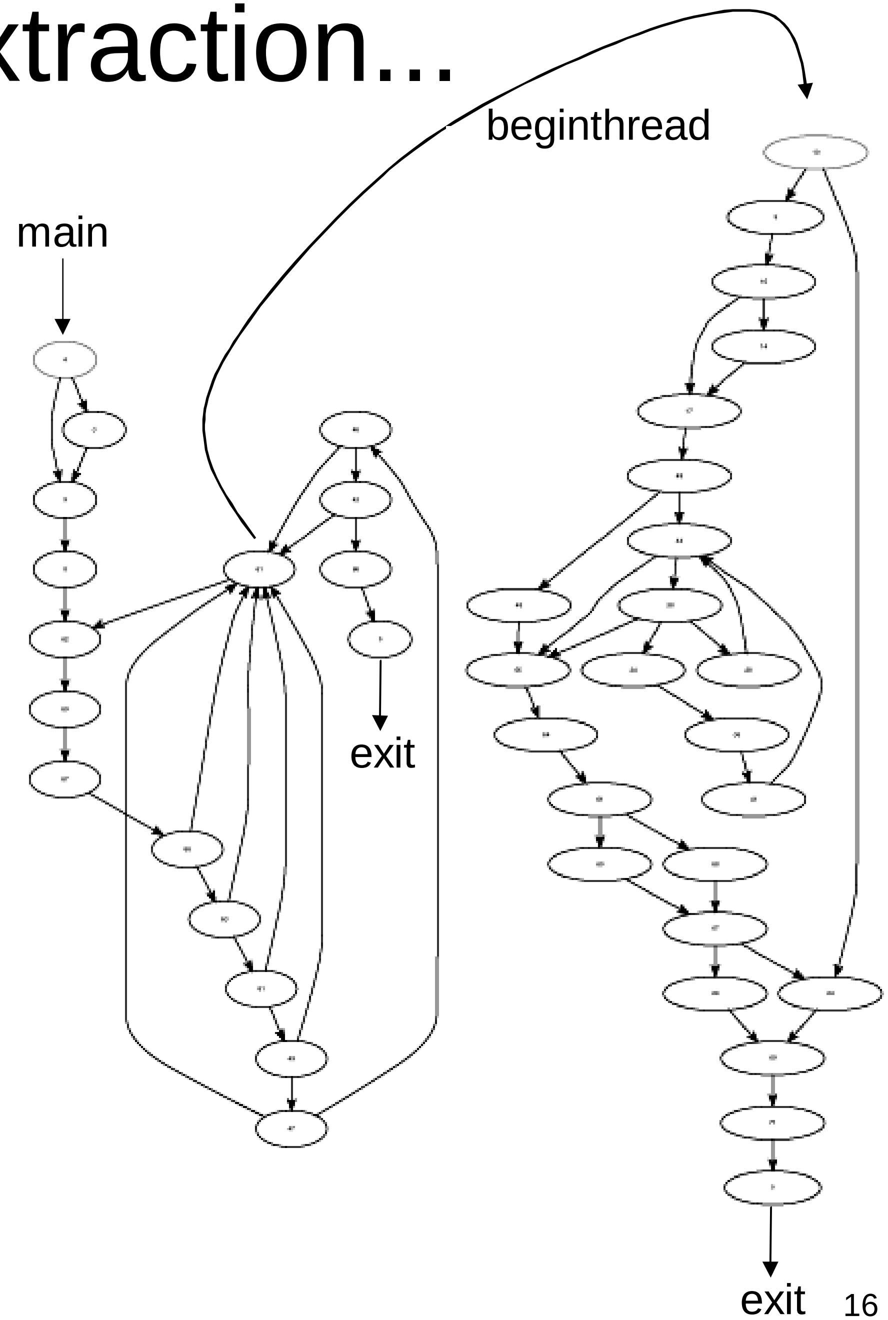


- lexical analysis & parsing
  - produce control flow graph
- complexity control:
  - slicing
  - abstraction
  - restriction etc.
- model generation
  - interprets the cfg
  - applies the controls
  - converts into a Spin automata model



