BEADL: A New Real-Time Language for Behavioral Experiments

Stephen A. Edwards

Columbia University

Octopi Workshop
Chalmers University of Technology
Gothenburg, Sweden
December 2018
Adam Kepecs, Cold Spring Harbor Laboratory
[Lak et al., Neuron 84(1), 2014]
Bpod: An Open Hardware Platform for Behavioral Monitoring and Control

Sanworks.io, spun out of Kepecs’ lab.
Teensy 3.6: ARM Cortex M4, 180 MHz
sma = NewStateMatrix();
sma = AddState(sma, 'Name', 'ITI',
    'Timer', S.ITI,
    'StateChangeConditions', {'Tup', 'PreState'},
    'OutputActions', {});
%Pre task states
sma = AddState(sma, 'Name', 'PreState',
    'Timer', S.GUI.PreCue,
    'StateChangeConditions', {'Tup', 'CueDelivery'},
    'OutputActions', {'BNCState', 1});
%Cue
sma = AddState(sma, 'Name', 'CueDelivery',
    'Timer', S.GUI.CueDuration,
    'StateChangeConditions', {'Tup', 'Delay'},
    'OutputActions', {'SoftCode', S.Cue});
%Delay
sma = AddState(sma, 'Name', 'Delay',
    'Timer', S.Delay,
    'StateChangeConditions', {'Tup', 'ExtraCueDelivery'},
    'OutputActions', {});
%Extra Cue for L3-SecondaryCue
sma = AddState(sma, 'Name', 'ExtraCueDelivery',
    'Timer', S.ExtraCueDuration,
    'StateChangeConditions', {'Tup', 'ExtraDelay'},
    'OutputActions', {'SoftCode', S.ExtraCue});
%Extra Delay for L3-SecondaryCue
sma = AddState(sma, 'Name', 'ExtraDelay',
    'Timer', S.ExtraDelay,
    'StateChangeConditions', {'Tup', 'Outcome'},
    'OutputActions', {});
%Reward
sma = AddState(sma, 'Name', 'Outcome',
    'Timer', S.Outcome,
    'StateChangeConditions', {'Tup', 'PostOutcome'},
    'OutputActions', {'ValveState', S.Valve});
%Post task states
sma = AddState(sma, 'Name', 'PostOutcome',
    'Timer', S.GUI.PostOutcome,
    'StateChangeConditions', {'Tup', 'exit'},
    'OutputActions', {});
SendStateMatrix(sma);

---

Describe as FSM
Build FSM with Matlab API
Download FSM to firmware
Firmware: FSM interpreter w/100 µs heartbeat
BEADL: The Idea

Controller  Stimuli  Subject  Responses  Results

outputs
valve dispense  # Channel w/ 1 event
led on off  # Two possible events

inputs
gate enter exit

task simple_example:
"Subject Attraction"  # State label
valve dispense  # Generate event
await
10 s:  # Timeout
"Failure"
goto "Subject Attraction"
gate enter:  # Event arrived

"Light Stimulus"
led on  # Generate event
await 100 ms  # Simple delay
led off
Deterministic formal semantics

Explicit model-time delays only; platform-independent timing above some minimum delay (synchronous logic)

“Bare metal” microcontroller implementations: hardware counter/timer drives timing, timer interrupts for scheduling

Schedulability/static timing analysis done at compile time
BEADL: Possible Single-Threaded Implementation

void interrupt1() {
    now = get_platform_time();
    switch (state) {
    case S1:
        Attract: report("Attract");
            valve_dispense();
            state = S2, schedule(now + SEC(10)); return;
    case S2:
        switch (get_interrupting_event()) {
        case TIMEOUT:
            Fail: report("Fail");
                goto Attract;
        case GATE_ENTER: break;
        default: return; }
    Stimulus: report("Stimulus");
            led_on();
            state = S3, schedule(now + MS(100)); return;
    case S3:
            led_off();
            state = STOPPED; return;
    case STOPPED:
            return;
    }
}
BEADL: Parallel Composition

*Language Design is Library Design* — Stroustrup

A desired BEADL library: input debounce

Nervous rats often jitter before making a decision; want a library that discards “on” events shorter than $x$ ms

⇒ Parallel composition?

Feedback loops?

Simultaneous events?

