# A CutEr Tool

Kostis Sagonas

# Overview

- Testing
  - demo: unit, property-based, and concolic testing in Erlang
- Concolic execution for Erlang
  - demo
- Support for type specifications
  - short demo
- **CutEr**: A **C**oncolic **U**nit **T**esting tool for **Er**lang
- A “real” experience from using CutEr
  - short demo
- Concluding remarks & future work

# Testing

- Testing is important
- Unit testing is the most widely used method

  Tools: xUnit, EUnit

- Functional languages have mainly explored variants of property-based random testing

  Tools: Haskell QuickCheck, EQC, PropEr, Triq, ...

First demo!

# A first example

- A program unit:

```
classify(L) ->
  case lists:sum(L) of
    S when S < 0 -> negative;
    S when S < 4711 -> small;
    S when S > 4711 -> big;
    _ -> erlang:error(badmatch)
  end.
```

A function that classifies a list of numbers

- In general, pattern matching in Erlang provides a powerful mechanism for program assertions

```
[42,X,X|_] = f(...)
```

# Testing

- In imperative languages, researchers have argued for the benefits of concolic testing
  - Fully automatic testing approach
  - **Concolic** = **Conc**rete + Symb**olic**
  - Aims to achieve high path coverage

  Tools: DART, CUTE, Symbolic Java PathFinder, jCUTE, SAGE, ...

# Concolic execution

- Also known as dynamic symbolic execution
- **Main idea:**
  - during concrete execution, collect symbolic constraints on program inputs that cause the program to follow a specific execution path and
  - use these constraints to force execution of other paths
- **Properties/advantages:**
  - concrete execution makes available accurate information about program state which may not be easily accessible when using e.g. random testing or static analysis techniques

# Implementation of concolic execution

- Symbolic execution is enabled by instrumenting the program with code that collects *path constraints* without disrupting its concrete execution
- Each variable that depends on input has both a concrete and a symbolic value associated to it
- Path constraints are expressed in an appropriate logic
- Off-the-shelf constraint solvers, often SMT ones, are used to solve these constraints and generate new inputs that will steer the future test runs to explore unexplored paths
- The execution paths can be expressed as a *symbolic execution tree*
  - each leaf node has a path constraint describing the input values that force the program to follow that specific path

# A second example

```
-module(sf2).
-export([foo/1]).

foo(L) ->
  lists:foreach(fun fcmp/1, L).

fcmp(X) ->
  case cmp(X) of
    gt -> ok;
    lt -> ok
  end.

cmp(X) when X > 42 -> gt;
cmp(42) -> eq;
cmp(X) when X < 42 -> lt.
```

