

Extracting Mode Diagrams from Blech Code



`await condition`

Daniel Lucas, Alexander Schulz-Rosengarten, Reinhard von Hanxleden
Kiel University, Germany

Friedrich Gretz, Franz-Josef Grosch
Robert Bosch GmbH, Corporate Research, Renningen, Germany

SYNCHRON 2021, La Rochette, France
Originally FDL'21, Antibes, France, Sep. 8–10

FDL 2018

Forum on specification & Design Languages



@Munich, Germany
September 10-12, 2018

Technical Program

@Munich, Germany



UNIVERSITÀ
di VERONA
Department of
INFORMANICA

Technical
University
of Munich



Technical Program

| | 10.09 Monday | 11.09 Tuesday | 12.09 Wednesday |
|------|--------------------------------|--------------------------------|--------------------------------|
| 8:30 | Registration | Keynote: Franz-Josef Grosch | Keynote: Jean-Pierre Talpin |
| 9:00 | Tutorial: Heterogeneous System | | 2 |



● Franz-Josef
Grosch

🏛 Robert Bosch
GmbH

2nd Keynote: Blech - a safe synchronous language for embedded real-time programming

Abstract:

Product development at companies such as Bosch requires systems engineering for digital hardware and mechatronic components as well as software engineering for deeply-embedded resource-constrained real-time applications cooperating with distributed cloud applications. While many of the involved engineering disciplines greatly benefit from model-based approaches and from advances in software infrastructures, deeply embedded software is still based on manually written C code, a few components generated from models and glued together with the help of an embedded operating systems like OSEK. Making software safe with the help of tight coding conventions and static analyzers is a time-consuming task. Modern implementation technologies to address software architecture and qualities or to make embedded programming appealing for software professionals are largely missing. We regard synchronous languages to be suitable for solving many of the issues in the integration (causality) and synchronization (clocks) of time-triggered and event-triggered embedded functions that exhibit their behavior over time steps and are coordinated according to their mode-switching in a structured synchronous control flow. Unfortunately, existing synchronous languages do not support modern implementations technologies well such as aggregated data types, object-based programming and separate compilation. Searching for an imperative, strongly-typed, synchronous language (with deterministic concurrent composition, and synchronous control flow), equipped with the aforementioned features for encapsulation and composition (aggregated data types, modules, separate compilation) and supporting programming parallel tasks deployed to separate cores (clock refinement and

What this talk is about ...