model extraction...

```
int
main(int argc, char *argv[])
{       char buf[512];
        int ns;

        if (!socket setup((unsigned short) TB SCHED))
                exit(1);    /* err msg already given */

        if (argc > 1)
                preload();

        printf("tb_sched: ready...\n");
        fflush(stdout);
        for (;;)
        {       sanity();
                fflush(stdout);
                ns = wait_for_request(buf, sizeof(buf));
                u cnt++;    /* update count - ok if it wraps */

                if (strncmp(buf, "offer ", strlen("offer ")) == 0)
                {       char *n = &buf[strlen("offer ")];
                        char *g = strrchr(buf, ' ');
                        if (g) { *g = '\0'; g++; }
                        add worker(fromaddress, n, g);
                }
                else if (strncmp(buf, "withdraw", strlen("withdraw")) == 0)
                        del worker(fromaddress);
                else if (strncmp(buf, "jobdone", strlen("jobdone")) == 0)
                        job done(fromaddress); /* from prepjob */
                else if (strncmp(buf, "done ", strlen("done ")) == 0)
                        task done(fromaddress, &buf[strlen("done ")]);
                else if (strncmp(buf, "trail ", strlen("trail ")) == 0)
                        do_wrap(fromaddress, &buf[strlen("trail ")]);
                else if (strncmp(buf, "status", strlen("status")) == 0)
                        status();
                else if (strncmp(buf, "restrict", strlen("restrict")) == 0)
                        restricted = 1;
                else if (strncmp(buf, "release", strlen("release")) == 0)
                        restricted = 0;
                else if (strncmp(buf, "exit", strlen("exit")) == 0)
                        break;
                else    /* default */
                        add task(buf);

                 beginthread(assignments, 0, NULL);

                closesocket((SOCKET) (ns-1));
        }
        printf("tb_sched: exits...\n");
        exit(0);
}
```



extensions in Spin v4 that make this possible

(implemented Sep. 2000, added to distribution Jan. 2003)

five new Promela primitives:

data:

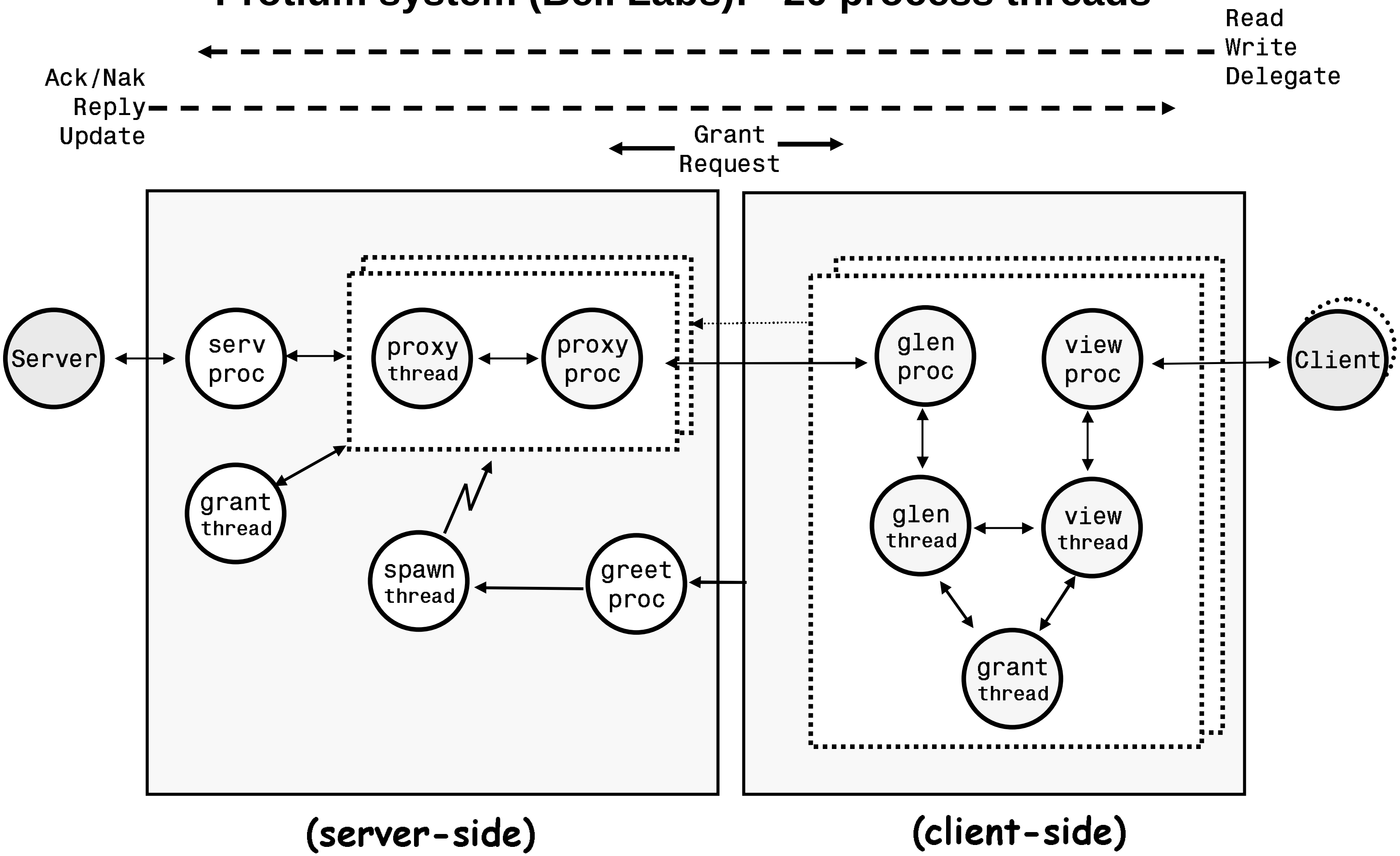
- **`c_decl`** — to define new C data *types* that can be used later in c_state or c_track primitives
- **`c_state`** — to declare new C variables *inside* state vector scoped as: *Global*, *Local,* or *Hidden* *(meant for used by modex only, not humans)*
- **`c_track`** — to report C data objects, declared outside the model (in C code), as holding *state information*

