How I found five lurking race conditions in mnesia with 200 lines of QuickCheck code

John Hughes

CHALMERS

QuviQ

"We know there is a lurking bug somewhere in the dets code. We have got 'bad object' and 'premature eof' every other month the last year. We have not been able to track the bug down since the dets files is repaired automatically next time it is opened.“

Tobbe Törnqvist, Klarna, 2007

What is it?

Invoicing services for web shops

Distributed database: transactions, distribution, replication

Tuple storage

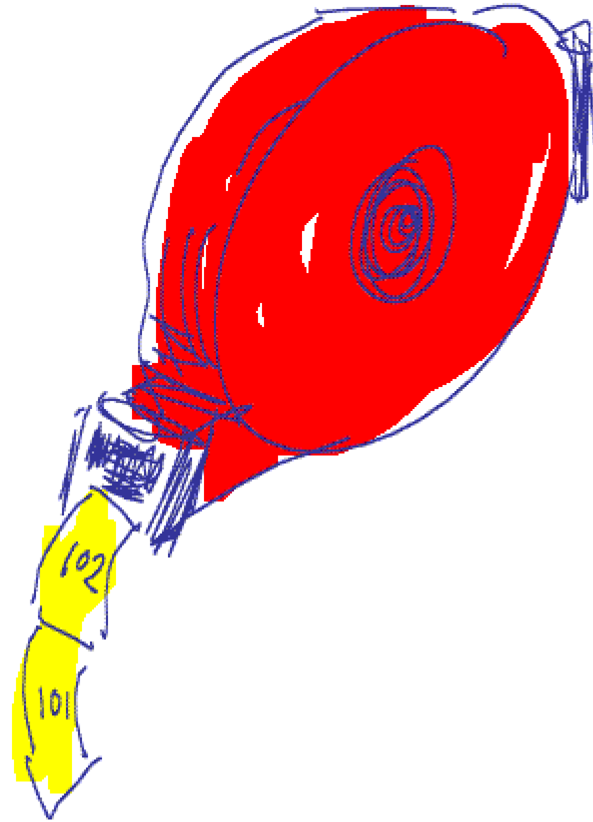
CHALMERS

QuviQ

Imagine Testing This…

dispenser:take_ticket()

dispenser:reset()



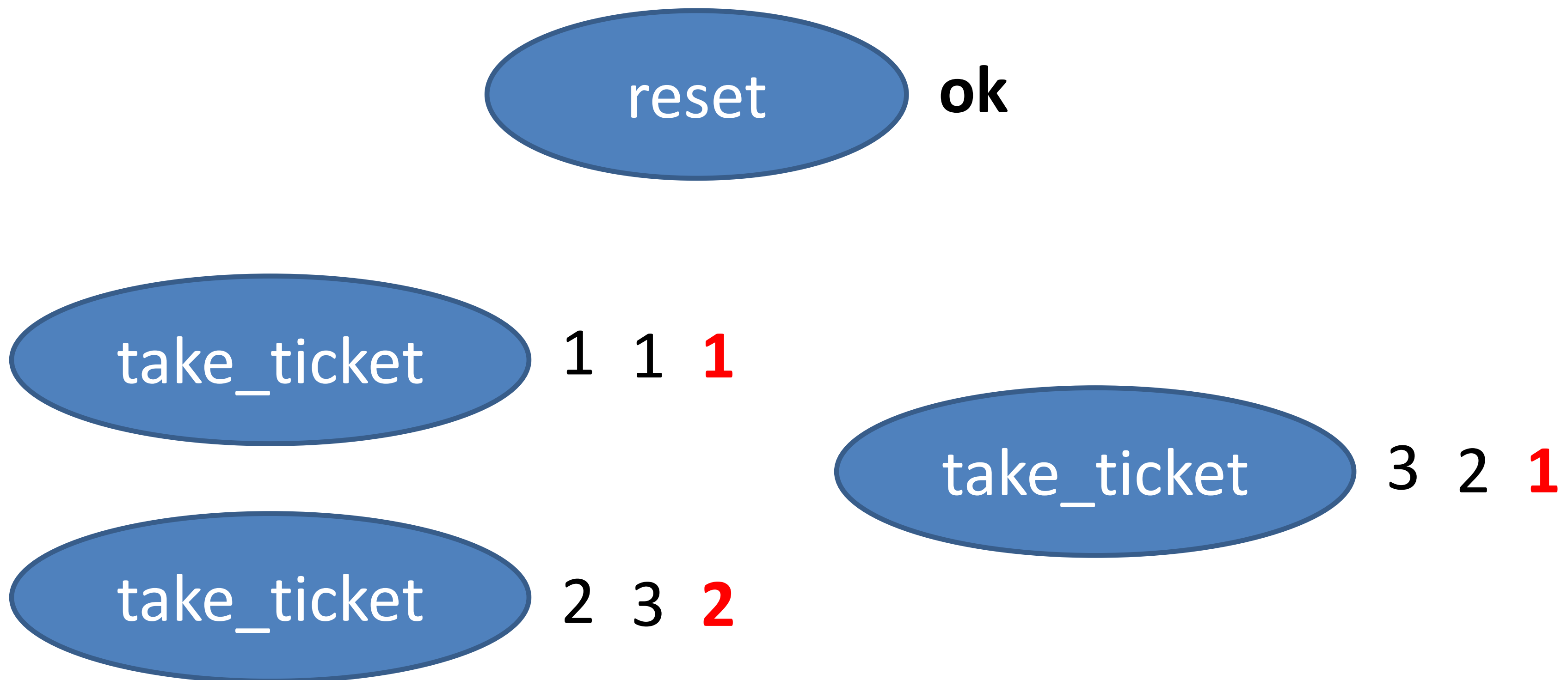
A Unit Test in Erlang

```
test_dispenser() ->
    ok = reset(),
    1  = take_ticket(),
    2  = take_ticket(),
    3  = take_ticket(),
    ok = reset(),
    1  = take_ticket().
```