~~New language/model of computation~~

~~Softwaresynthesis~~

Semantics



Documentation



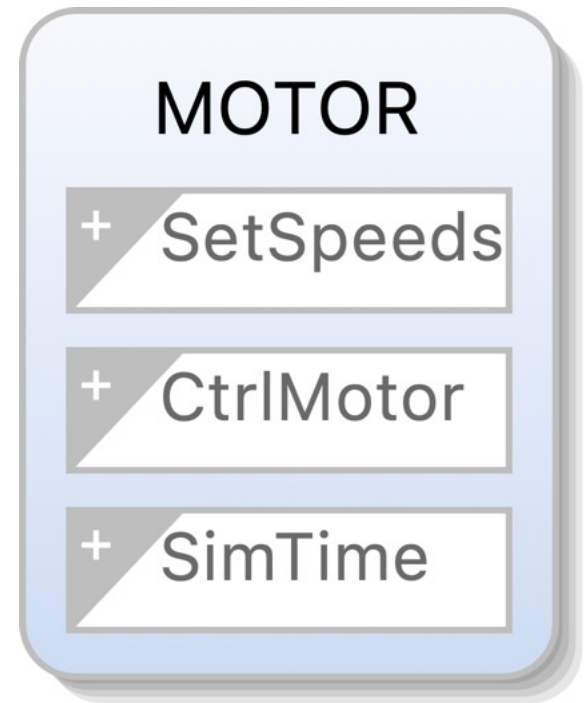
```

1 1/**
2 Controller for stepper motor
3 */
4 #define MOTOR {
5   output int currentSec = 0
6   output int maxSec
7   input bool accel , decel
8   input bool stop
9   output bool motor = false
10  output float v
11  output int pMotorSec
12  int pSetSpeedMinSec = 100000
13  int pSetSpeedMaxSec = 100000
14  output int pSec
15  output int pMinSec = pSetSpeedMinSec
16  float dt = 1
17  float vMax = 20
18  float cSPeriodPeriod = 1
19
20 /** Process user inputs, set the speed/motor period */
21 region SetSpeeds:
22
23   initial state SetSpeeds "" {
24     bool clk
25
26     region ProcessInputs:
27
28     initial state Init
29     immediate go to Running
30
31     state Running {
32       entry do v = 0
33
34       region CalcV:
35
36       initial state Pause
37       if clk & accel & !decel go to Accel
38       if clk & decel & !accel go to Decel
39
40       state Accel
41       immediate do v := dt go to CheckMax
42
43       state Decel
44       immediate do v := -dt go to CheckMin
45
46       state CheckMax
47       immediate if v := vMax go to SetPeriod
48       immediate do v = vMax go to SetPeriod
49
50       state CheckMin
51       immediate if v := -vMax go to SetPeriod
52       immediate do v = -vMax go to SetPeriod
53
54       state SetPeriod
55       immediate if v == 0 do pMotorSec = 0 go to Pause
56       immediate do pMotorSec = 100000 * cSPeriodPeriod / v go to Pause
57     }
58     if clk & stop do pMotorSec = 1000 abort to Running
59
60     region GenClk:
61
62     initial state GenClkState {
63       int myMaxMinSec, myMinMaxSec
64
65       initial state Init
66       immediate do clk = true;
67       myMaxMinSec = currentSec + pSetSpeedMinSec;
68       myMinMaxSec = currentSec + pSetSpeedMaxSec
69       go to AssertWakeTime
70
71       connector state AssertWakeTime
72       immediate do wakeSec = myMinMaxSec go to Pause
73
74       @SuppressWarnings("LAST")
75       state Pause
76       if currentSec = myMaxMinSec do clk = false go to AssertWakeTime
77       go to Init
78     }
79   }
80
81   region CtrlMotor:
82
83   initial state CtrlMotor "" {
84     bool clk
85
86     region GenClk:
87
88     initial state GenClkState "" {
89       int myMaxMinSec
90
91       initial state Stopped
92       immediate if pMotorSec > 0
93         do myMaxMinSec = currentSec + pMotorSec;
94         clk = true go to AssertWakeTime
95
96       connector state AssertWakeTime
97       immediate do wakeSec = myMaxMinSec go to Running
98
99       @SuppressWarnings("LAST")
100      state Running
101      do clk = false go to ResetClock
102
103       connector state ResetClock
104       immediate if pMotorSec > 0 & currentSec < myMaxMinSec
105         go to AssertWakeTime
106       immediate go to Stopped
107     }
108
109     region Motor:
110
111     initial state Low
112     if clk do motor = true go to High
113
114     state High
115     if clk do motor = false go to Low
116   }
117
118   region SimTime:
119
120   initial state SimTimeState "" {
121     during do pSec = currentSec;
122     currentSec = pre(wakeSec);
123     pSec = wakeSec = pre(wakeSec);
124     pMinSec = min pSec;
125     ** during do pSec = currentSec;
126     currentSec = pre(wakeSec);
127     pSec = currentSec = pSec;
128     pMinSec = min pSec */
129   }
130 }

```

Motivation

VS.



