Software Quality Engineering
07. Automata-based Testing

Achiya Elyasaf

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

הנדסת מערכות תוכנה ומידע
Software and Information Systems Engineering

Software Testing and QA Theory and Practice, Naik andTripathy

Today’s Agenda

- Why CTD is not enough?
- FSM Representation
- Coverage Criteria
- Introduction to Behavioral Programming
- Story-based testing (Provengo)

Where CTD fails?

Vars	Medical Devices	Browser	Server	NASA GSFC	Network Security
1	66	29	42	68	20
2	97	76	70	93	65
3	99	95	89	98	90
4	100	97	96	100	98
5		99	96		100
6		100	100		

Table 1. Number of variables involved in triggering software faults
*** NASA GSFC is a distributed scientific database**

https://csrc.nist.rip/groups/SNS/acts/ftfi.html

Automata-Based Testing (ABT)

- Model system behavior as a finite state machine (FSM)
- Each walk on the FSM is a possible path.
- Now: `input_vector(ABT) = input_vector(CTD)` × `order of events`

FSM Representation

pco = point of control and observation

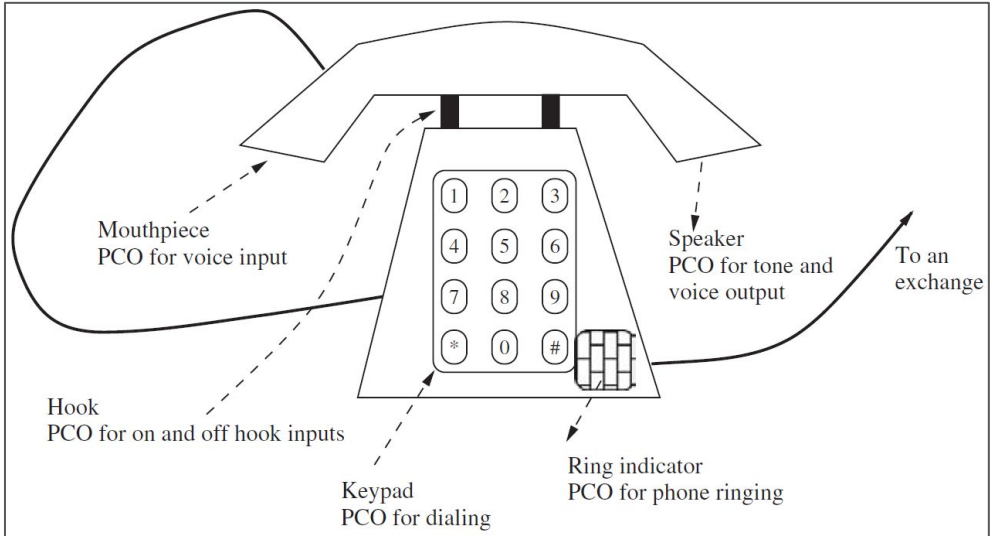


Figure 10.6 PCOs on a telephone.

TABLE 10.1 PCOs for Testing Telephone PBX

PCO	In/Out View of System	Description
Hook	In	The system receives off-hook and on-hook events.
Keypad	In	The caller dials a number and provides other control input.
Ring indicator	Out	The callee receives ring indication.
Speaker	Out	The caller receives tones (dial, fast busy, slow busy, etc.) and voice.
Mouthpiece	In	The caller produces voice input.

FSM Representation

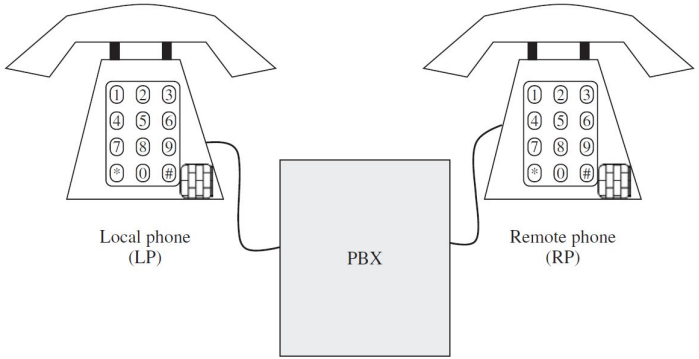


Figure 10.7 FSM model of a PBX.

private branch exchange (PBX) = a telephone system

A FSM M is defined as a tuple as follows:

$M = \langle S, I, O, s0, \delta, \lambda \rangle$, where

S is a set of states,

I is a set of inputs,

O is a set of outputs,

$s0$ is the initial state,

$\delta : S \times I \rightarrow S$ is a next-state function, and

$\lambda : S \times I \rightarrow O$ is an output function.

FSM Representation

TABLE 10.2 Set of States in FSM of Figure 10.8

Abbreviation	Expanded Form	Meaning
OH	On hook	A phone is on hook.
AD	Add digit	The user is dialing a number.
SB	Slow busy	The system has produced a slow busy tone.
FB	Slow busy	The system has produced a fast busy tone.
RNG	Ring	The remote phone is ringing.
TK	Talk	A connection is established.
LON	Local on hook	The local phone is on hook.
RON	Remote on hook	The remote phone is on hook.
IAF	Idle after Fast busy	The local phone is idle after a fast busy.

TABLE 10.3 Input and Output Sets in FSM of Figure 10.8

Input	Output
OFH: Off hook	DT: Dial tone
ONH: On hook	RING: Phone ringing
#1: Valid phone number	RT: Ring tone
#2: Invalid phone number	SBT: Slow busy tone
NOI: No input	FBT: Fast busy tone
	IT: Idle tone
	—: Don't care

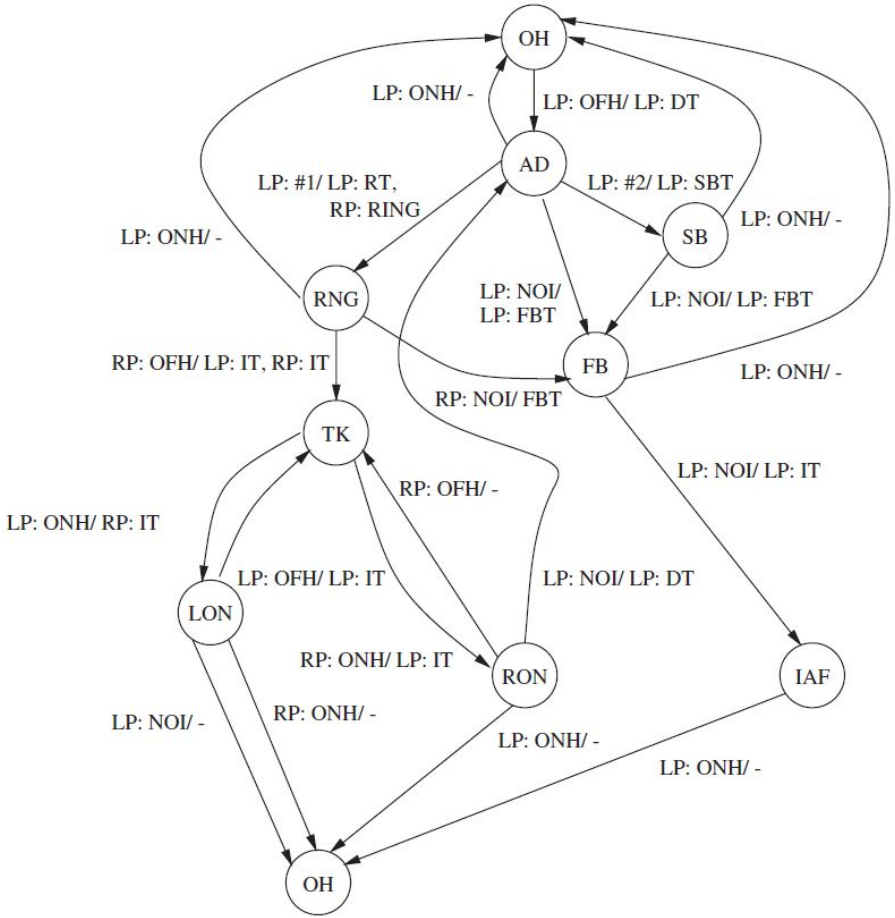


Figure 10.8 FSM model of PBX.