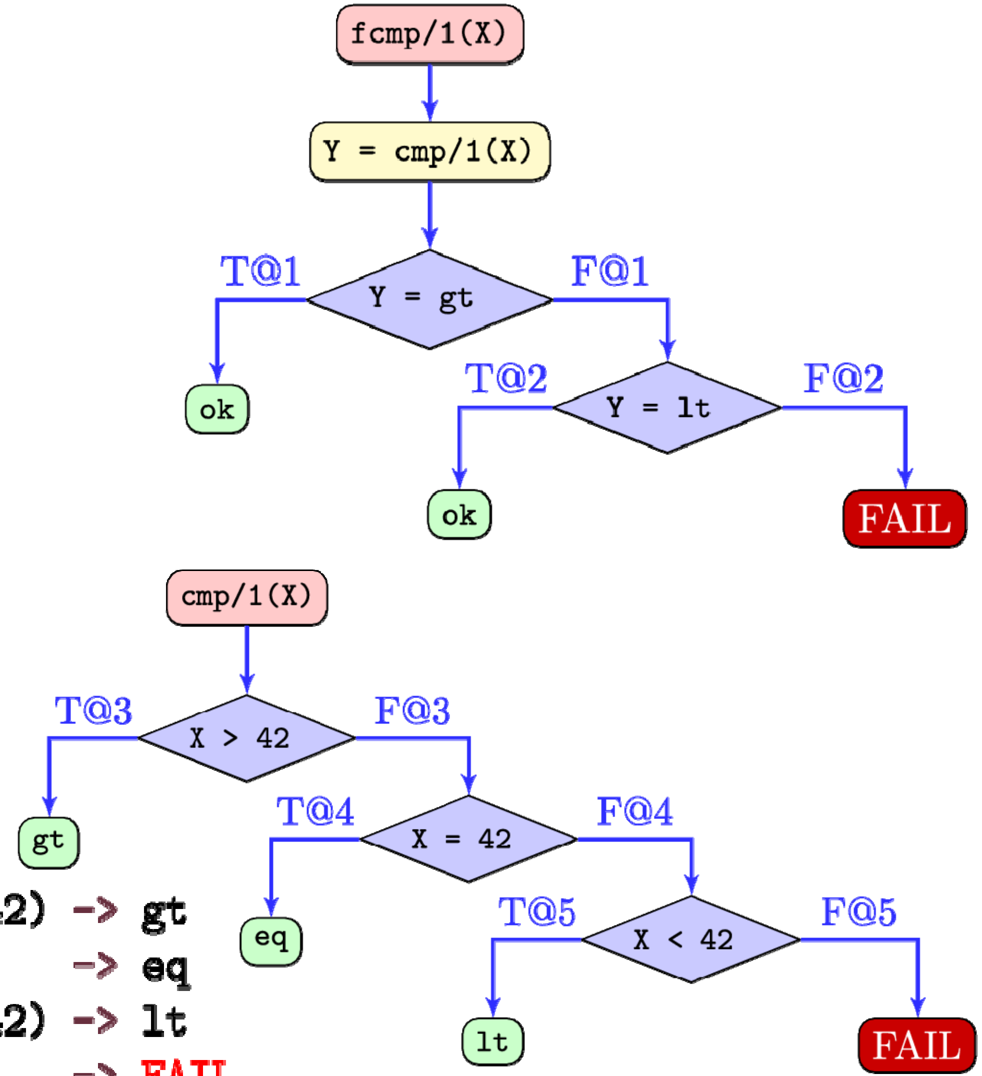
Second demo!

# Second example in Core Erlang

```
module example [foo/1] =
  foo/1 = fun (_cor0) ->
    call lists:foreach (fcmp/1, _cor0)

  fcmp/1 = fun (_cor0) ->
    case <apply cmp/1 (_cor0)> of
      <gt>     when true -> ok
      <lt>     when true -> ok
      <_cor1> when true -> FAIL
    end

  cmp/1 = fun (_cor0) ->
    case <_cor0> of
      <X>      when call erlang:'>' (_cor0, 42) -> gt
      <42>     when true                        -> eq
      <X>      when call erlang:'<' (_cor0, 42) -> lt
      <_cor1> when true                         -> FAIL
    end
```