Key for understanding: Abstraction

Goal

Facilitate understanding of state-oriented software

1. What are the states?
2. When do we change state?
3. What hierarchy is there?
4. What concurrency is there?
5. ...

Illustrate/validate this with Blech language

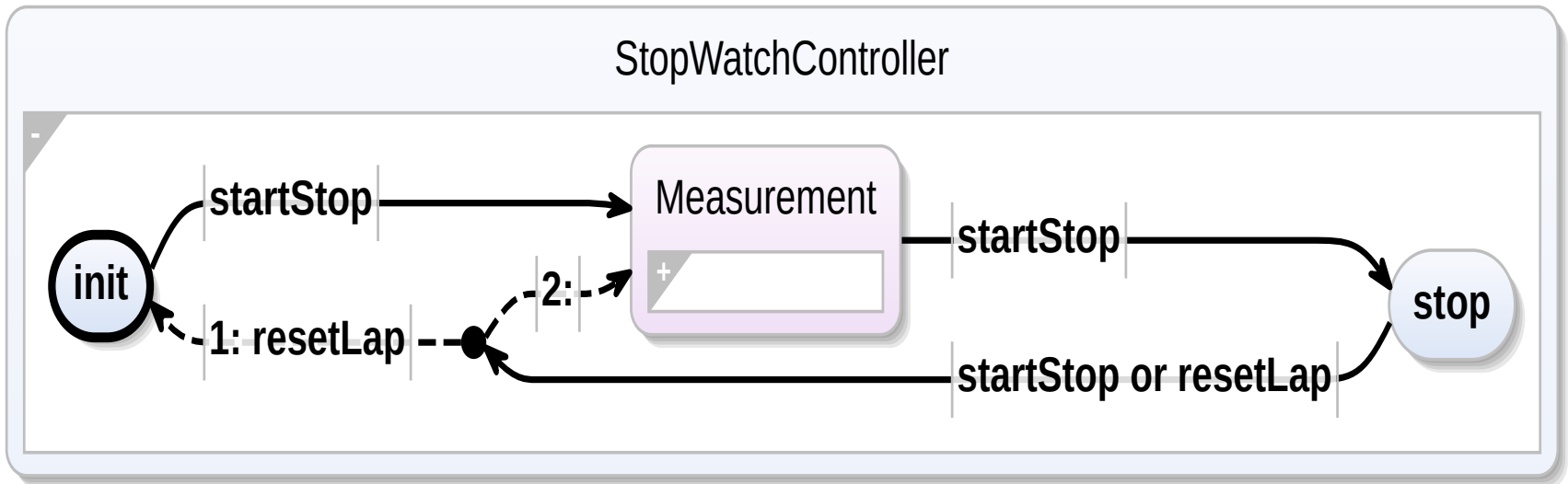
However, the general approach should be applicable to other languages as well!

Related Work

- Gracanin et al.
Software Visualization
Innovations in Systems and Software Engineering
2005
- Fuhrmann, von Hanxleden
Taming Graphical Modeling
MODELS'10
- Sen, Mal
Extracting finite state representation of Java programs
Software & Systems Modeling 2016
- Smyth et al.
Model extraction for legacy C programs with SCCharts
ISoLa'16
- Prochnow et al.
Synthesizing Safe State Machines from Esterel
LCTES'06

Example

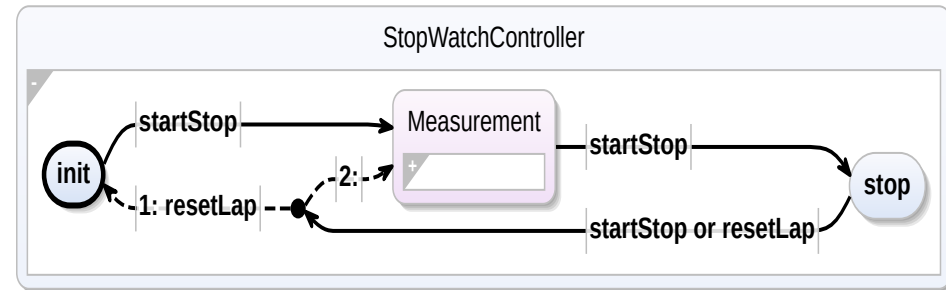
1. What are the states?
2. When do we change state?
3. What hierarchy is there?



Example

1. What are the states?
2. When do we change state?
3. What hierarchy is there?

```
1 activity StopwatchController
2 (startStop: bool, resetLap: bool) // Read-only inputs
3 (display: Display) // Read-write outputs
4 var totalTime: int32
5 var lastLap: int32
6 repeat
7     totalTime = 0
8     lastLap = 0
9     writeTicksToDisplay(totalTime)(display)
10    await startStop // State init
11    repeat
12        cobegin weak
13            await startStop
14        with weak
15            run Measurement(resetLap)(totalTime, lastLap, display)
16        end
17        writeTicksToDisplay(totalTime)(display)
18        await startStop or resetLap // State stop
19        // Run again if only startStop was pressed
20    until resetLap end // Back to init if resetLap was pressed
21 end
22 end
```