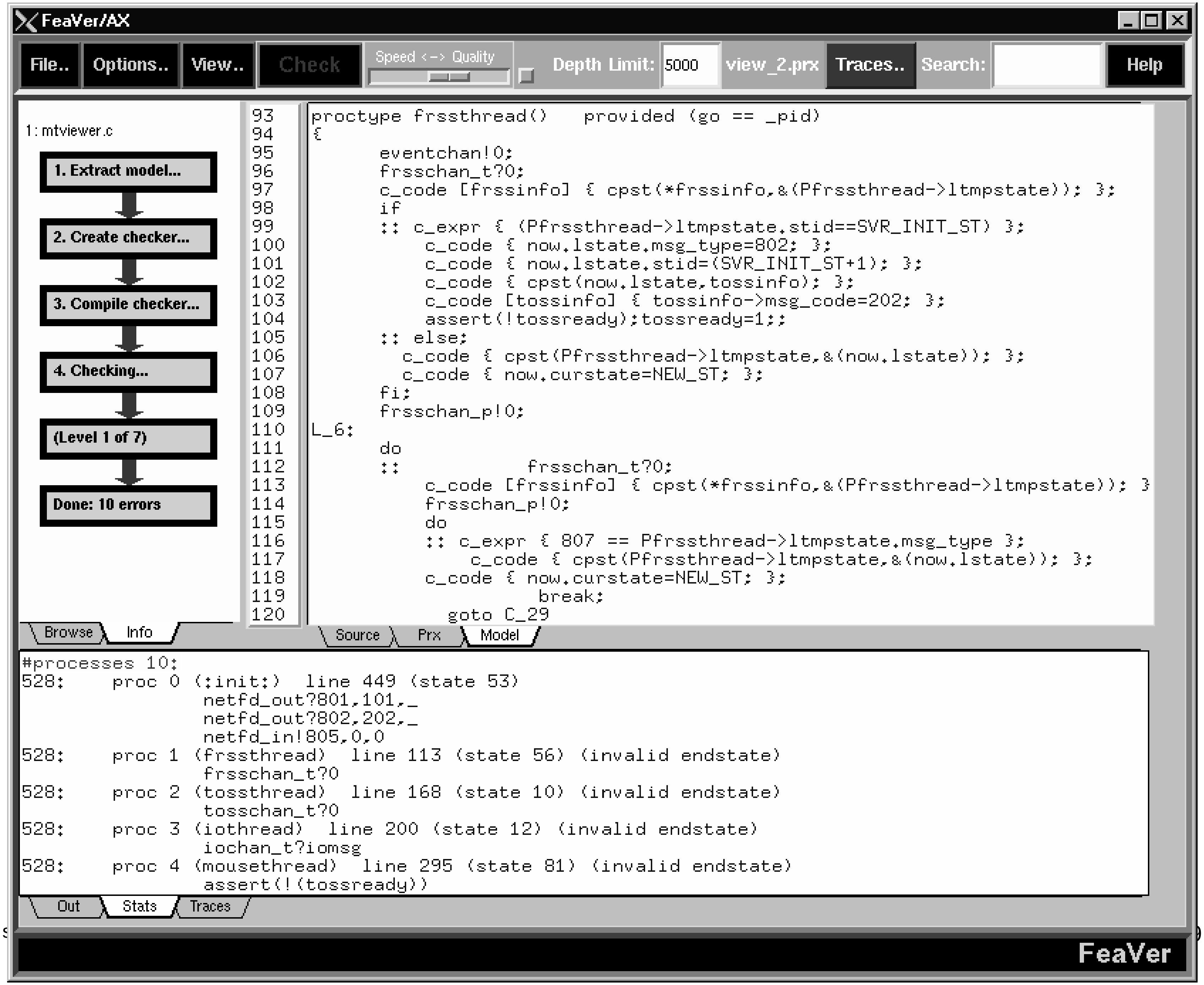
behavior:

- **`c_expr`** — blocking: execute a C expression and use the boolean return value in the model
- **`c_code`** — non-blocking: execute a fragment of C code atomically and deterministically (like a d_step)