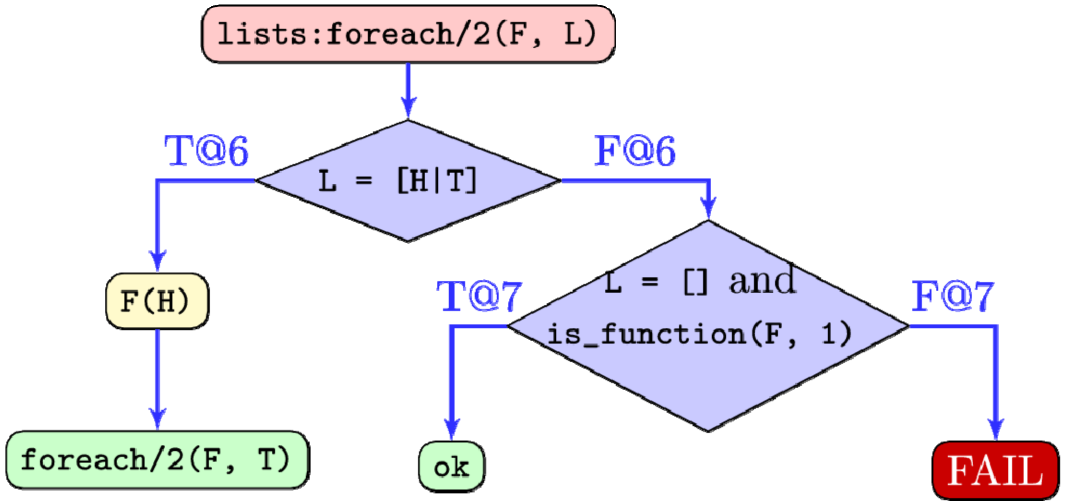
# Control flow graphs of functions

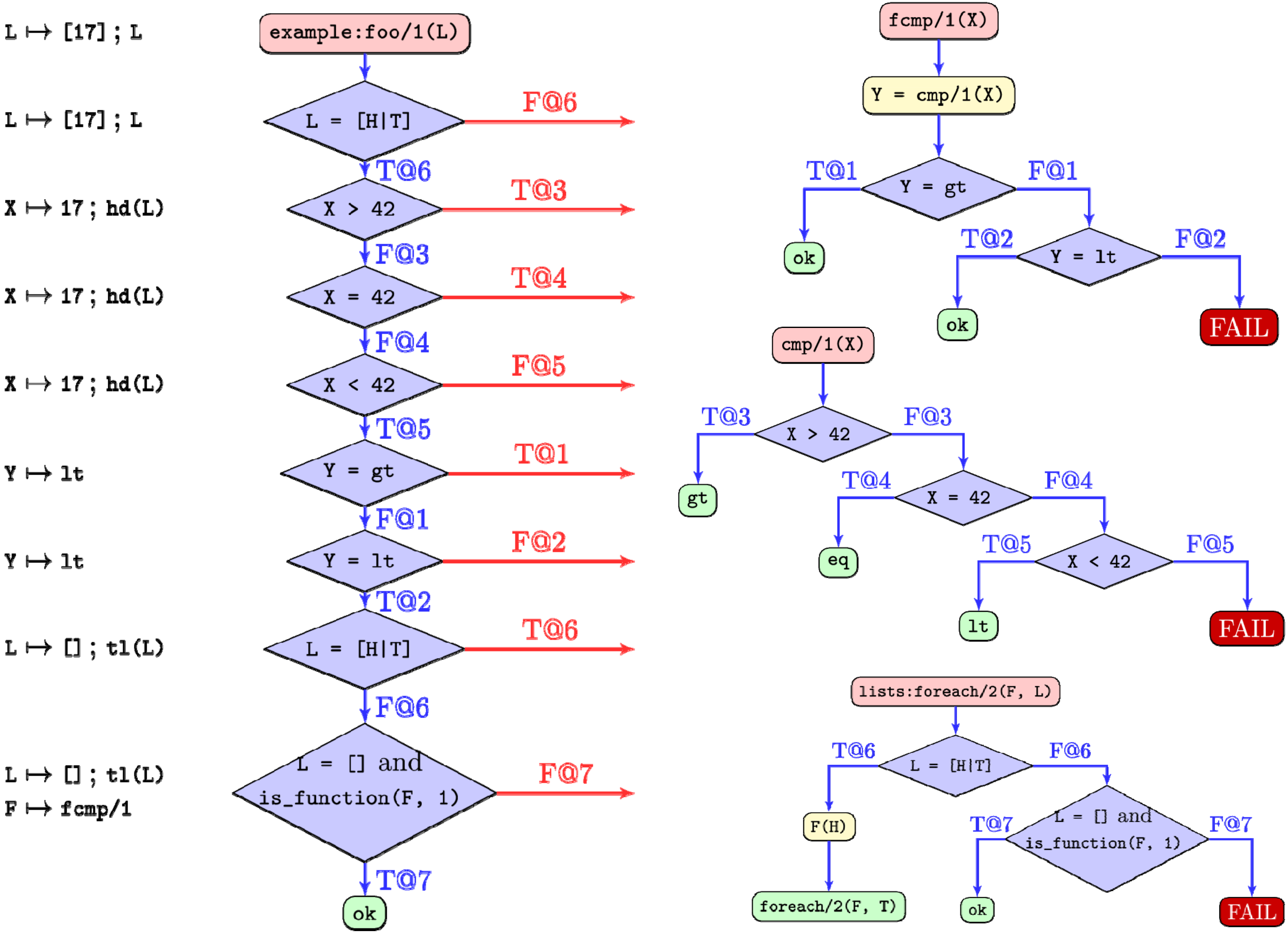
```
module example [foo/1] =
  foo/1 = fun (_cor0) ->
    call lists:foreach (fcmp/1, _cor0)

  fcmp/1 = fun (_cor0) ->
    case <apply cmp/1 (_cor0)> of
      <gt>     when true -> ok
      <lt>     when true -> ok
      <_cor1>  when true -> FAIL
    end

  cmp/1 = fun (_cor0) ->
    case <_cor0> of
      <X>      when call erlang:'>' (_cor0, 42) -> gt
      <42>     when true                         -> eq
      <X>      when call erlang:'<' (_cor0, 42) -> lt
      <_cor1>  when true                         -> FAIL
    end
```