Extraction Process

Phase 1: Structural Translation

From Blech to Mode Diagrams

Phase 2: Label Extraction

State naming

Phase 3: Optimization

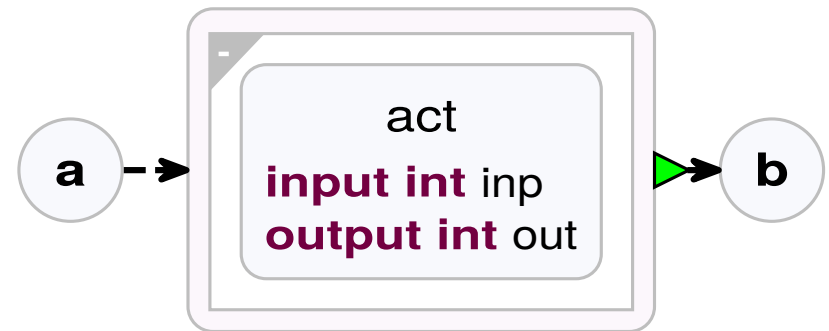
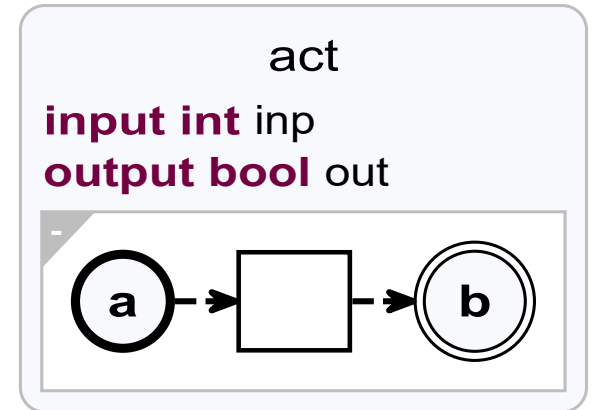
Hierarchy Flattening

Transient State Elimination

Phase 1: Structural Translation

```
activity act (inp: int32) (out: bool)  
  //...  
end
```

```
run act(inp)(out)
```