Test Generation From FSM

Is it enough to cover all edges?

Is it redundant to test both

`1-2-1-3-4` and `1-3-4-1-2` ?

Do we know for sure that after `1-2`, the system is indeed in `OH`?

We model what we *believe* the system should behave, how we *perceive* the system requirements.

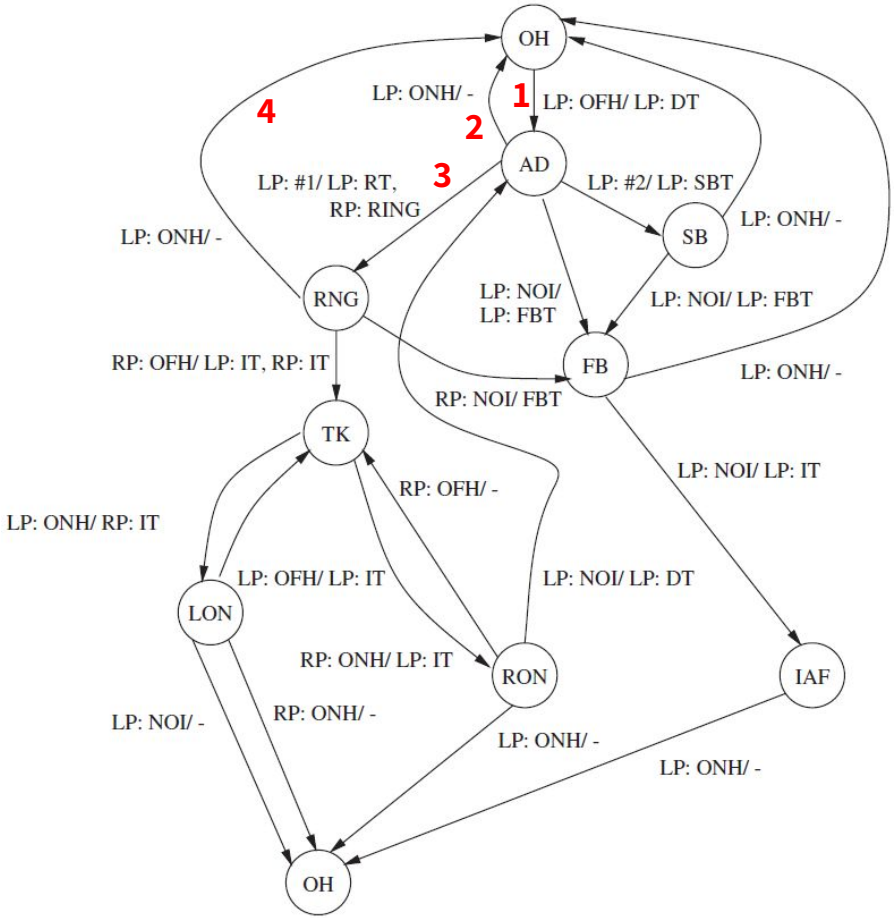


Figure 10.8 FSM model of PBX.

Coverage Criteria

- State Coverage
- Branch Coverage
- All paths (i.e., with/without cycles, with max length)
- 2-way
- Combinatorial Sequence Test Design (CSTD)

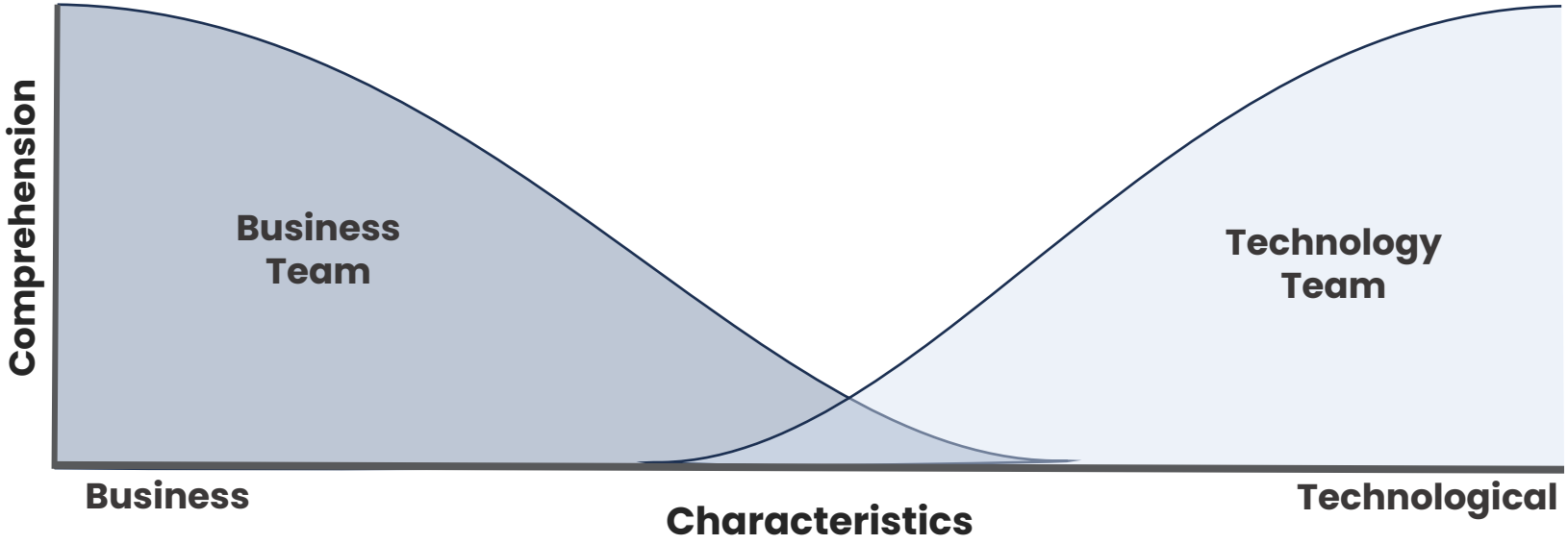
A. Elyasaf, E. Farchi, O. Margalit, G. Weiss and Y. Weiss, "Generalized Coverage Criteria for Combinatorial Sequence Testing," in IEEE Transactions on Software Engineering, vol. 49, no. 8, pp. 4023-4034, Aug. 2023, doi: 10.1109/TSE.2023.3279570.

Problems With ABT?

- Requires time and expertise.
- Systems are too complex for visual models. Complicate to model their intricate logic.
- Cannot test everything in-sprint. How to target test generation to fit testing in-sprint?
- Visual models are good for simple UIs, the backbone may be APIs and back-end. These are difficult to model.
- Add a maintenance overhead. Impossible to update models on every change to the system. Model will become irrelevant fast.
- ABT is not relevant in agile development.