Contradictions?
Ptolemy
Edward Lee et al., UC Berkeley, 1980s to present

Fig 2.1, System Design, Modeling, and Simulation Using Ptolemy II, 2014

Originally for simulating synchronous dataflow; this remains its primary strength
Heterogeneous Modeling in Ptolemy

Director = simulation controller; imposes operational semantics
Ptolemy’s Discrete Event Domain

Fig 7.4, System Design, Modeling, and Simulation Using Ptolemy II, 2014
Very subtle bug: PoissonClock generates its first event at time 0; the Previous block emits no event in response, AddSubtract only receives the single event tagged “0” and outputs a spurious “0.”
Ptolemy’s Discrete Event Domain

![Diagram of a system design model using Ptolemy II.](image)

Solution: Add a sampler to drop the first event

![Diagram of a system design model with a sampler added.](image)

Fig 7.4, System Design, Modeling, and Simulation Using Ptolemy II, 2014

Fig 7.5, System Design, Modeling, and Simulation Using Ptolemy II, 2014
Achieving Determinacy

Channels convey events

Event : Value \times Timestamp

Timestamp : Model time \times Microstep \times Topological Level
: \mathbb{R} \times \mathbb{N}_0 \times \mathbb{N}_0

Events on a channel may have identical timestamps

Topological levels are straightforward for this example:

0 1 2 3 4 5

Fig 7.5, System Design, Modeling, and Simulation Using Ptolemy II, 2014
Scheduling Algorithm: Dealing With Simultaneous Events

Maintain an event queue ordered by timestamp

Timestamps ordered by model time, microstep, then topological level

1. Select event $e$ with lowest timestamp $t$ in event queue. Let $a$ be the actor to which $e$ is to be applied
2. Set model time to $t$
3. Let $E$ be all events with timestamp $t$ for actor $a$ in the queue
4. Remove events $E$ from the queue
5. Fire actor $a$ on events $E$ (may generate more events)
6. Repeat
Feedback Loops

Every feedback loop must be broken with a time delay actor

A delay of “0” is actually a delay of a single microstep
Microsteps can lead to madness

...or Zeno-like behavior
Last resort: Priorities (for actors with side-effects)

Fig 7.9, System Design, Modeling, and Simulation Using Ptolemy II, 2014
Real-time scheduler maintains event queue

If next queued event is “too far” in the future, delay with a timer interrupt

Interrupts on inputs generate new events
Clever contribution of PTIDES is a distributed implementation strategy that answers “When I can advance my clock, i.e., when are there no older events?”

I don’t like its semantic model, especially its need to explicitly break feedback loops with microstep delays.

Microsteps: “A poor man’s fixed point evaluator”

The main culprit: allowing multiple events on a channel
Simultaneous Events

What should we do with simultaneous events?

We could simply legislate them away at the input, but they are easy to generate internally.

What should this do?
Simultaneous Events

What should we do with simultaneous events?

We could simply legislate them away at the input, but they are easy to generate internally.

Seems reasonable: output is double the input
Simultaneous Events

What should we do with simultaneous events?

We could simply legislate them away at the input, but they are easy to generate internally.

Should this be allowed? What should its output be?
node Watchdog (set, reset, u_tps: bool; delay: int) returns (alarm: bool);
var is_set : bool;
    remain : int;
let
    alarm = is_set and (remain = 0) and pre(remain) > 0;
    is_set = false -> if set then true
        else if reset then false
            else pre(is_set);
    remain = 0 -> if set then delay
        else if u_tps and pre(remain) > 0
            then pre(remain) – 1

tel

Declarative dataflow style; expressing control is awkward
Every loop must have a unit delay ("pre") No microsteps
Implicit clock not tied to wallclock time

[Halbwachs, Caspi, et al.]
The Esterel Synchronous Programming Language

module ABRO:
input A, B, R;
output O;

loop
  [ await A || await B ];
  emit O
  each R
end module

Imperative style with sequencing, concurrency, conditionals, and exceptions

More subtle causality constraints; “constructive” causality requires a per-state analysis

[Berry et al.]
BEADL: A Work in Progress

- **Semantics**
  Event-driven with explicit model time advances
  Synchronous: Esterel-like with Lustre-style causality?
  No reactions to absent events (timeouts only)

- **Implementation**
  PTIDES-like: Interrupt driven
  Events: timeouts, input arrivals
  Model time “matched” to platform time

- **Schedulability**
  Run-time deadline checking
  Compile-time WCET analysis?