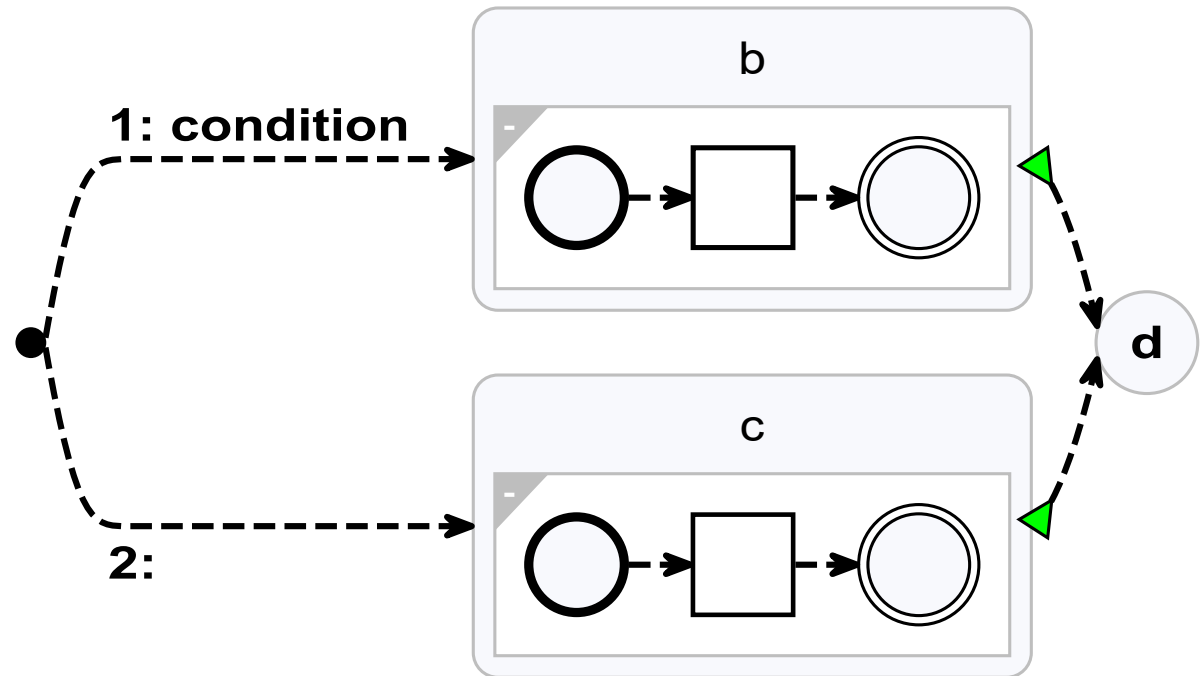


Phase 1: Structural Translation

await condition

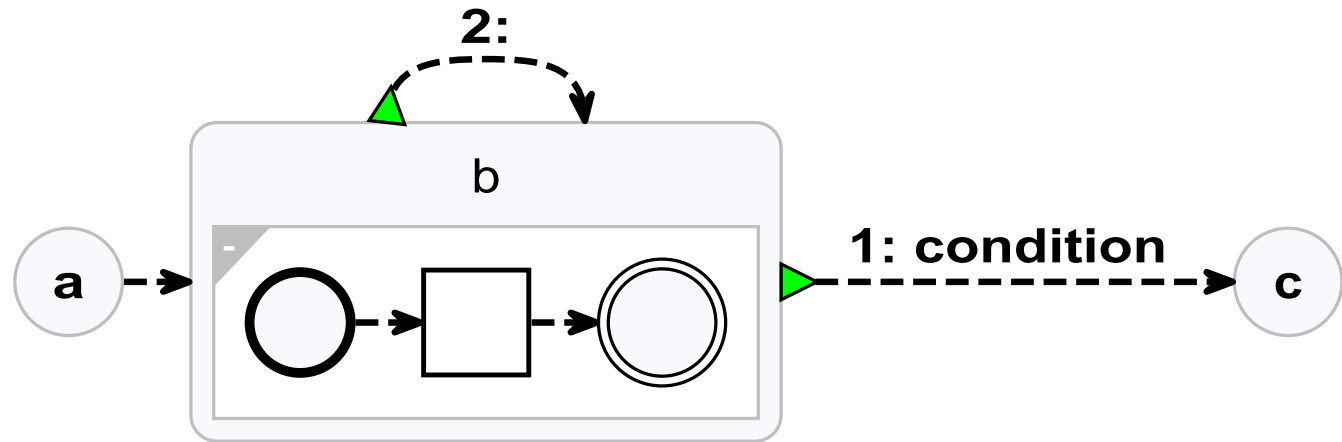


```
if condition then  
  // ..  
else  
  // ..  
end
```

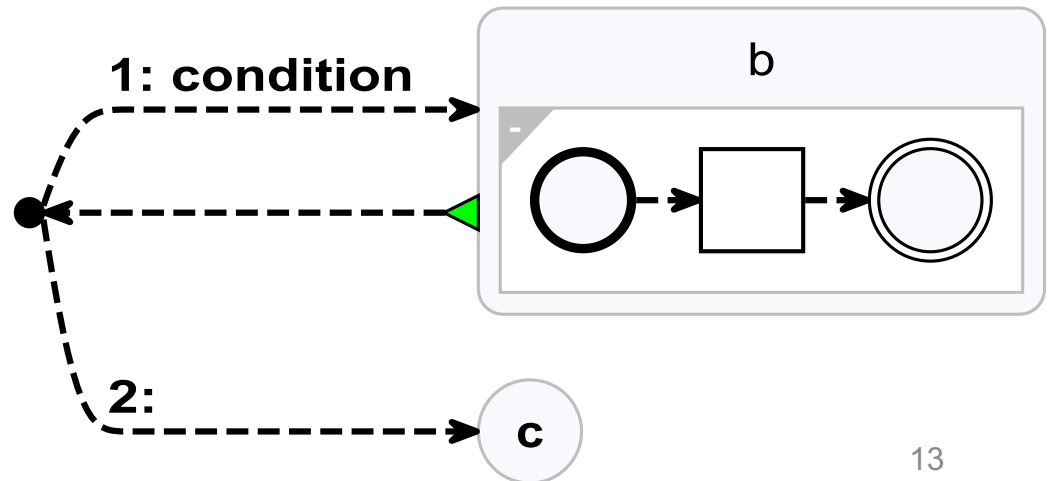


Phase 1: Structural Translation

```
repeat  
  // ...  
until condition end
```

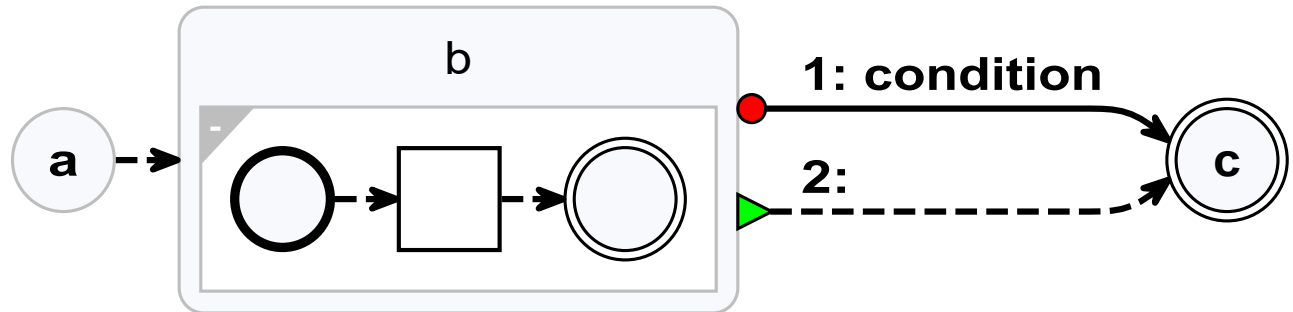


```
while condition repeat  
  // ...  
end
```