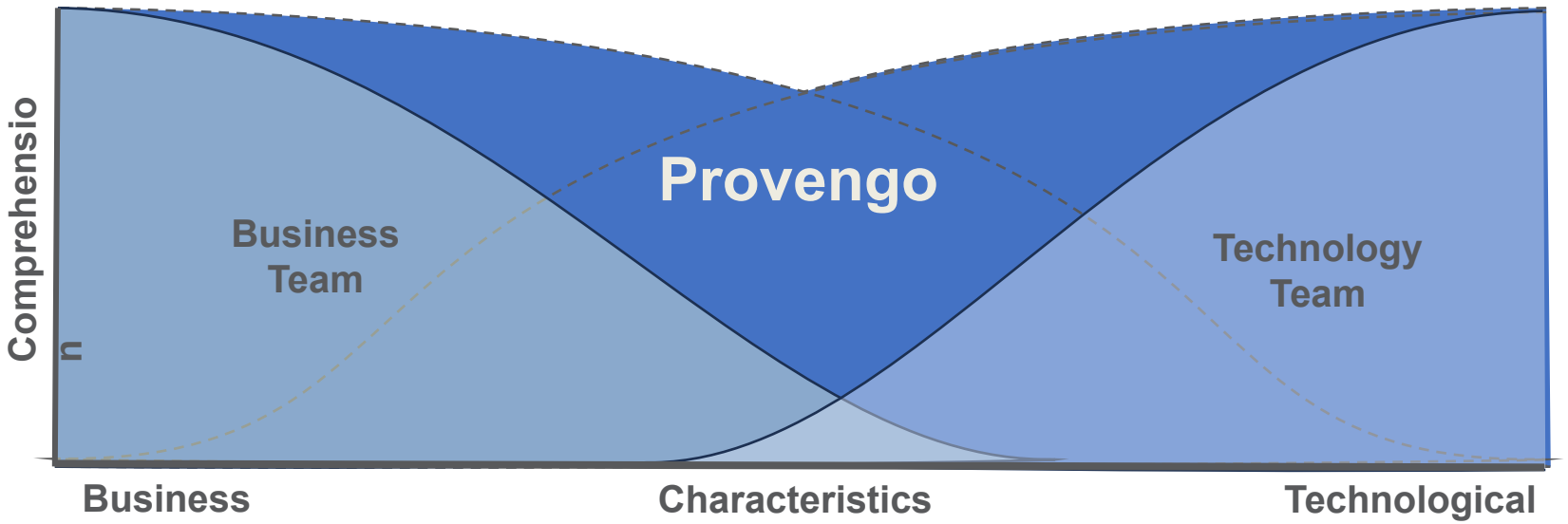
Introducing Provengo

THE CHALLENGE

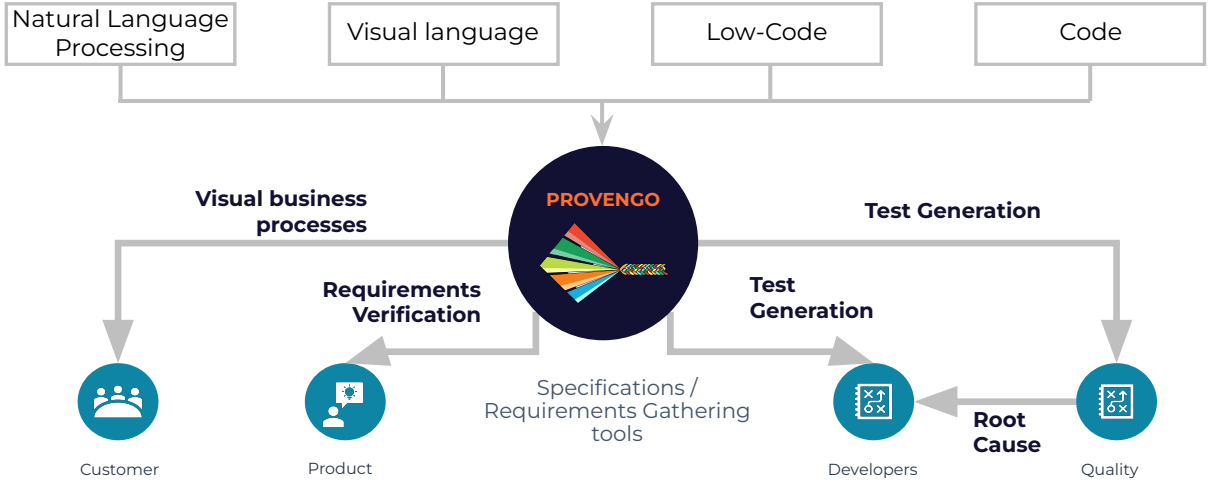


Today, all alignment is handled manually, driving up costs, delivery dates and quality issues

THE SOLUTION

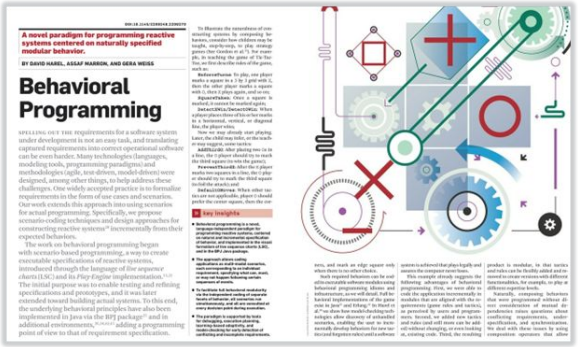


Introducing Provengo



Behavioral Programming

An approach for software development that enables incremental development in a natural way.



A novel paradigm for programming reactive systems centered on naturally specified modular behavior.

BY DAVID HAREL, ASSAF MARRON, AND GERA WEISS

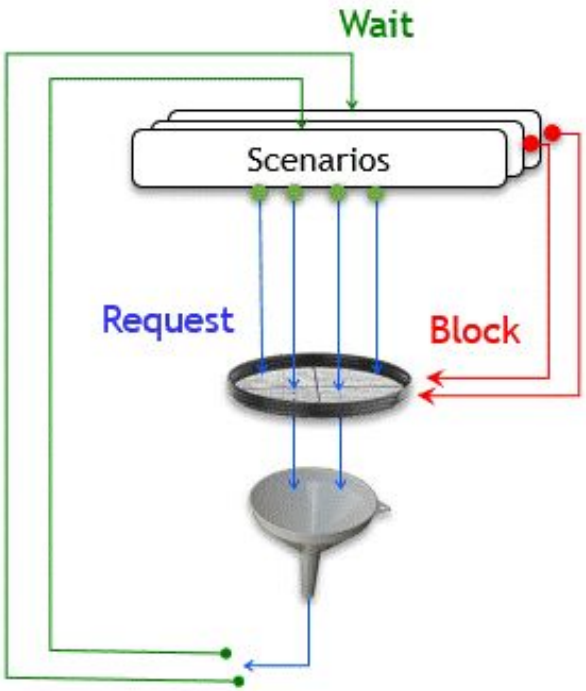
Behavioral Programming

David Harel, Assaf Marron, Gera Weiss
Communications of the ACM, July 2012, Vol. 55 No. 7, Pages 90-100
10.1145/2209249.2209270

Key Features:

- A behavioral application consists of behavior threads, each representing an independent scenario the system should or shouldn't follow.
- These independent behavior threads are interwoven at run-time, yielding integrated system behavior.
- For example, in a game-playing application, each game rule and each playing strategy would be programmed separately and independently with little or no awareness of other modules.