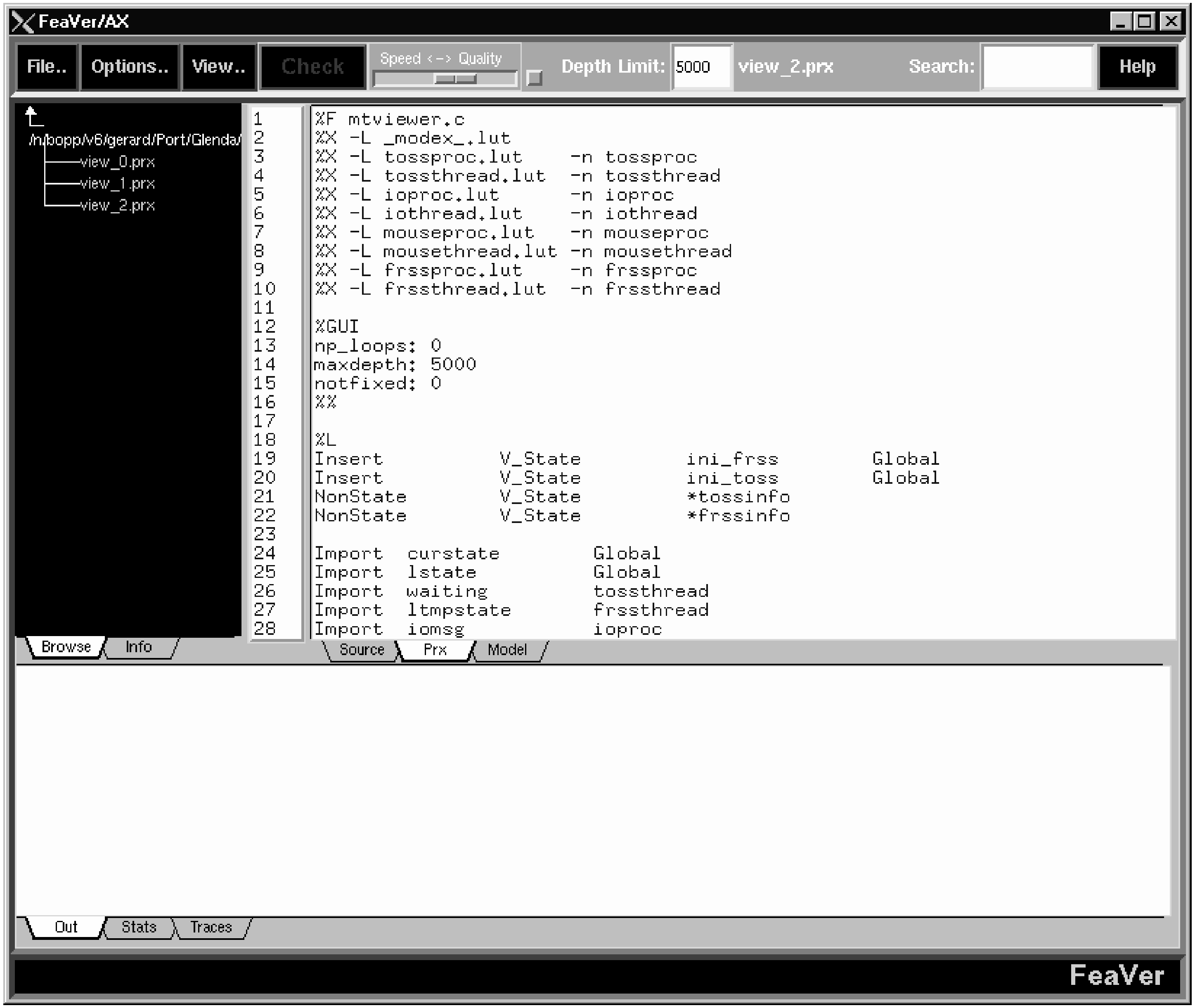
so does it really work on larger applications?