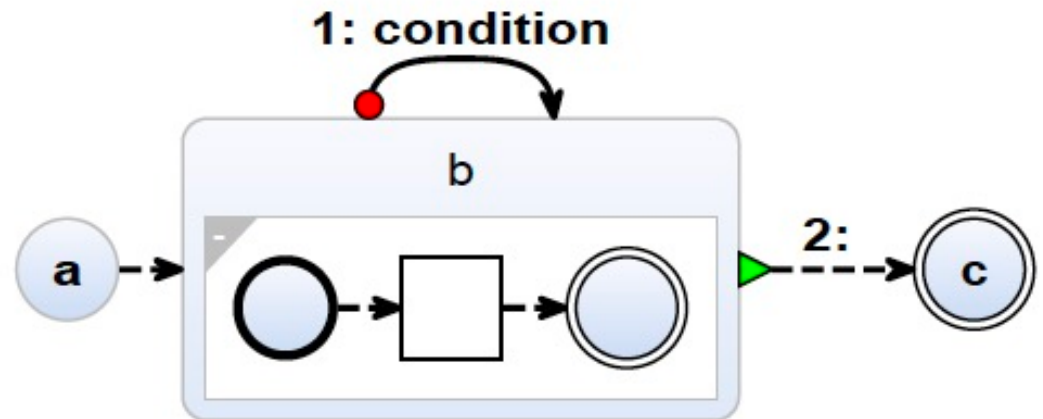


Phase 1: Structural Translation

```
when condition  
  abort  
  // ..  
end
```

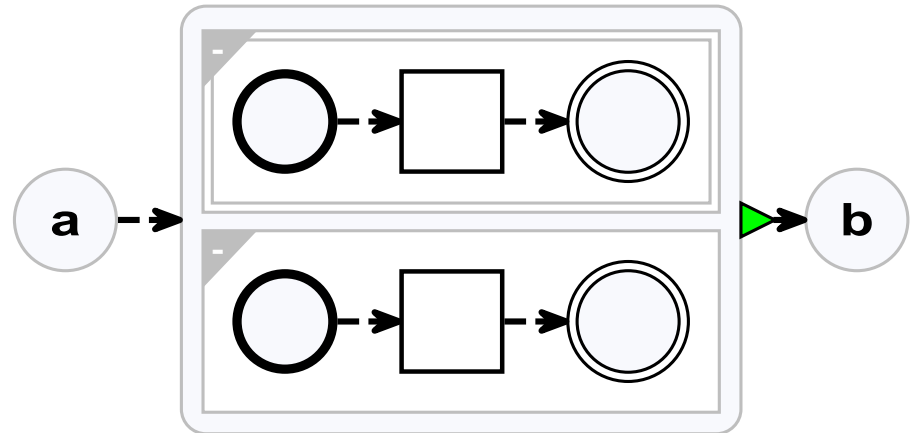


```
when condition reset  
  // ...  
end
```