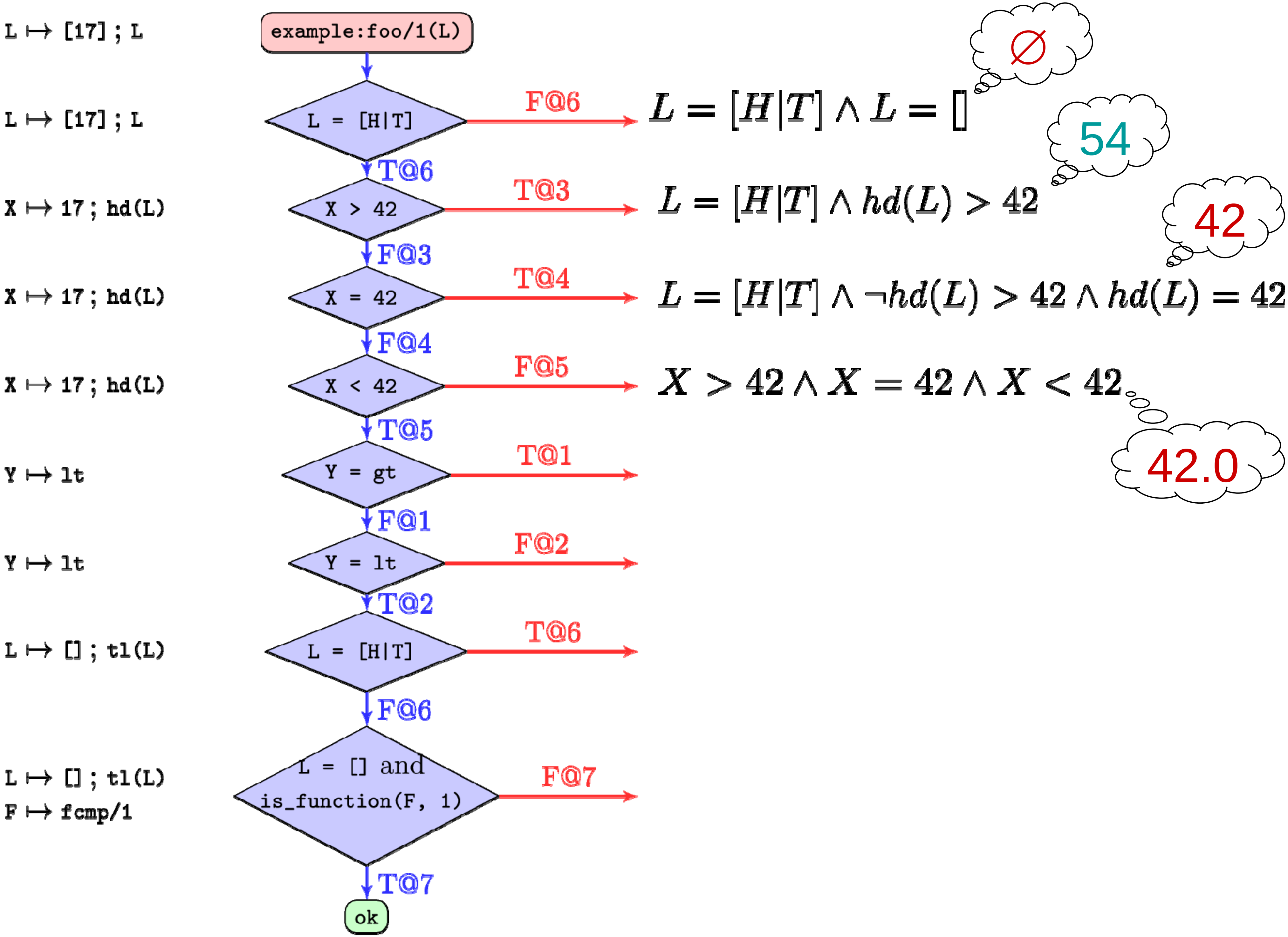
# Control flow graphs of functions

```
module lists [..., foreach/2, ...] =
  ...
  foreach/2 = fun (_cor1,_cor0) ->
    case <_cor1,_cor0> of
      <F,[H|T]> when true ->
        do apply F (H)
           apply foreach/2 (F, T)
      <F,[]> when call erlang:is_function (_cor0, 1) -> ok
      <_cor3,_cor2> when true                         -> FAIL
    end
  ...
```