Behavioral Programming

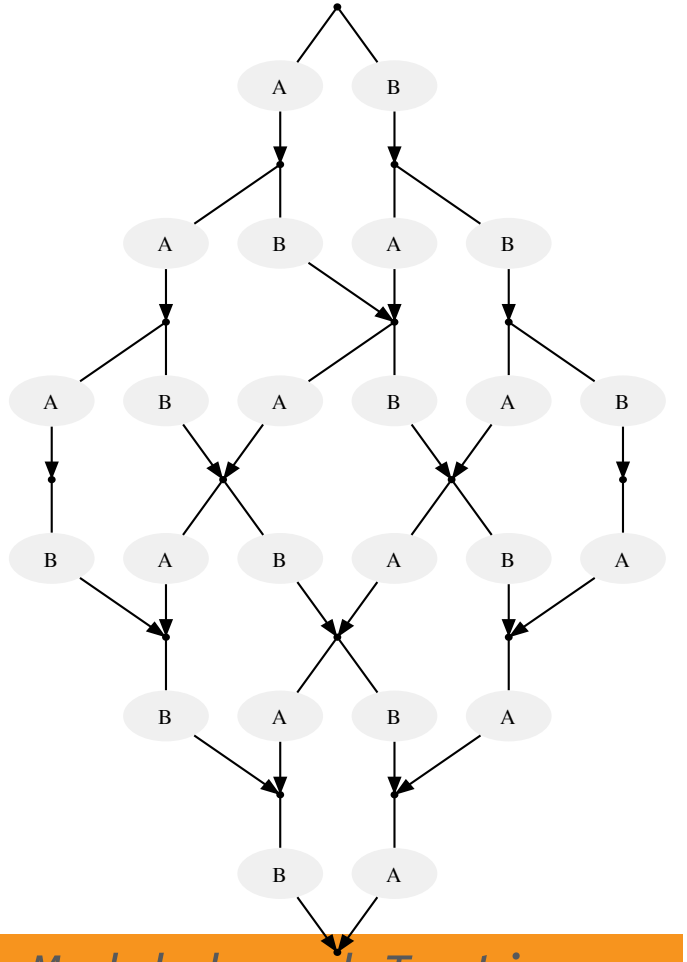


```
bthread("name1", function () {
    sync({request: Event("A")})
    sync({request: Event("A")})
    sync({request: Event("A")})
})

bthread("name2", function () {
    sync({request: Event("B")})
    sync({request: Event("B")})
    sync({request: Event("B")})
})
```

We want to have three “A” events in our run

We want to have three “B” events in our run

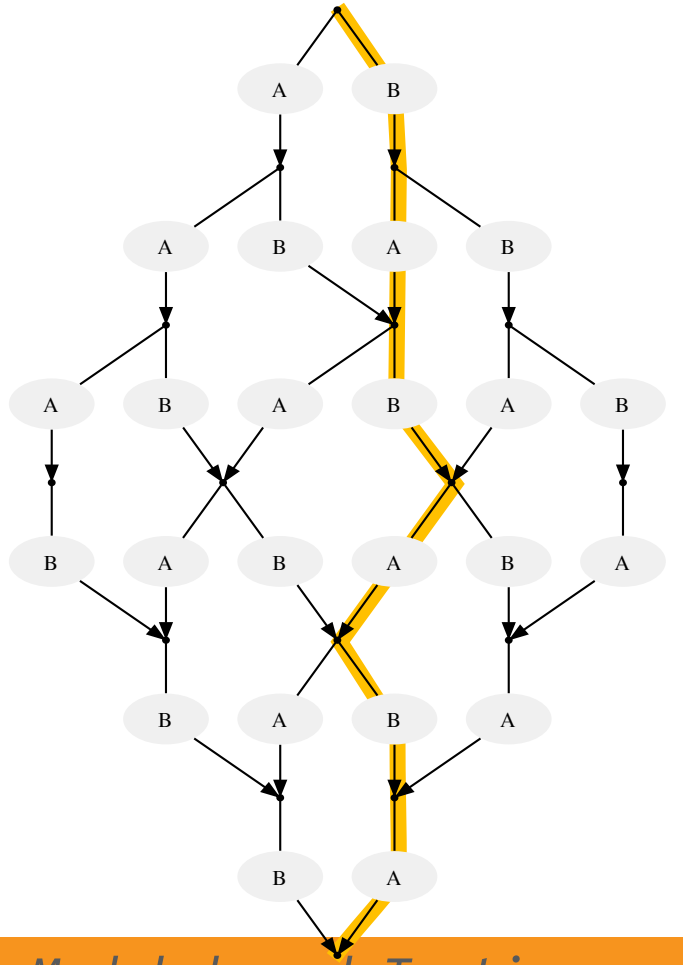
```
bthread("name1", function () {
    sync({request: Event("A")})
    sync({request: Event("A")})
    sync({request: Event("A")})
})

bthread("name2", function () {
    sync({request: Event("B")})
    sync({request: Event("B")})
    sync({request: Event("B")})
})
```



SQ - Model-based Testing

```
bthread("name1", function () {
    sync({request: Event("A")})
    sync({request: Event("A")})
    sync({request: Event("A")})
})

bthread("name2", function () {
    sync({request: Event("B")})
    sync({request: Event("B")})
    sync({request: Event("B")})
})
```

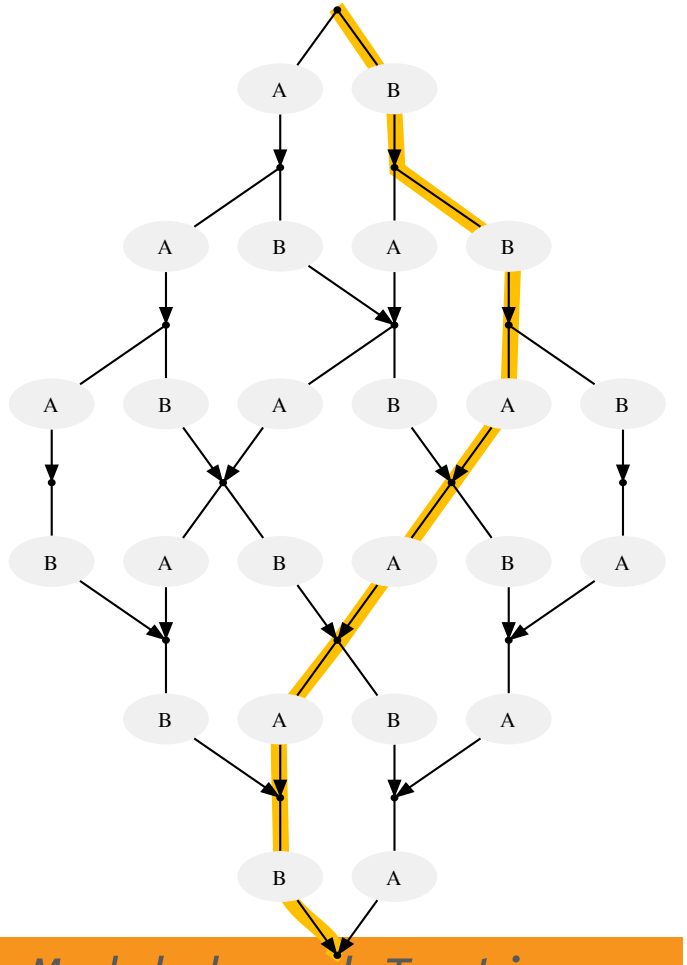
```
$ provengo run .
...
[RUN  ] INFO Preparing to run
[RUN>random] INFO B-program started
[RUN>random] INFO Selected: [B]
[RUN>random] INFO Selected: [A]
[RUN>random] INFO Selected: [B]
[RUN>random] INFO Selected: [A]
[RUN>random] INFO Selected: [B]
[RUN>random] INFO Selected: [A]
[RUN  ] INFO Test Result: SUCCESS
```