Expected results

BUT…

A Parallel Unit Test

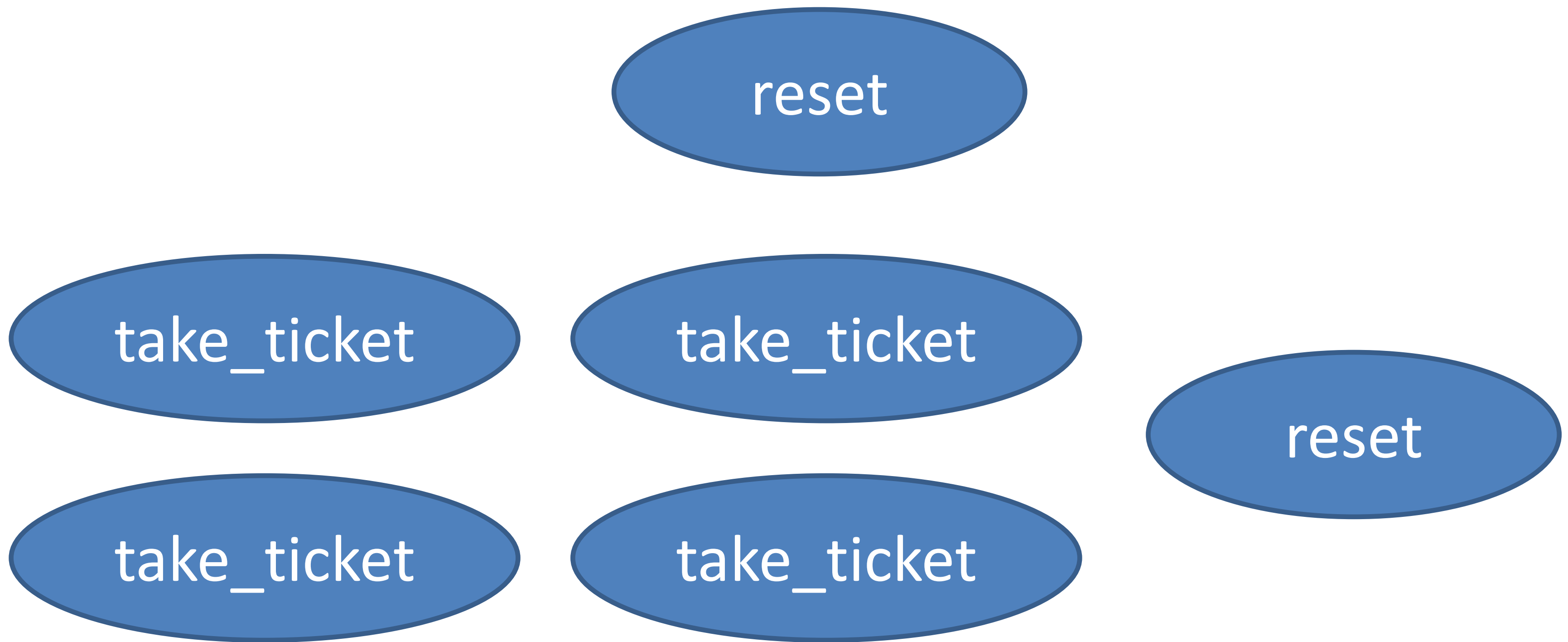


- Three possible correct outcomes!