L ↦ [17] ; L
L ↦ [17] ; L
X ↦ 17 ; hd(L)
X ↦ 17 ; hd(L)
X ↦ 17 ; hd(L)
Y ↦ lt
Y ↦ lt
L ↦ [] ; tl(L)
L ↦ [] ; tl(L)
F ↦ fcmp/1
example:foo/1(L)
L = [H|T]
F@6
T@6
X > 42
T@3
F@3
X = 42
T@4
F@4
X < 42
F@5
T@5
Y = gt
T@1
F@1
Y = lt
F@2
T@2
L = [H|T]
T@6
F@6
L = [] and
is_function(F, 1)
F@7
T@7
ok
fcmp/1(X)
Y = cmp/1(X)
T@1
Y = gt
F@1
ok
T@2
Y = lt
F@2
ok
FAIL
cmp/1(X)
T@3
X > 42
F@3
gt
T@4
X = 42
F@4
eq
T@5
X < 42
F@5
lt
FAIL
lists:foreach/2(F, L)
T@6
L = [H|T]
F@6
F(H)
T@7
L = [] and
is_function(F, 1)
F@7
foreach/2(F, T)
ok
FAIL



L ↦ [17] ; L
L ↦ [17] ; L
X ↦ 17 ; hd(L)
X ↦ 17 ; hd(L)
X ↦ 17 ; hd(L)
Y ↦ lt
Y ↦ lt
L ↦ [] ; tl(L)
L ↦ [] ; tl(L)
F ↦ fcmp/1
example:foo/1(L)
L = [H|T]
F@6
T@6
X > 42
T@3
F@3
X = 42
T@4
F@4
X < 42
F@5
T@5
Y = gt
T@1
F@1
Y = lt
F@2
T@2
L = [H|T]
T@6
F@6
L = [] and is_function(F, 1)
F@7
T@7
ok
$L = [H|T] \wedge L = []$
∅
54
$L = [H|T] \wedge hd(L) > 42$
42
$L = [H|T] \wedge \neg hd(L) > 42 \wedge hd(L) = 42$
$X > 42 \wedge X = 42 \wedge X < 42$
42.0

# Search strategy

- Which decision node to reverse?
- We use two metrics:
  - If a decision node exists whose reversed (red) label has not yet been visited, reverse it
  - Else reverse the decision node which is closer to the root
- Stop when there are no decision nodes left to reverse

# Depth-bounded search

- Depth counts **case** constructs that precede the decision node

- All constraints related to patterns and guards of a specific **case** construct are considered to be at the same depth

- Prune decision nodes whose depth exceeds a threshold

# Support for type specifications

- Type specifications impose additional constraints on program inputs
- For the first demo program:

```erlang
-type ret() :: 'negative' | 'small' | 'big'.
-spec classify([number()]) -> ret().
```

- For the second demo program:

```erlang
-spec foreach(fun((T) -> term()), [T]) -> ok.
```

```erlang
-spec foo([term()]) -> ok.
-spec foo([integer()]) -> ok.
```

```
-spec foo([term()]) -> ok.
```

$$\boxed{is\_list(L)} \land L = [H|T] \land hd(L) > 42$$

$$\boxed{is\_list(L)} \land L = [H|T] \land hd(L) > 42 \land hd(L) = 42$$

$$\boxed{is\_list(L)} \land L = [H|T] \land L = []$$

$$\boxed{is\_list(L)} \land X = hd(L) \land X > 42 \land X = 42 \land X < 42$$

```
-spec foo([integer()]) -> ok.
```

$$\boxed{is\_integer\_list(L) \implies L = [] \lor is\_integer(hd(L)) \land is\_integer\_list(tl(L))}$$

$$X = hd(L) \land \boxed{is\_integer(X)} \land X > 42 \land X = 42 \land X < 42$$

Third demo!