SQ - Model-based Testing

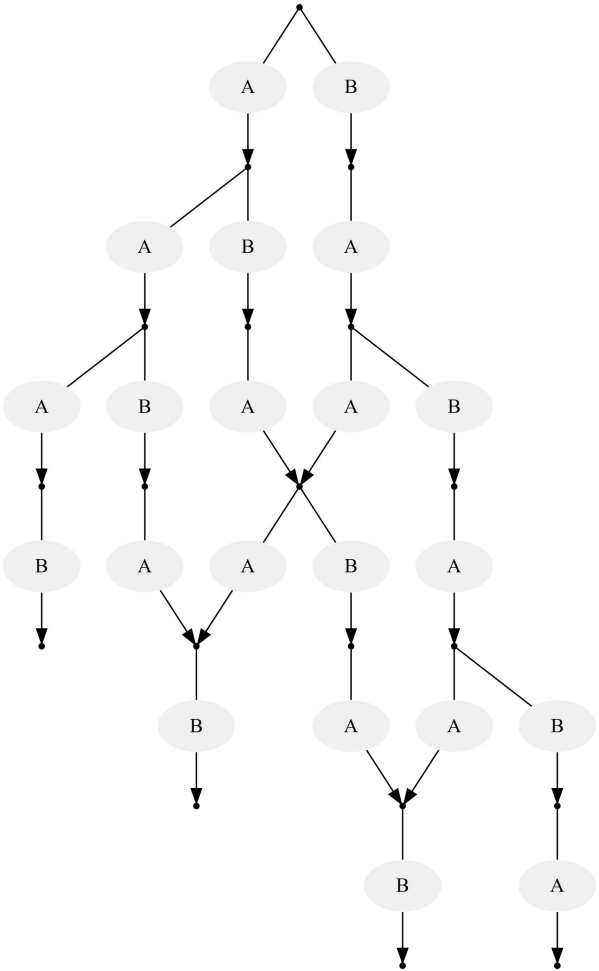
```
bthread("name1", function () {
    sync({request: Event("A")})
    sync({request: Event("A")})
    sync({request: Event("A")})
})

bthread("name2", function () {
    sync({request: Event("B")})
    sync({request: Event("B")})
    sync({request: Event("B")})
})
```

```
$ provengo run .
...
[RUN  ] INFO Preparing to run
[RUN>random] INFO B-program started
[RUN>random] INFO Selected: [B]
[RUN>random] INFO Selected: [B]
[RUN>random] INFO Selected: [A]
[RUN>random] INFO Selected: [A]
[RUN>random] INFO Selected: [A]
[RUN>random] INFO Selected: [B]
[RUN  ] INFO Test Result: SUCCESS
```



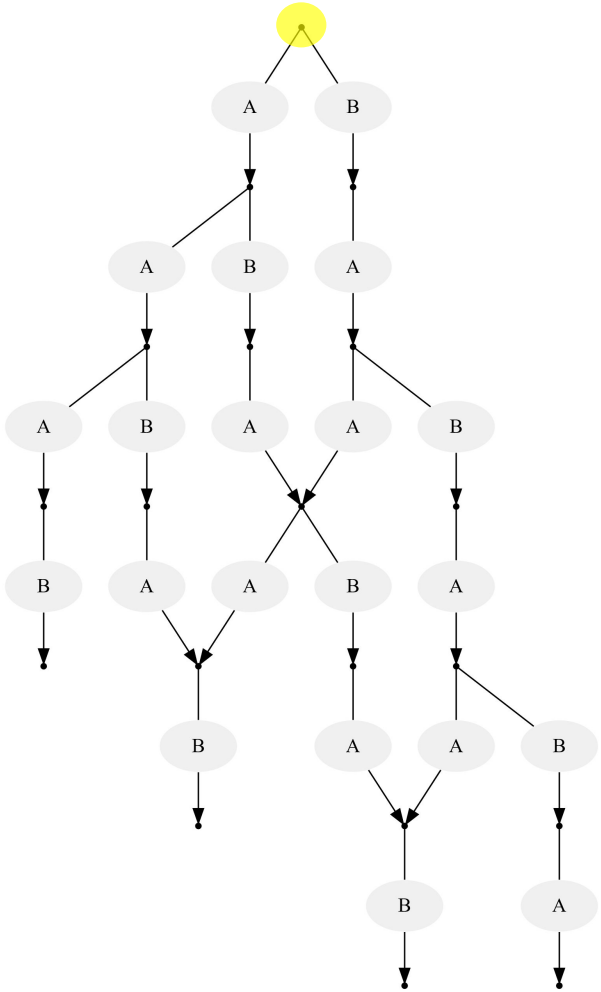
SQ - Model-based Testing

```
bthread("name1", function () {
    sync({ request: Event("A") })
    sync({ request: Event("A") })
    sync({ request: Event("A") })
})

bthread("name2", function () {
    sync({ request: Event("B") })
    sync({ request: Event("B") })
    sync({ request: Event("B") })
})


bthread("name3", function () {
    while(true) {
        sync({ waitFor: Event("B") })
        sync({ waitFor: Event("A"), block: Event("B") })
    }
})
```



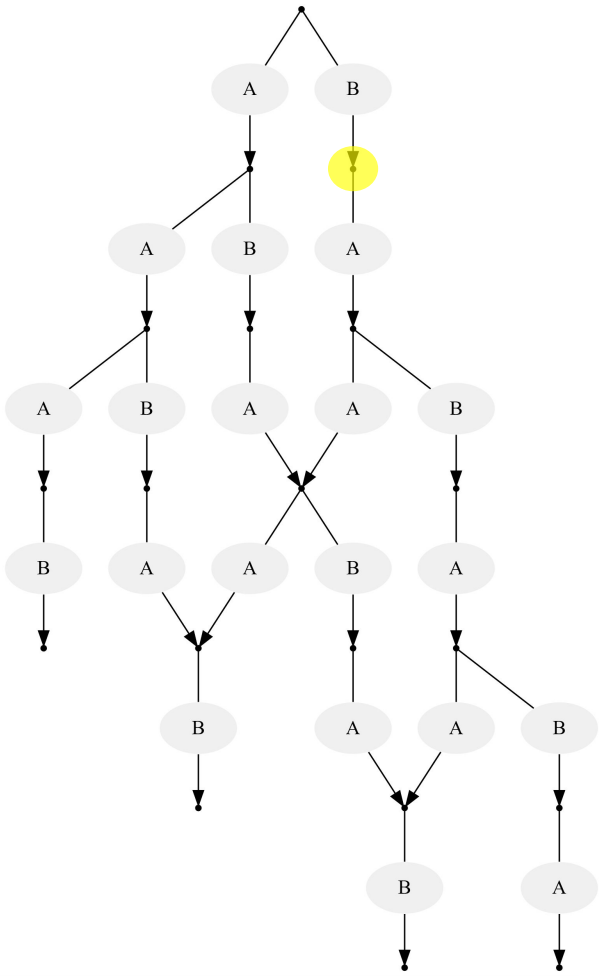
Don’t generate two “B” in a row

```
bthread("name1", function () {
    sync({ request: Event("A") })
    sync({ request: Event("A") })
    sync({ request: Event("A") })
})

bthread("name2", function () {
    sync({ request: Event("B") })
    sync({ request: Event("B") })
    sync({ request: Event("B") })
})


bthread("name3", function () {
    while(true) {
        sync({ waitFor: Event("B") })
        sync({ waitFor: Event("A"), block: Event("B") })
    }
})
```