CHALMERS

QuviQ

Another Parallel Test



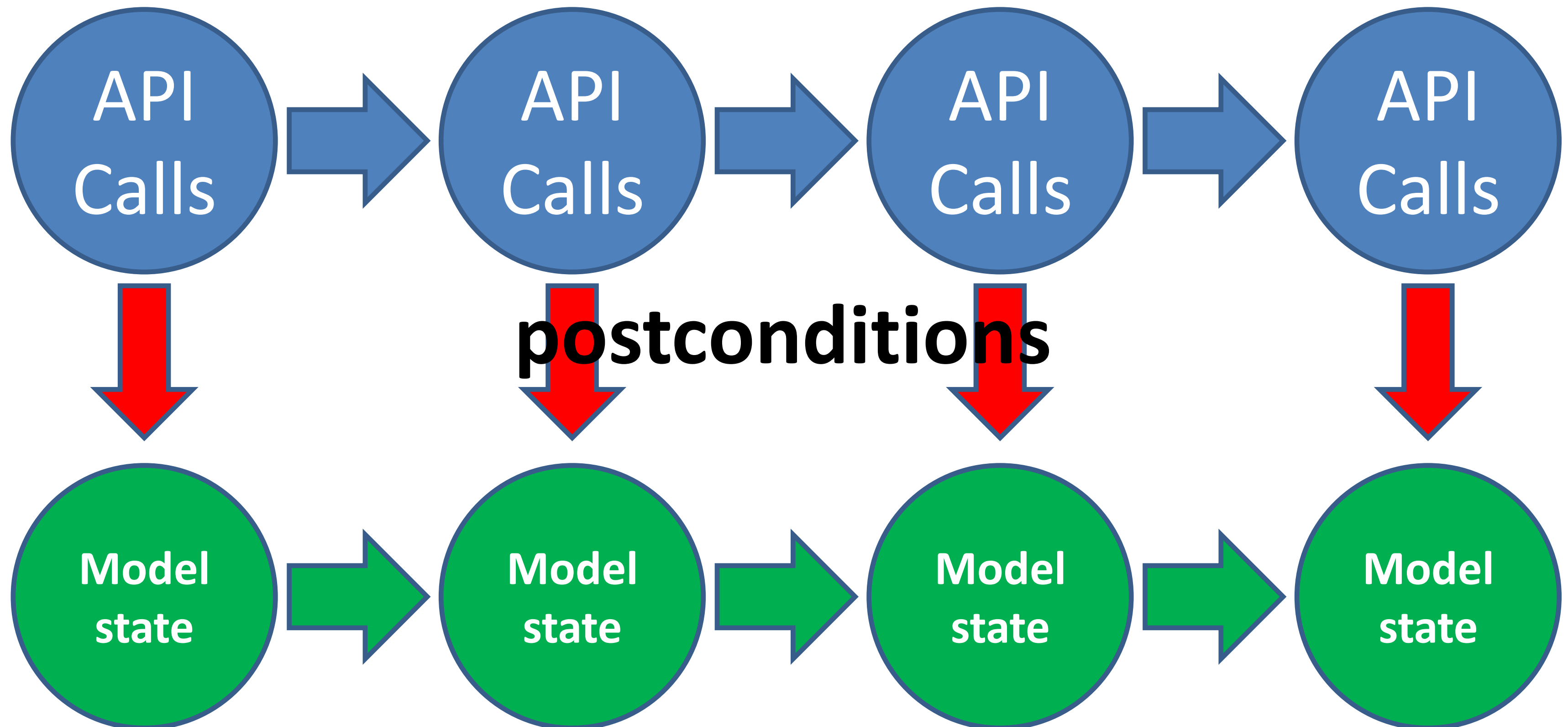
- 42 possible correct outcomes!