# The first example with some twists

- A program unit:

```
classify(L) when length(L) < 4 -> tiny;
classify(L) ->
  case lists:foldl(fun erlang:'+'/2, 0, L) of
    S when S < 0 -> negative;
    S when S < 4711 -> small;
    S when S > 4711 -> big
  end.
```

A function that classifies a list of numbers

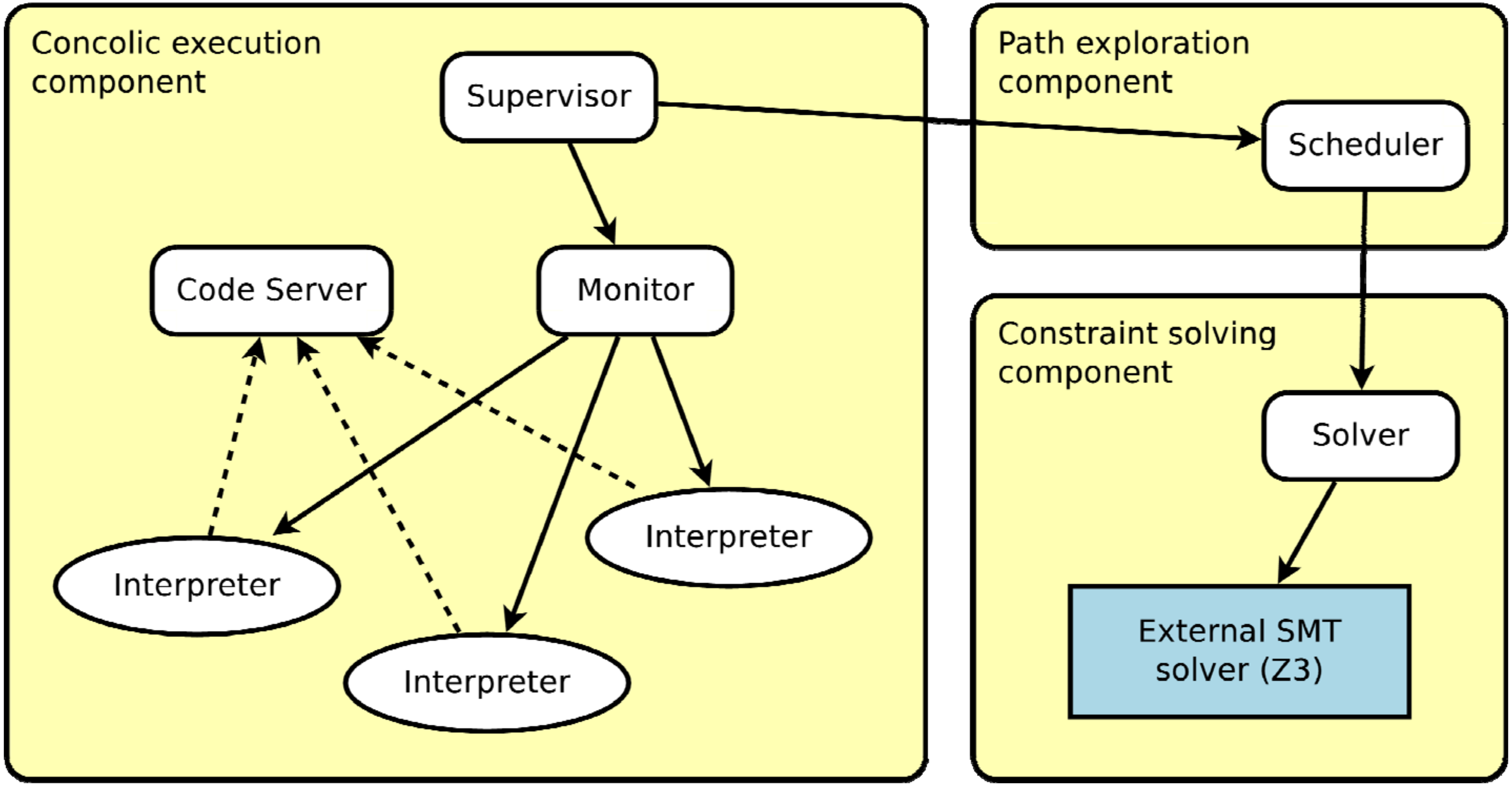
# One more twist

- A program unit:

```
classify(_, L) when length(L) < 4 -> tiny;
classify(F, L) ->
  case lists:foldl(F, 0, L) of
    S when S < 0 -> negative;
    S when S < 4711 -> small;
    S when S > 4711 -> big
  end.
```