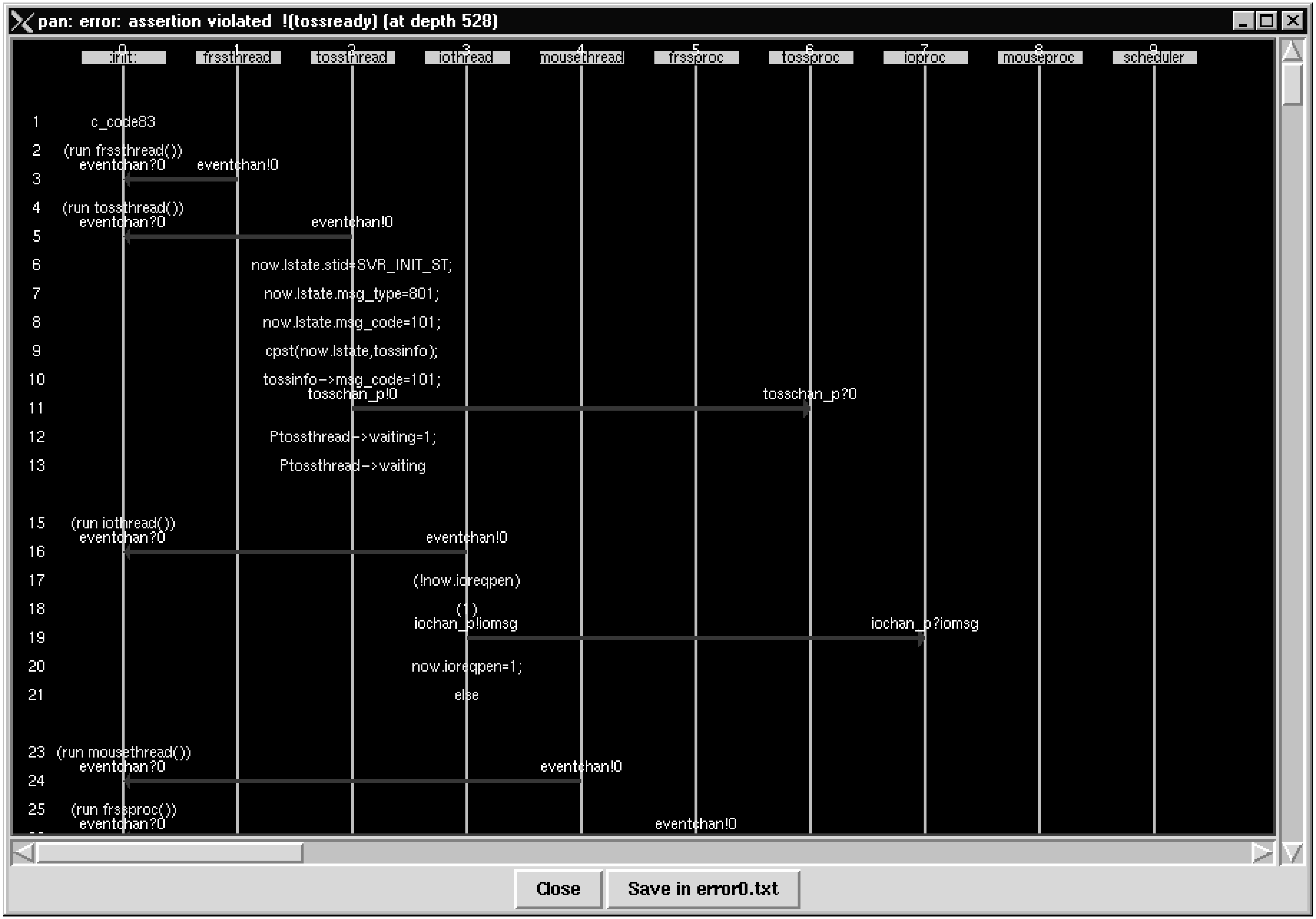
Protium system (Bell Labs): ~20 process threads




FeaVer/AX
File..
Options..
View..
Check
Speed <-> Quality
Depth Limit:
5000
view_2.prx
Traces..
Search:
Help
1: mtviewer.c
1. Extract model...
2. Create checker...
3. Compile checker...
4. Checking...
(Level 1 of 7)
Done: 10 errors
Browse
Info
proctype frssthread()   provided (go == _pid)
{
eventchan!0;
frsschan_t?0;
c_code [frssinfo] { cpst(*frssinfo,&(Pfrssthread->ltmpstate)); };
if
:: c_expr { (Pfrssthread->ltmpstate.stid==SVR_INIT_ST) };
c_code { now.lstate.msg_type=802; };
c_code { now.lstate.stid=(SVR_INIT_ST+1); };
c_code { cpst(now.lstate,tossinfo); };
c_code [tossinfo] { tossinfo->msg_code=202; };
assert(!tossready);tossready=1;;
:: else;
c_code { cpst(Pfrssthread->ltmpstate,&(now.lstate)); };
c_code { now.curstate=NEW_ST; };
fi;
frsschan_p!0;
L_6:
do
::   frsschan_t?0;
c_code [frssinfo] { cpst(*frssinfo,&(Pfrssthread->ltmpstate)); }
frsschan_p!0;
do
:: c_expr { 807 == Pfrssthread->ltmpstate.msg_type };
c_code { cpst(Pfrssthread->ltmpstate,&(now.lstate)); };
c_code { now.curstate=NEW_ST; };
break;
goto C_29
Source
Prx
Model
#processes 10:
528:   proc 0 (:init:) line 449 (state 53)
netfd_out?801,101,_
netfd_out?802,202,_
netfd_in!805,0,0
528:   proc 1 (frssthread) line 113 (state 56) (invalid endstate)
frsschan_t?0
528:   proc 2 (tossthread) line 168 (state 10) (invalid endstate)
tosschan_t?0
528:   proc 3 (iothread) line 200 (state 12) (invalid endstate)
iochan_t?iomsg
528:   proc 4 (mousethread) line 295 (state 81) (invalid endstate)
assert(!(tossready))
Out
Stats
Traces
FeaVer