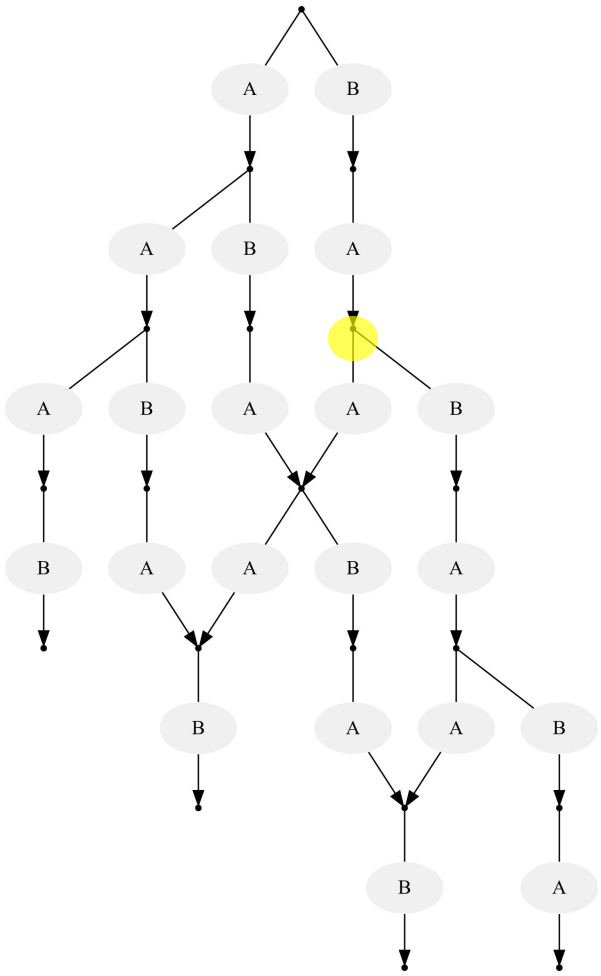
SQ - Mod

```
bthread("name1", function () {
    sync({ request: Event("A") })
    sync({ request: Event("A") })
    sync({ request: Event("A") })
})

bthread("name2", function () {
    sync({ request: Event("B") })
    sync({ request: Event("B") })
    sync({ request: Event("B") })
})


bthread("name3", function () {
    while(true) {
        sync({ waitFor: Event("B") })
        sync({ waitFor: Event("A"), block: Event("B") })
    }
})
```



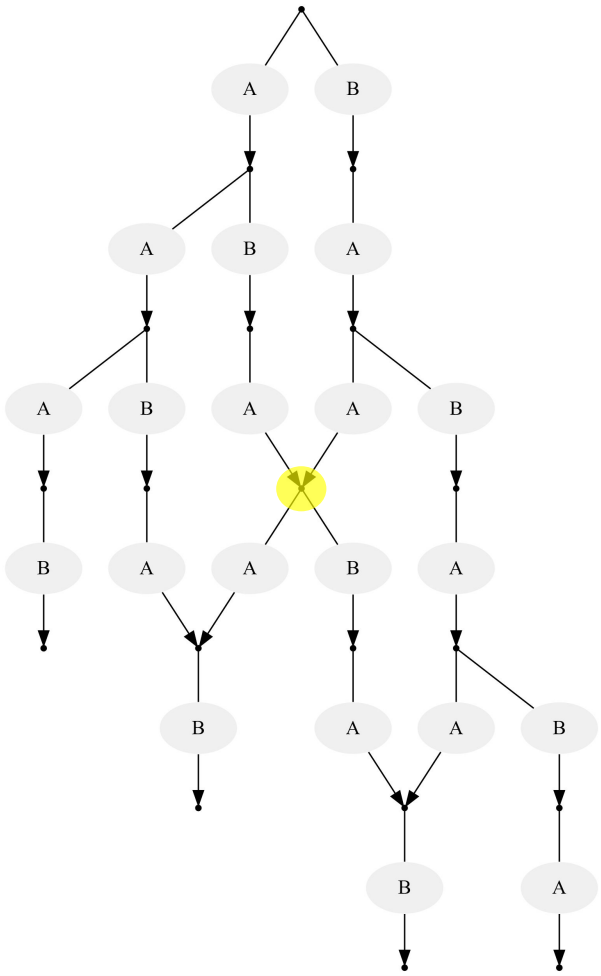
SQ - Mod

```
bthread("name1", function () {
    sync({ request: Event("A") })
    sync({ request: Event("A") })
    sync({ request: Event("A") })
})

bthread("name2", function () {
    sync({ request: Event("B") })
    sync({ request: Event("B") })
    sync({ request: Event("B") })
})


bthread("name3", function () {
    while(true) {
        sync({ waitFor: Event("B") })
        sync({ waitFor: Event("A"), block: Event("B") })
    }
})
```



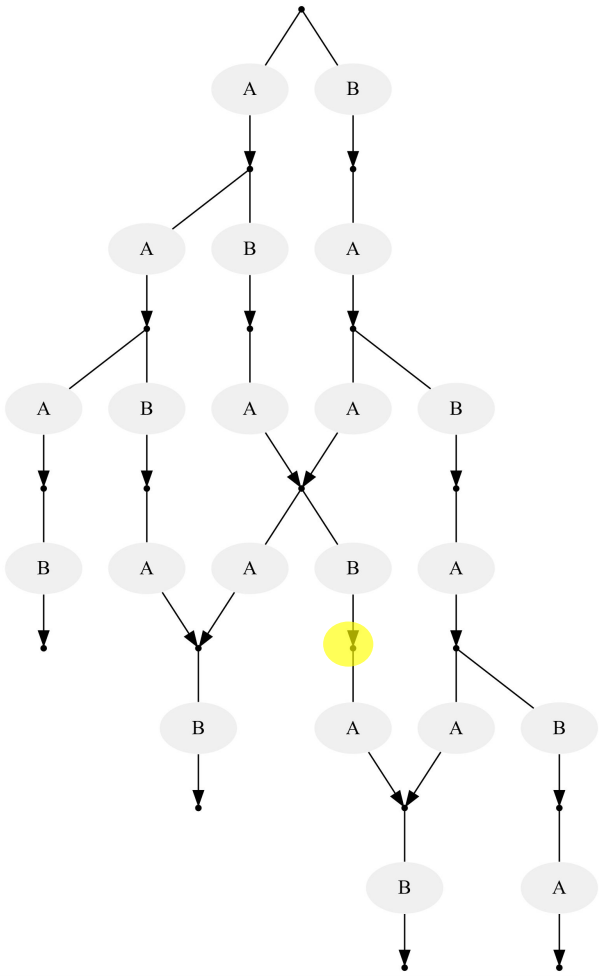
SQ - Mod

```
bthread("name1", function () {
    sync({ request: Event("A") })
    sync({ request: Event("A") })
    sync({ request: Event("A") })
})

bthread("name2", function () {
    sync({ request: Event("B") })
    sync({ request: Event("B") })
    sync({ request: Event("B") })
})


bthread("name3", function () {
    while(true) {
        sync({ waitFor: Event("B") })
        sync({ waitFor: Event("A"), block: Event("B") })
    }
})
```



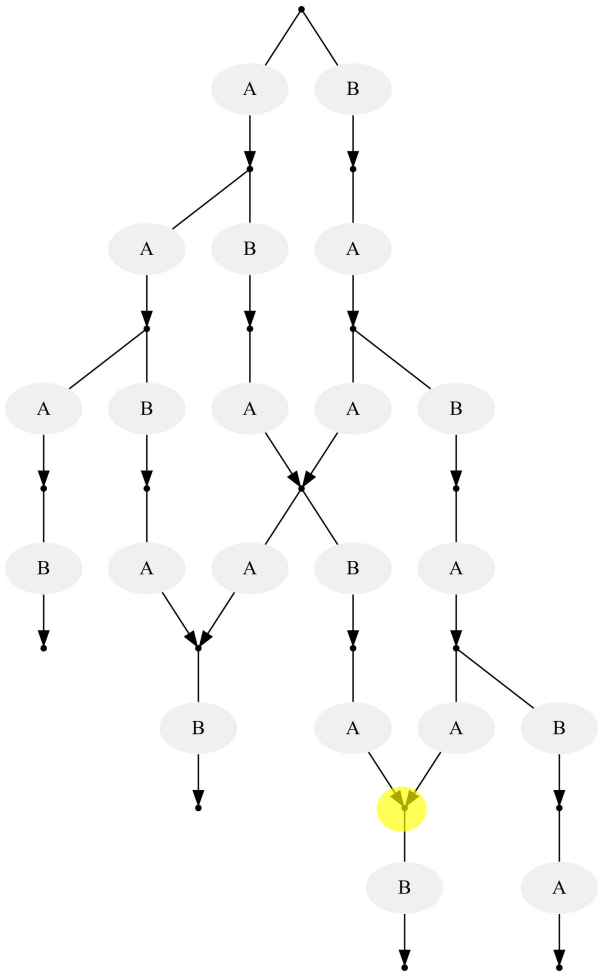
SQ - Mod

```
bthread("name1", function () {
    sync({ request: Event("A") })
    sync({ request: Event("A") })
    sync({ request: Event("A") })
})

bthread("name2", function () {
    sync({ request: Event("B") })
    sync({ request: Event("B") })
    sync({ request: Event("B") })
})


bthread("name3", function () {
    while(true) {
        sync({ waitFor: Event("B") })
        sync({ waitFor: Event("A"), block: Event("B") })
    }
})
```