Property-Based Testing to the rescue!

- Write *properties* instead of expected outputs
  - e.g. sort([A,B,C]) == [1,2,3]

- Use *models* to decide if tests pass…

QuickCheck State Machine Models

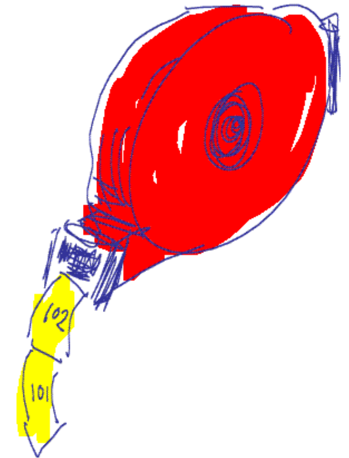


CHALMERS

QuviQ

The Model

- Just an integer!



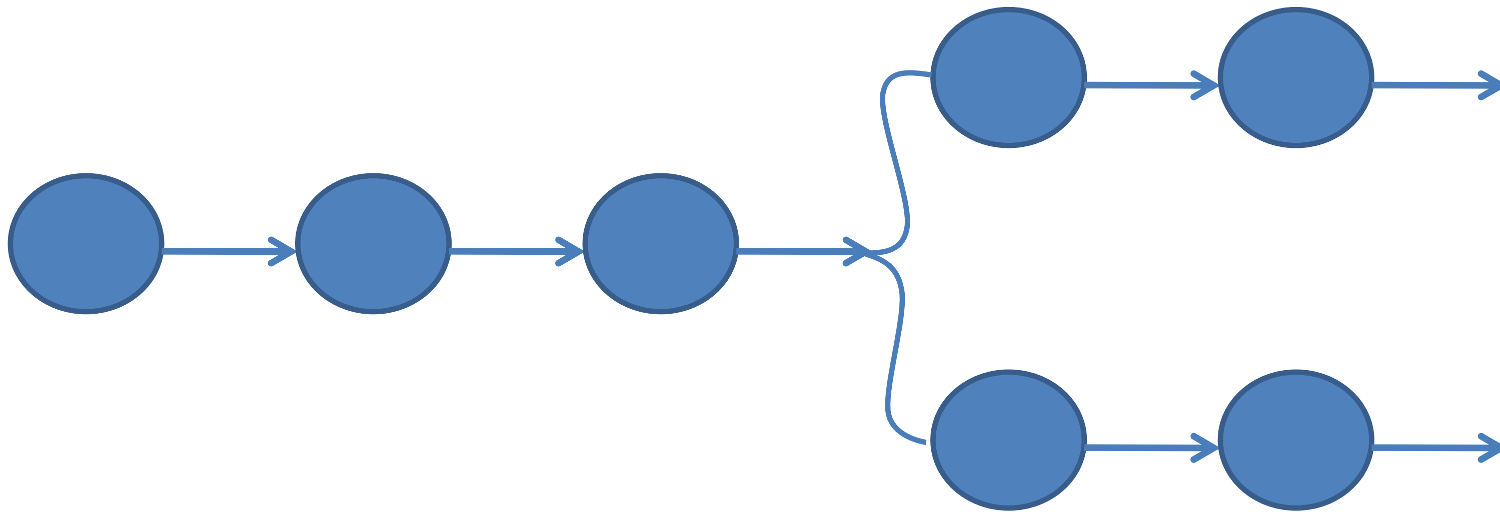
- State transitions

```
next_state(S,_V,{call,_,reset,_}) ->
     0;
next_state(S,_V,{call,_,take_ticket,_}) ->
     S+1.
```

- Postconditions

```
postcondition(S,{call,_,take_ticket,_},Res) ->
     Res == S+1;
```

Parallel Test Cases



- Use the *same* model!