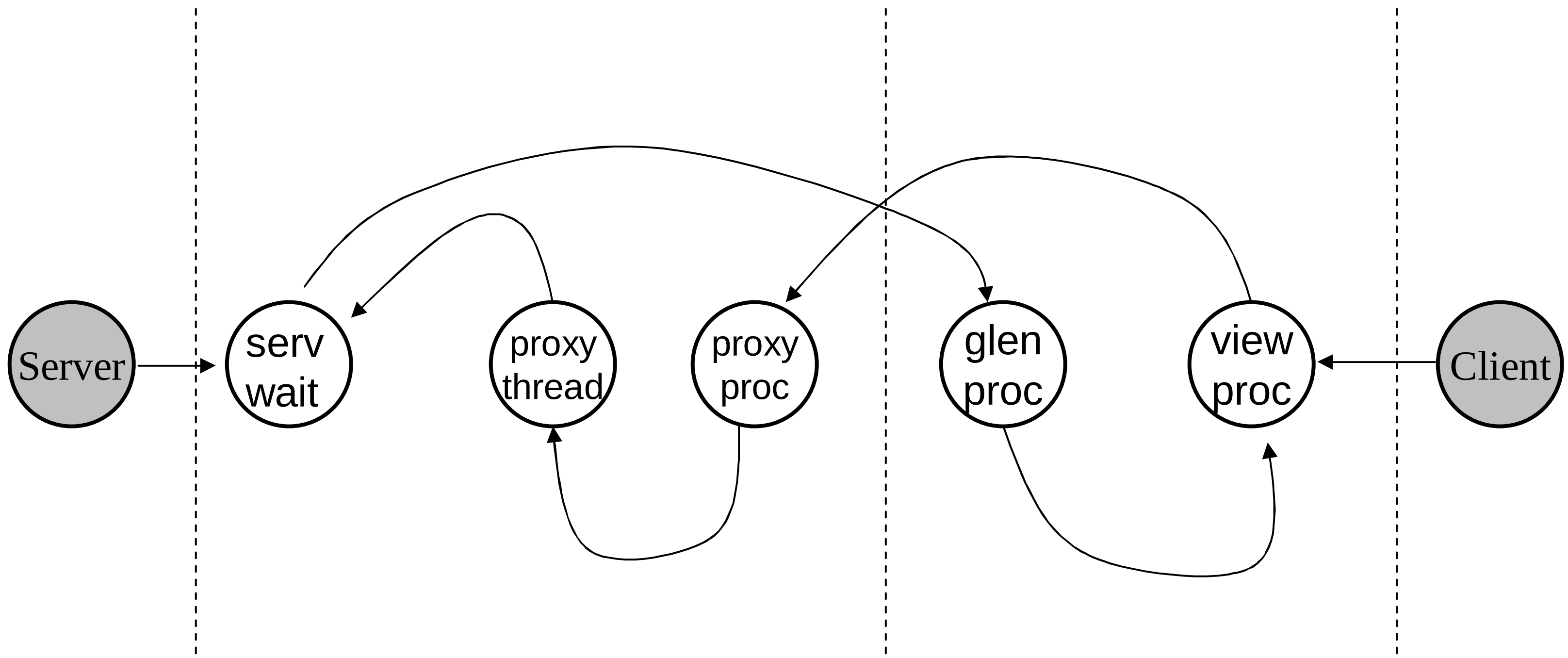
FeaVer Test Harness



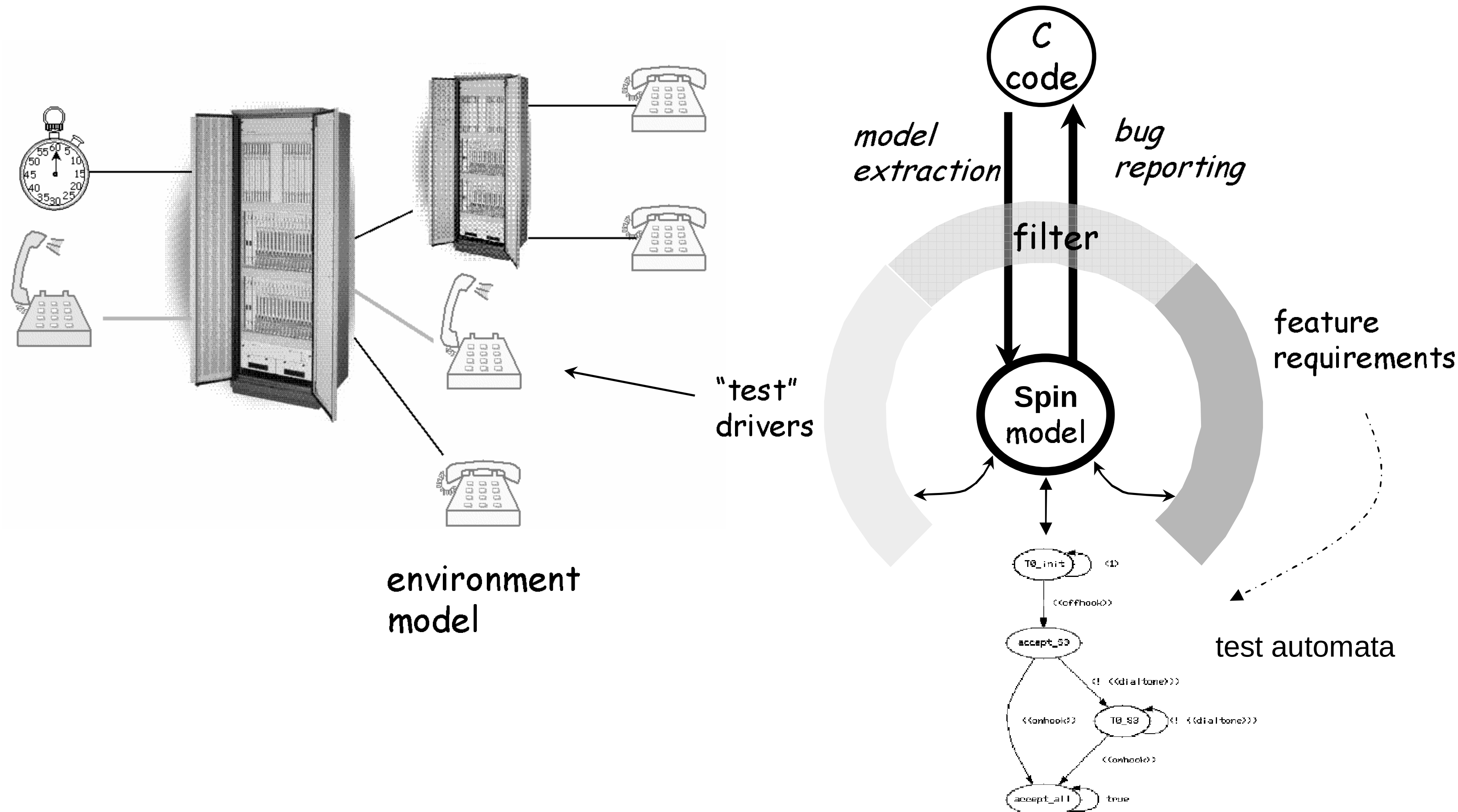

pan: error: assertion violated !(tossready) (at depth 528)
:init:
frssthread
tossthread
iothread
mousethread
frssproc
tossproc
ioproc
mouseproc
scheduler
0
1
2
3
4
5
6
7
8
9
1 c_code83
2 (run frssthread())
eventchan?0
eventchan!0
3
4 (run tossthread())
eventchan?0
eventchan!0
5
6 now.lstate.stid=SVR_INIT_ST;
7 now.lstate.msg_type=801;
8 now.lstate.msg_code=101;
9 cpst(now.lstate,tossinfo);
10 tossinfo->msg_code=101;
tosschan_p!0
tosschan_p?0
11
12 Ptossthread->waiting=1;
13 Ptossthread->waiting
15 (run iothread())
eventchan?0
eventchan!0
16
17 (!now.ioreqpen)
18 (1)
iochan_p!iomsg
iochan_p?iomsg
19
20 now.ioreqpen=1;
21 else
23 (run mousethread())
eventchan?0
eventchan!0
24
25 (run frssproc())
eventchan?0
eventchan!0
Close
Save in error0.txt

