Testing Vs. Verification

- Testing
  - Concrete inputs
  - Concrete assertions over outputs / state
  - No control over asynchrony, scheduling
  - Low investment, low return
- Verification
  - Formally specify program, behavior
  - Systematically test inputs, schedules
  - High investment, high return

Goal:

High software quality assurance with an investment similar to unit tests

— bloom

- A declarative language for distributed programming
- Computation is transformation
- State is uniformly represented as sets

Local Updates
System Events
Bloom Rules
atomic, local, deterministic

Network
Now
Next

multicast <- (message * member) do |mes, mem| [mem.address, mes.id, mes.payload] end

<Collection> <Accumulator> <From List> <Expression>

BloomUnit: declarative testing for distributed programs

Peter Alvaro, Andrew Hutchinson, Neil Conway, William R. Marczak, Joseph M. Hellerstein

Input generation

- Exclusion constraints: what records cannot appear in an input instance:
  all p1, p2 : pipe_in | (p1.src = p2.src and p1.ident = p2.ident) => p1.payload = p2.payload

(blah)

- Inclusion constraints: what records MUST appear in an input instance:
  some p1, p2 : pipe_in | p1 != p2 => (p1.src = p2.src and p1.dst = p2.dst)

(there are at least two messages between two endpoints)

- Input generation ~ model-finding problem
- Encode constraints in the Alloy language
  - FO + TC
- Alloy finds k nonisomorphic models
- Convert models => input instances

Exploring distributed executions

- All distributed executions are nondeterministic
- Each concrete input => set of executions
- Message timings / orderings may differ
- Too large a space to search exhaustively!

Intuition:

- Explore k random message orderings
  - In practice, k=2 catches many bugs!

Optimization:

- CALM Theorem:
  - Consistency as logical monotonicity
  - Monotonic => race-free
- Only explore messages orderings when downstream logic is nonmonotonic
- Search only “interesting” orderings

Declarative specifications

An abstract delivery protocol

module DeliveryProtocol
state do
  interface input, pipe_in,
  |[dst, src, idend] => |[payload]
  interface output, pipe_sent,
  |[dst, src, idend] => |[payload]
  |[dst, src, idend] => |[payload]
end

A FIFO delivery specification

module FIFOspec include DeliveryProtocol
bloom do
  fail <= (pipe_out_log * pipe_out_log).pairs do |p1, p2| if p1.src = p2.src and p1.dst = p2.dst and p1.ident < p2.ident and p1.time >= p2.time
    "#(p1.inspect) < #(p2.inspect)"
  end
  end
end

"delivery order (timestamps) never deviates from sender order (encoded into ident),"

- Specifications: queries over execution traces
- Specifications encode invariants
  - Queries capture incorrect behaviors
  - Specifications > Assertions
- Easy to write, but cover all inputs