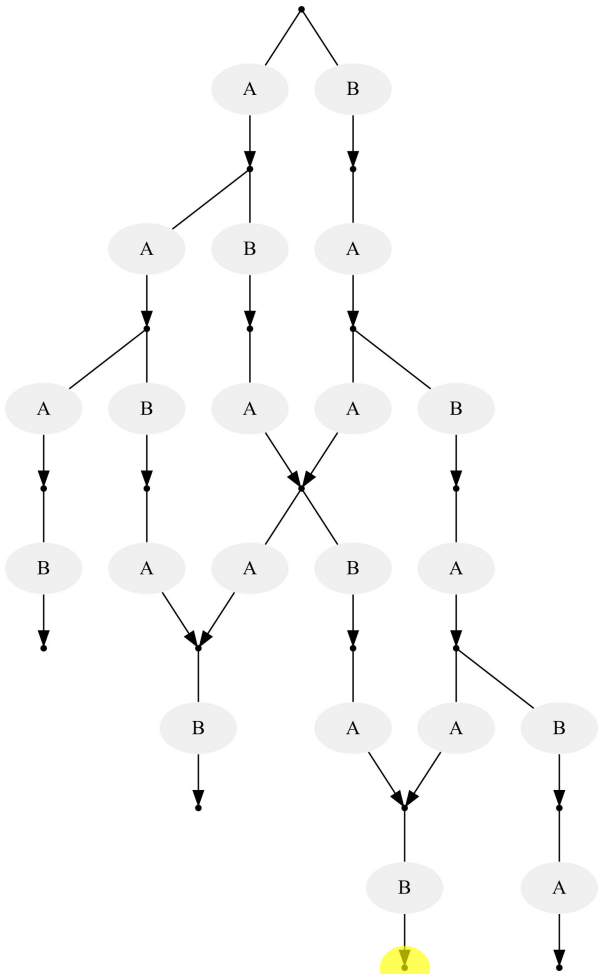
SQ - Mod

```
bthread("name1", function () {
    sync({ request: Event("A") })
    sync({ request: Event("A") })
    sync({ request: Event("A") })
})

bthread("name2", function () {
    sync({ request: Event("B") })
    sync({ request: Event("B") })
    sync({ request: Event("B") })
})


bthread("name3", function () {
    while(true) {
        sync({ waitFor: Event("B") })
        sync({ waitFor: Event("A"), block: Event("B") })
    }
})
```



SQ - Mod

```
bthread("name1", function () {
    sync({ request: Event("A") })
    sync({ request: Event("A") })
    sync({ request: Event("A") })
})

bthread("name2", function () {
    sync({ request: Event("B") })
    sync({ request: Event("B") })
    sync({ request: Event("B") })
})


bthread("name3", function () {
    while(true) {
        sync({ waitFor: Event("B") })
        sync({ waitFor: Event("A"), block: Event("B") })
    }
})
```



SQ - Mod

```
bthread("A sequence", function () {
    sync({ request: Event("A") })
    sync({ request: Event("A") })
    sync({ request: Event("A") })
})

bthread("B sequence", function () {
    sync({ request: Event("B") })
    sync({ request: Event("B") })
    sync({ request: Event("B") })
})

bthread("C sequence ", function () {
    sync({ request: Event("C") })
    sync({ request: Event("C") })
    sync({ request: Event("C") })
})
```

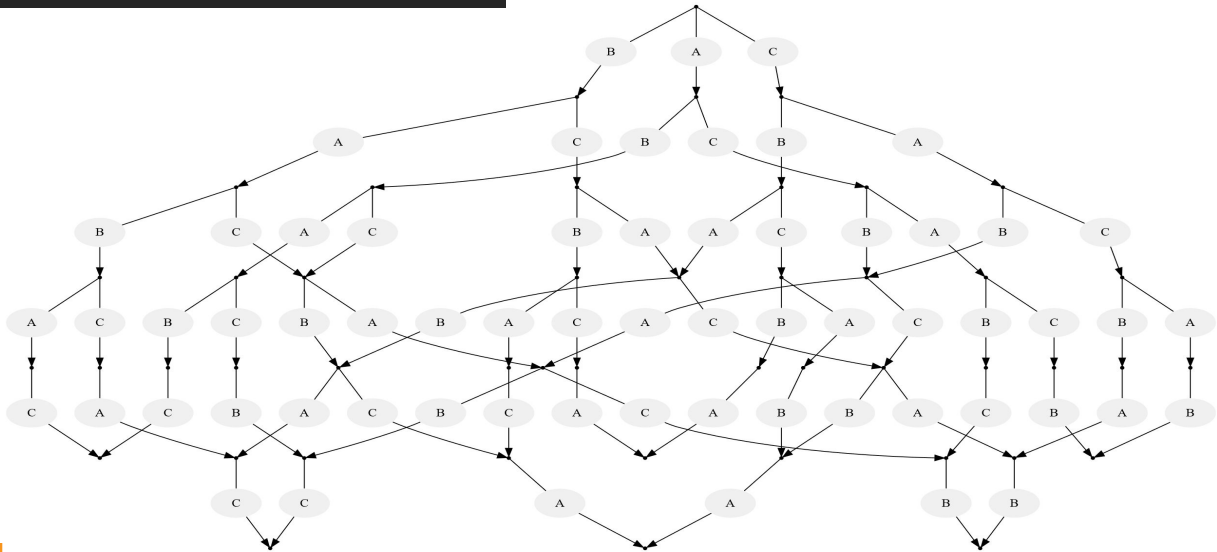
Same, same, but

```
const events = [Event("A"), Event("B"), Event("C")]
const LENGTH = 3

events.forEach(function (e) {
    bthread(e.name + " sequence", function () {
        for (var i = 0; i < LENGTH; i++) {
            sync({ request: e })
        }
    })
})
```

```
bthread("Don't allow two consecutive events of the same type", function () {
    last = Event(undefined)

    while (true) {
        last = sync({ block: last, waitFor: events })
    }
})
```

```
const events = [Event("A"), Event(“B"),
Event(“C")]
const LENGTH = 3

events.forEach(function (e) {
    bthread(e.name + " sequence", function () {
        for (var i = 0; i < LENGTH; i++) {
            sync({ request: e })
        }
    })
})
```

```
bthread(“No stuttering", function () {
    last = Event(undefined)

    while (true) {
        last = sync({ block: last, waitFor: events })
    }
})
```