circular wait problem detected in ~2 minutes

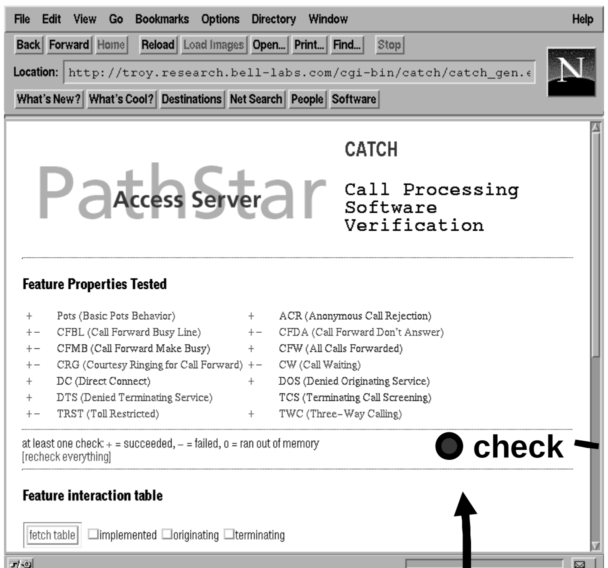
example scenario of 931 steps

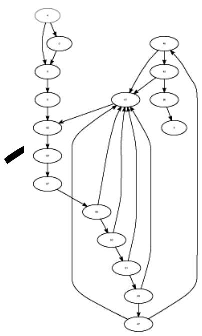


the PathStar verification (1998-2000)



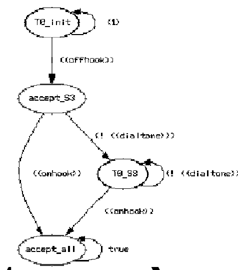
feature verification





extract model of source code

×

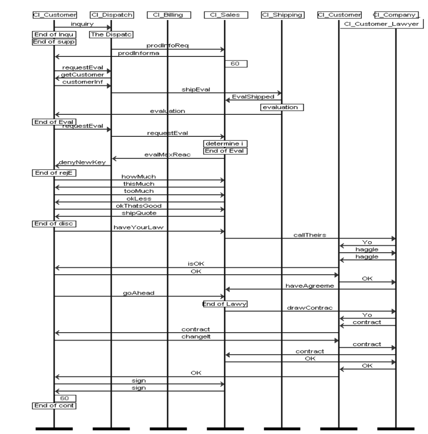


extract model of property

run checks

=

document error-traces



Standard Web Browser Interface

web-based interface
full automation

(Holzmann&Smith BLTJ2000)

documentation
more info
examples
etc.:

http://cm.bell-labs.com/cm/cs/what/modex
http://cm.bell-labs.com/cm/cs/what/feaver

FeaVer 1.0 User Guide

Gerard J. Holzmann and Margaret H. Smith

Preface

FeaVer is a tool that can be used for the verification of distributed systems code written in ANSI-standard C. This guide is meant to be read as a first introduction to the tool and to the methodology on which it is based.

We begin with a brief explanation of the need for a tool that can trap concurrency related errors, followed by an overview of FeaVer's basic mode of operation. The main theme introduced here, and that will return throughout this document, is the definition of a *verification test harness*. The working of FeaVer is always determined by the test harness definition. Learning to use FeaVer, therefore, means learning to design and debug test harness descriptions.

FeaVer can be used either with a basic command-line interface or with the help of a more refined graphical user interface. In some cases the command-line interface is preferable, giving slightly more insight into the details of FeaVer's operation. But undeniably, especially when working with a fully developed test harness description, the graphical user interface can be more convenient. For first-time use, it is recommended to begin by developing a good basic understanding of the main processing steps that FeaVer performs when it runs. In the first few chapters of this guide, therefore, we will focus primarily on the command-line interface and a textual description of the main elements of a test harness. In a later chapter the graphical user interface is covered. A comprehensive review section at the end of this guide, is meant to be a point of reference for the more experienced user of FeaVer when constructing test harness definitions.

At the time of writing the FeaVer tool is being developed as a research prototype, and it continues to evolve. If you detect any information that is confusing or just plain wrong, please let us know and we will correct it.

64 pgs

some papers on modex/feaver

(pdf's are on http://spinroot.com/gerard/pubs.html)

- G.J. Holzmann and M.H. Smith, "*A practical method for the verification of event-driven software*," Proc. Intern. Conf. on Software Engineering, May 1999, Los Angeles, CA, pp. 597-607.
- G.J. Holzmann and M.H. Smith, "*Automating software feature verification*," Bell Labs Technical Journal, Vol. 5, No. 2, April-June 2000, pp. 72-87.
- G.J. Holzmann, "*Logic Verification of ANSI-C Code with Spin*," Proc. SPIN Workshop, Stanford Univ., CA, Aug-Sept. 2000, Springer Verlag, LNCS Vol. 1885, pp. 131-147.
- G.J. Holzmann, "*From Code to Models*," Proc. 2nd Int. Cong. on Concurrency to System Design, Newcastle, U.K., IEEE Computer Society Press, June 2001, pp. 3-10.
- G.J. Holzmann and M.H. Smith, "*An automated verification method for distributed systems software based on model extraction*," IEEE Trans. on Software Engineering, Vol. 28, No. 4, pp. 364-377, April 2002.
- G.J. Holzmann and R. Joshi, "*Model Driven Software Verification*," Proc. 11th Spin Workshop, Barcelona, Spain, April 2004, Springer Verlag, LNCS 2989, pp. 77-92.

synopsis