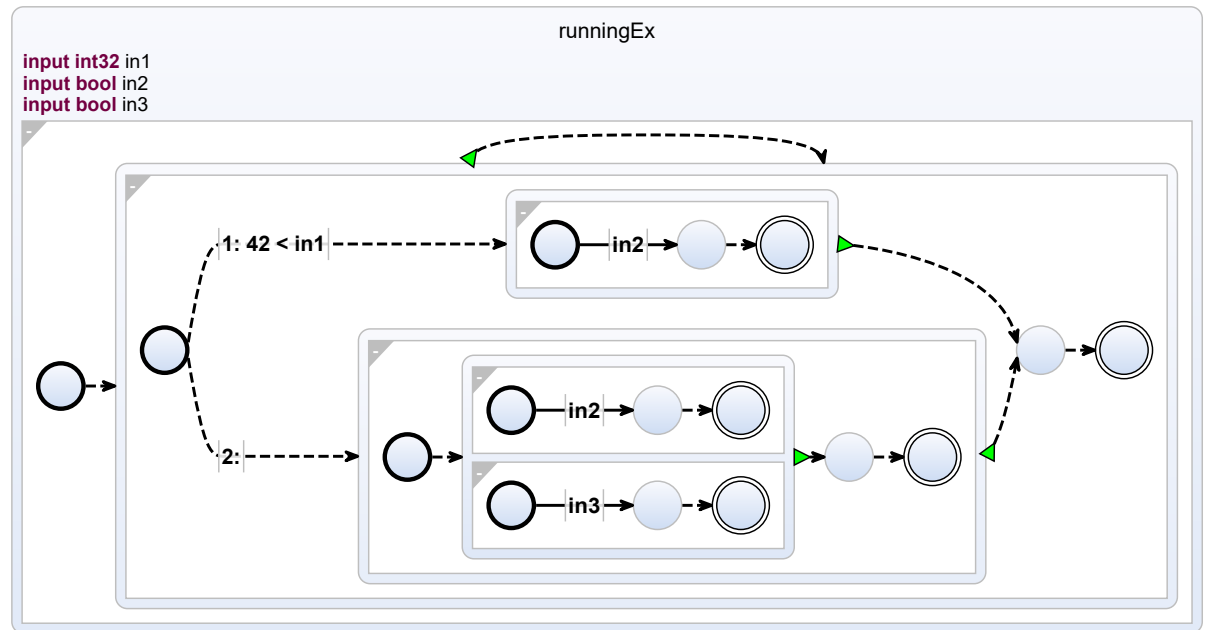
Phase 1: Structural Translation

```
cobegin weak  
  // ..  
with  
  // ..  
end
```



Running Example – After Phase 1

```
activity runningEx
(in1: int32, in2: bool, in3: bool)
repeat
  if in1 > 42 then
    await in2
  else then
    cobegin
      await in2
    with
      await in3
    end
  end
end
end
end
```



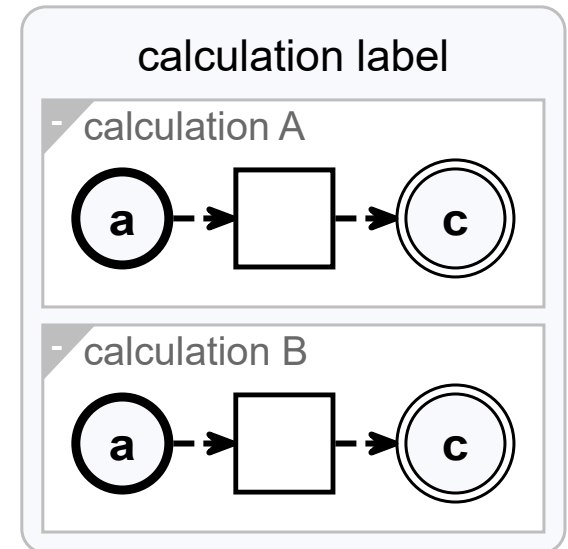
States are not named ⇒ Phase 2, Label Extraction
Bloated structure ⇒ Phase 3, Optimization