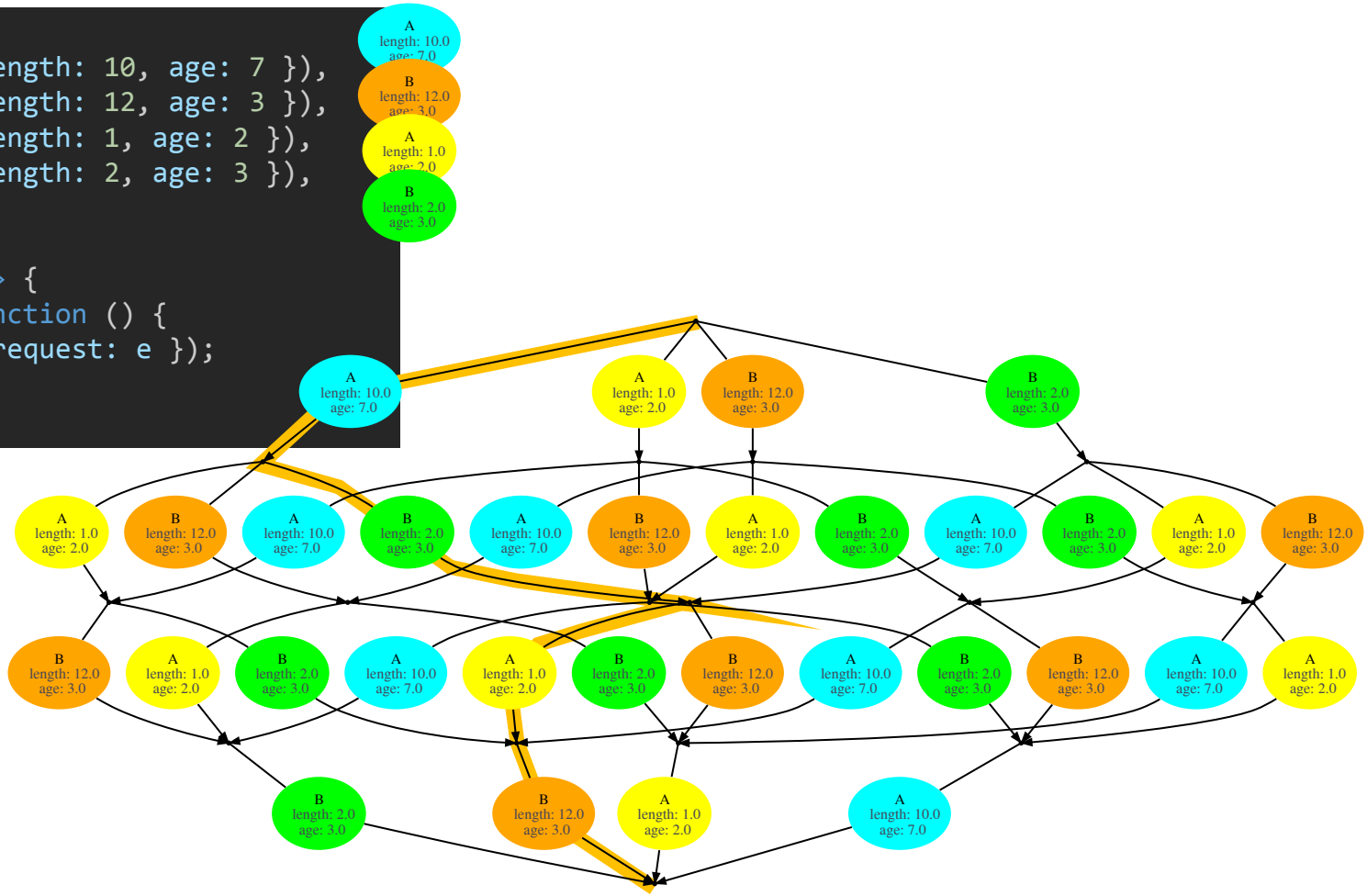


Events with data

```
const events = [
    Event('A', { length: 10, age: 7 }),
    Event('B', { length: 12, age: 3 }),
    Event('A', { length: 1, age: 2 }),
    Event('B', { length: 2, age: 3 }),
]

events.forEach(e => {
    bthread("", function () {
        bp.sync({ request: e });
    })
})
```

```
const events = [
    Event('A', { length: 10, age: 7 }),
    Event('B', { length: 12, age: 3 }),
    Event('A', { length: 1, age: 2 }),
    Event('B', { length: 2, age: 3 }),
]

events.forEach(e => {
    bthread("", function () {
        bp.sync({ request: e });
    })
})
```

```
const events = [
    Event('A', { length: 10, age: 7 }),
    Event('B', { length: 12, age: 3 }),
    Event('A', { length: 1, age: 2 }),
    Event('B', { length: 2, age: 3 }),
]

events.forEach(e => {
    bthread("", function () {
        bp.sync({ request: e });
    })
})
```



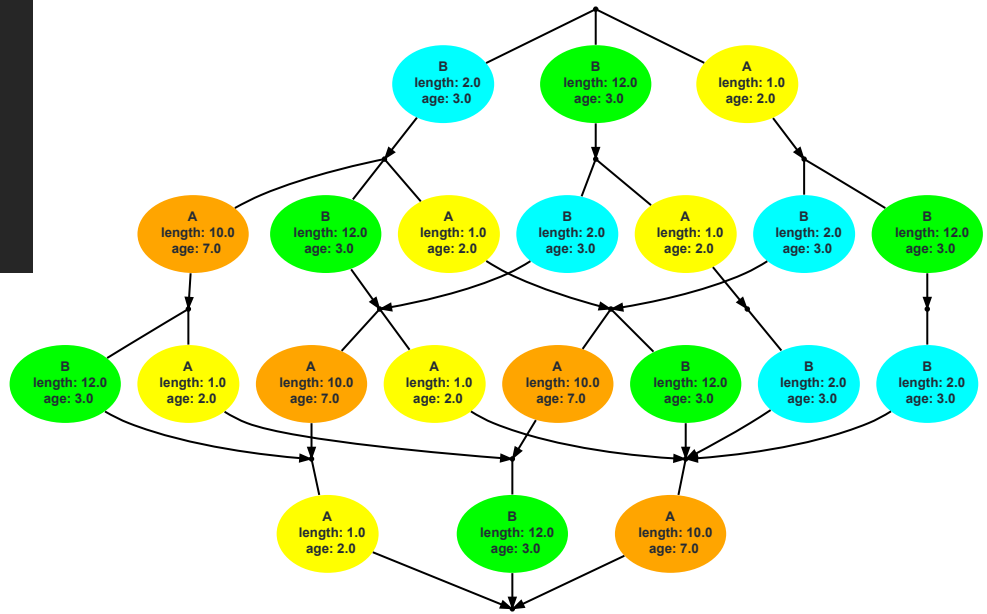
SQ - Model-based Testing

```
bthread("", function () {
    bp.sync({
        waitFor:
            EventSet("B with short length", e => e.name == 'B' && e.data.length < 4),

        block:
            EventSet("A with old age", e => e.name == 'A' && e.data.age > 6)
    });
});
```

```
bthread("", function () {
    bp.sync({
        waitFor:
            EventSet("B with short length", e => e.name == 'B' && e.data.length < 4),

        block:
            EventSet("A with old age", e => e.name == 'A' && e.data.age > 6)
    });
});
```

```
const events = [
    Event('A', { length: 10,
age: 7 }),
    Event('B', { length: 12,
age: 3 }),
    Event('A', { length: 1,
age: 2 }),
    Event('B', { length: 2,
age: 3 }),
]

events.forEach(e => {
  bthread("", function () {
    bp.sync({ request: e });
  })
})
```

```
bthread("", function () {
  bp.sync({
    waitFor:
      EventSet("B with short length", e => e.name == 'B' && e.data.length < 4),
    block:
      EventSet("A with old age", e => e.name == 'A' && e.data.age > 6)
  });
});
```



SQ - Model-based Testing

Introducing Provengo

- Provengo Course