CHALMERS

QuviQ

```
prop_parallel() ->
  ?FORALL(Cmds,parallel_commands(?MODULE),
    begin
      start(),
      {H,Par,Res} =
         run_parallel_commands(?MODULE,Cmds),
      Res == ok)
    end)).
```

Generate parallel test cases

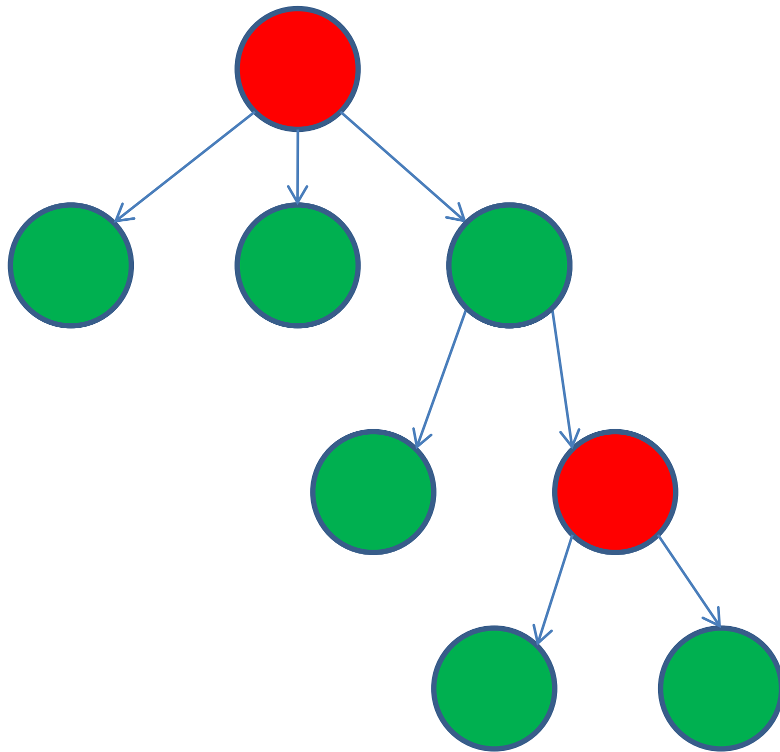
Run tests, check for a matching serialization

CHALMERS

QuviQ

DEMO

- Sometimes:



```
Prefix:
   take_ticket() --> 1
   reset() --> ok
   reset() --> ok
   reset() --> ok
   take_ticket() --> 1
   take_ticket() --> 2
   reset() --> ok
   take_ticket() --> 1

Parallel:
1. take_ticket() --> 2
   take_ticket() --> 3

2. take_ticket() --> 2

Result:
no_possible_interleaving
```

CHALMERS

```
take_ticket() ->
    N = read(),
    write(N+1),
    N+1.
```

Prefix:

Parallel:

1. take_ticket() --> 1

2. take_ticket() --> 1

Result: no_possible_interleaving

dets

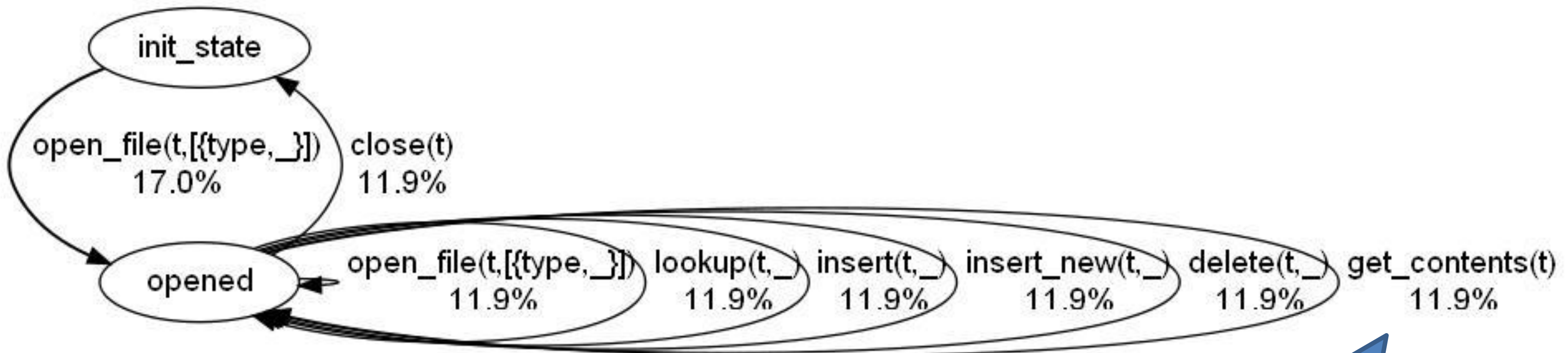
- Tuple store:

  {Key, Value1, Value2…}

- Operations:
  - insert(Table,ListOfTuples)
  - delete(Table,Key)
  - insert_new(Table,ListOfTuples)
  - …
- Model:
  - List of tuples (almost)

FSM for dets Tests

- The state machine specifies the shape of test cases



Derived operation to fetch the entire table

CHALMERS

QuviQ

State Data

- The state data tracks the expected contents of the table

```
-record(state,{contents,type}).
```

A list of tuples

set or bag

State Transitions in the Model

```
next_state_data(_,_,S,_V,{call,_,insert_new,[_,Objs]}) →
    case any_exist(Objs,S#state.contents) of
      true →
        S;
      false →
        S#state{contents=
          model_insert(S#state.type,S#state.contents,Objs)}
    end;
```

CHALMERS

QuviQ

Modelling Operations

```
model_insert(set,S,{K,V}=Obj) →
    lists:keydelete(K,1,S)++[Obj];
model_insert(bag,S,{_,_}=Obj) →
    (S--[Obj])++[Obj];
model_insert(T,S,[Obj|Objs]) →
    model_insert(T,model_insert(T,S,Obj),Objs);
model_insert(_,S,[]) →
    S.
```

List operations make it easy to give *precise* specifications

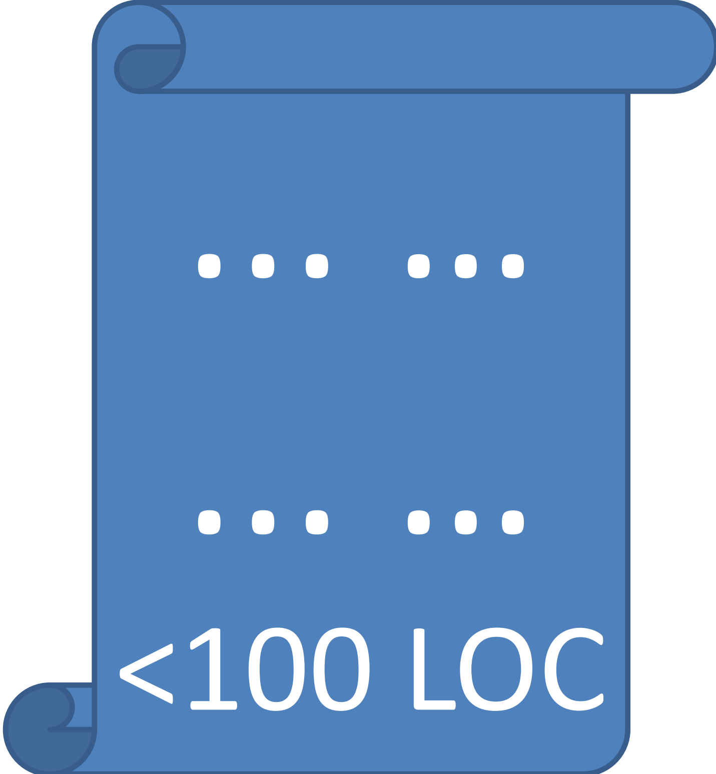
Postconditions

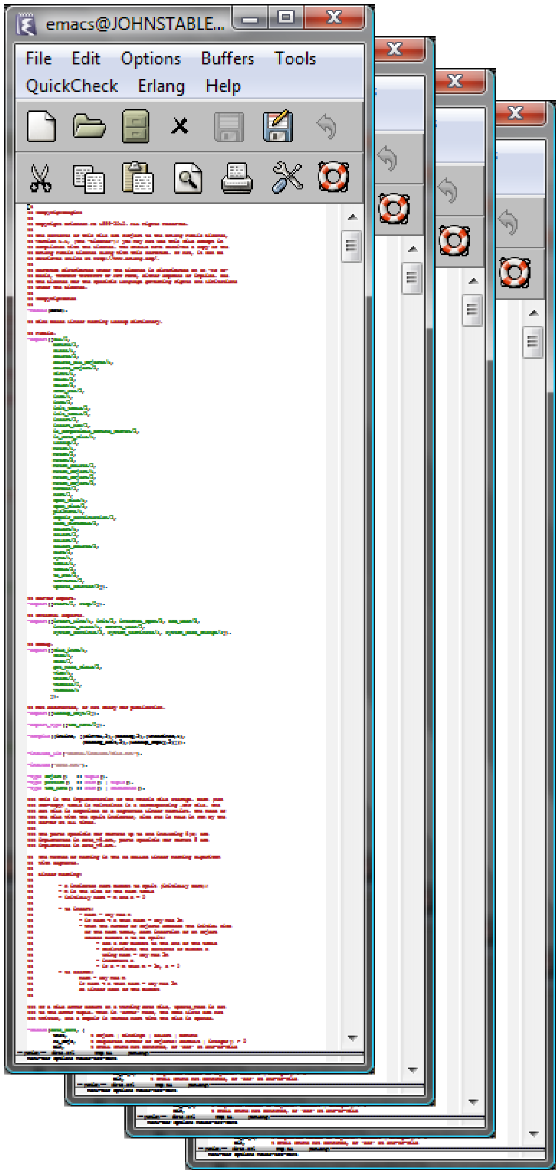
```
postcondition(_,_,S,{call,_,insert_new,[_,Objs]},Res) ->
    Res==not any_exist(Objs,S#state.contents);

postcondition(_,_,S,{call,_,lookup,[_,Key]},Res) ->
    lists:sort(Res) ==
        lists:sort([O || O={K,_} <- S#state.contents,K == Key]);
```

CHALMERS
QuviQ

… … etc etc …





> 6,000 LOC

CHALMERS

QuviQ

DEMO

CHALMERS

QuviQ

Bug #1

```
Prefix:
   open_file(dets

Parallel:
1. insert(dets_ta

2. insert_new(dets_table,[]) --> ok

Result: no_possible_interleaving
```

insert_new(Name, Objects) -> Bool

Types:

Name = name()

Objects = object() | [object()]

Bool = bool()

CHALMERS

QuviQ

Bug #2

```
Prefix:
   open_file(dets_table,[{type,set}]) --> dets_table

Parallel:
1. insert(dets_table,{0,0}) --> ok