A function that classifies a list of numbers

# CutEr: Concolic Unit Testing for Erlang



# SMT solving with Z3

- Define the most general type, i.e. **`term()`**

```
Term, TList, IList = Datatypes('Term, TList, IList')

Term.declare('int', ('ival', IntSort()))
Term.declare('real', ('rval', RealSort()))
Term.declare('atm', ('aval', IList))
Term.declare('lst', ('lval', TList))
Term.declare('tpl', ('tval', TList))

TList.declare('nil')
TList.declare('cons', ('hd', Term), ('tl', TList))

IList.declare('anil')
IList.declare('acons', ('ahd', IntSort()), ('atl', IList))
```

# Encoding values in Z3

- For example, for the terms:
  - **42**
  - **[17,42]**
  - **{42,ok}**

```
t1 = Term.int(42)
t2 = Term.lst(TList.cons(Term.int(17),
       TList.cons(Term.int(42), TList.nil)))
t3 = Term.tpl(TList.cons(Term.int(42),
       TList.cons(Term.atm(
         IList.acons(111, IList.acons(107, IList.anil))
       ), TList.nil)))
```

# Encoding axioms in Z3

- For example, the path constraint:

$$\neg\ (\texttt{L = [H|T]}) \wedge \neg\ (\texttt{L = []})$$

```
Not(And(Term.is_lst(L), TList.is_cons(Term.lval(L))))
Not(And(Term.is_lst(L), TList.is_nil(Term.lval(L))))
```

# CutEr

- Available on GitHub:

  **`https//github.com/aggelgian/cuter`**

- Requires Erlang/OTP 17.x or 18.x

Current known limitations:

- Does not support maps (yet!)
- Support for recursive types is still incomplete

# A bigger unit to test

- A post in the erlang-bugs mailing list:

**`http//erlang.org/pipermail/erlang-bugs/2015-May/004944.html`**

- Module **`otp_internal`** from Erlang/OTP 18.0-rc1

**`https//github.com/erlang/otp/blob/OTP-18.0-rc1/lib/stdlib/src/otp_internal.erl`**

[erlang-bugs] Crash on compile with deprecated functions (18-rc1) - Google Chrome

[erlang-bugs] Crash o ×

erlang.org/pipermail/erlang-bugs/2015-May/004944.html

# [erlang-bugs] Crash on compile with deprecated functions (18-rc1)

**Loïc Hoguin** <essen@ninenines.eu>
*Sat May 2 17:11:30 CEST 2015*

- Previous message: [erlang-bugs] FreeBSD FPE issue on ERTS_FP_CHECK_INIT Re: ERTS_FP_CHECK_INIT error of HiPE in 18.0-rc1 running on FreeBSD 10.1-STABLE
- Next message: [erlang-bugs] Crash on compile with deprecated functions (18-rc1)
- **Messages sorted by:** [ date ] [ thread ] [ subject ] [ author ]

---

```
Hello,

Some of my applications don't compile anymore because they have the
ssl:negotiated_next_protocol instead of negotiated_protocol.

Problem is there is a crash instead of a nice error:

src/gun.erl: internal error in lint_module;
crash reason: {case_clause,{deprecated,{ssl,negotiated_protocol}}}

   in function  otp_internal:obsolete/3 (otp_internal.erl, line 33)
   in call from erl_lint:deprecated_function/5 (erl_lint.erl, line 3551)
   in call from erl_lint:check_remote_function/5 (erl_lint.erl, line 3527)
   in call from erl_lint:expr/3 (erl_lint.erl, line 2166)
   in call from erl_lint:expr/3 (erl_lint.erl, line 2111)
   in call from erl_lint:expr/3 (erl_lint.erl, line 2250)
   in call from erl_lint:exprs/3 (erl_lint.erl, line 2044)
   in call from erl_lint:icrt_clause/3 (erl_lint.erl, line 3029)
```

Demo!

# Concluding remarks

- This presentation:
  - Concolic testing for the “functional” subset of Erlang
  - CutEr: a tool that implements this approach

**`https//github.com/aggelgian/cuter`**

- Future Work
  - Better search strategies
  - Experiment with more SMT solvers
  - Handle concurrency

Thanks!