2. insert_new(dets_table,{0,0}) --> …time out…
```

=ERROR REPORT==== 4-Oct-2010::17:08:21 ===
** dets: Bug was found when accessing table dets_table

Bug #3

```
Prefix:
   open_file(dets_table,[{type,set}]) --> dets_table

Parallel:
1. open_file(dets_table,[{type,set}]) --> dets_table

2. insert(dets_table,{0,0}) --> ok
   get_contents(dets_table) --> []

Result: no_possible_interleaving
```

Is the file corrupt?

```
corrupted(T) ->
   length(dets:match_object(T,'_'))
   =/=
   dets:info(T,no_objects).
```

Bug #4

```
Prefix:
   open_file(dets_table,[{type,bag}]) --> dets_table
   close(dets_table) --> ok
   open_file(dets_table,[{type,bag}]) --> dets_table

Parallel:
1. lookup(dets_table,0) --> []

2. insert(dets_table,{0,0}) --> ok

3. insert(dets_table,{0,0}) --> ok

Result: ok
```

CHALMERS QuviQ

Bug #5

```
Prefix:
   open_file(dets_table,[{type,set}]) --> dets_table
   insert(dets_table,[{1,0}]) --> ok

Parallel:
1. lookup(dets_table,0) --> []
   delete(dets_table,1) --> ok

2. open_file(dets_table,[{type,set}]) --> dets_table

Result: ok
false
```

CHALMERS

QuviQ

Bug Probabilities

Bug	Failure probability
insert_new wrong return type	0.43%
Insert_new badarg	0.55%
open_file discards concurrent changes	0.32%
premature_eof	0.25%
bad_object	0.10%

CHALMERS

QuviQ

"We know there is a lurking bug somewhere in the dets code. We have got 'bad object' and 'premature eof' every other month the last year."

T[illegible] Tö[illegible]st, Kl[illegible]na, 2007

Each bug fixed the day after reporting the failing case

How come?

- Race conditions are *hard* to write test cases for
  - So people don't!
  - Usually left until *integration testing*
- If it's not tested, why should it work?

Take Home Lesson

Race conditions *can* be found by unit testing—with QuickCheck

CHALMERS

QuviQ

"%%% This is the implementation of the mnesia file storage."

—*dets.erl*