Phase 2: Label Extraction

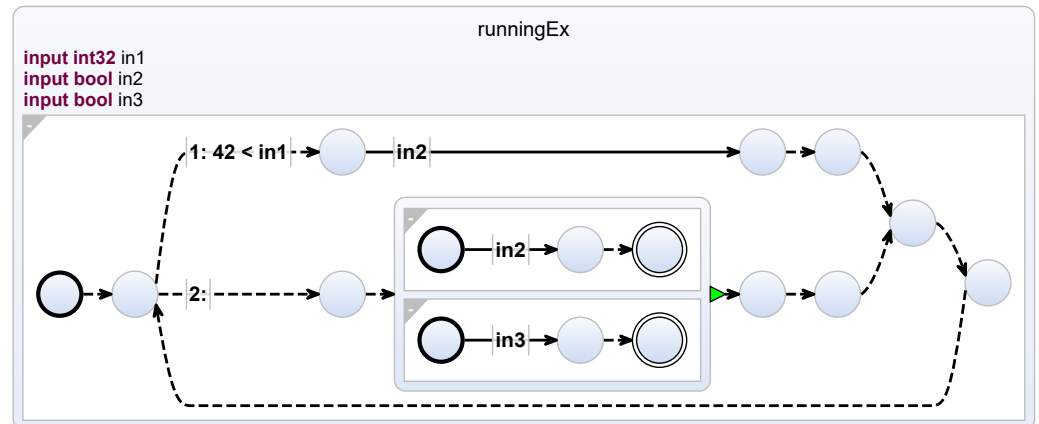
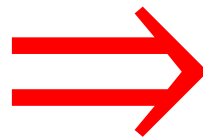
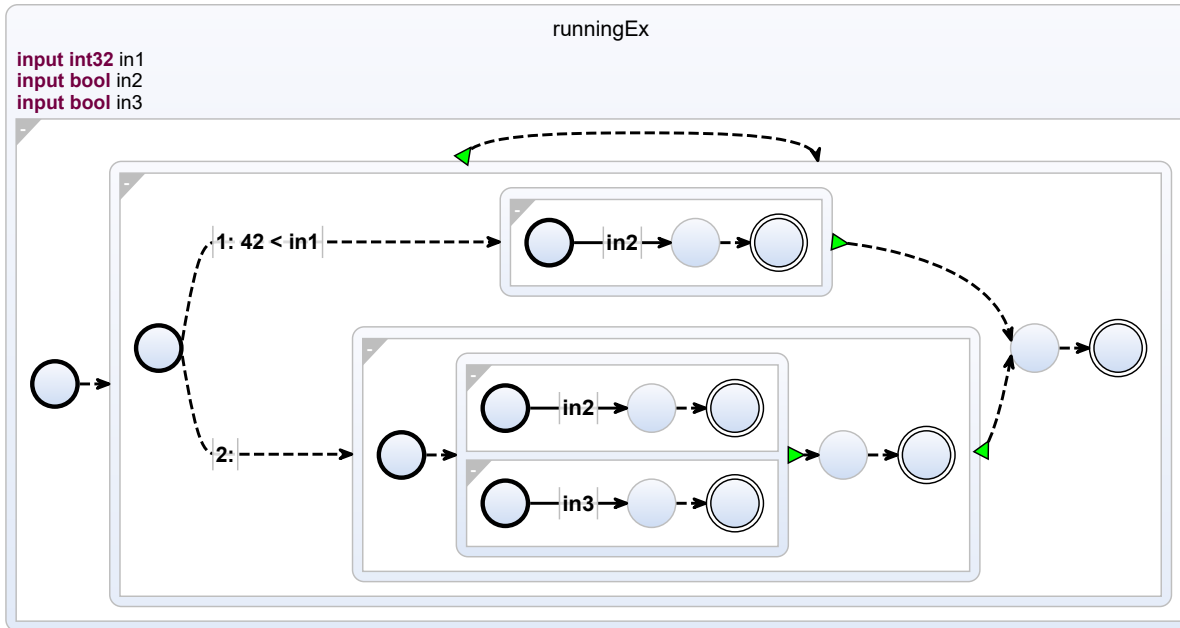
```
@@[label="aLabel"]  
await condition
```



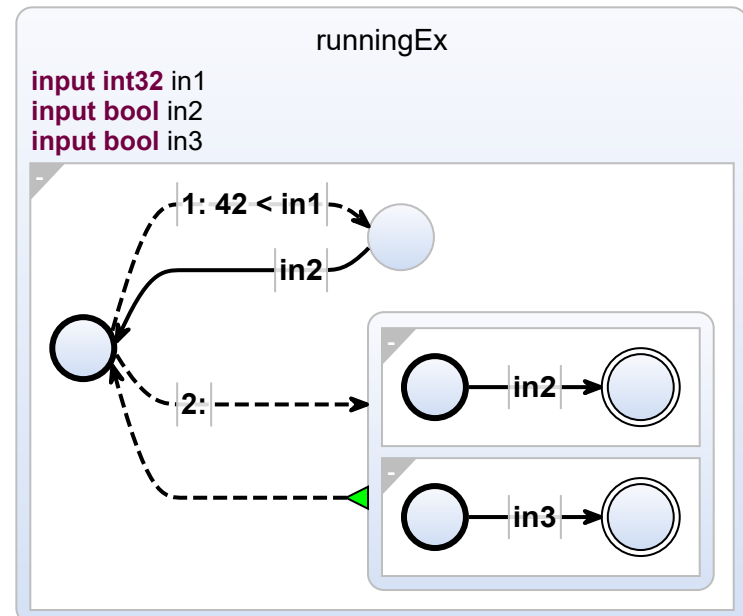
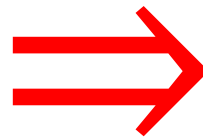
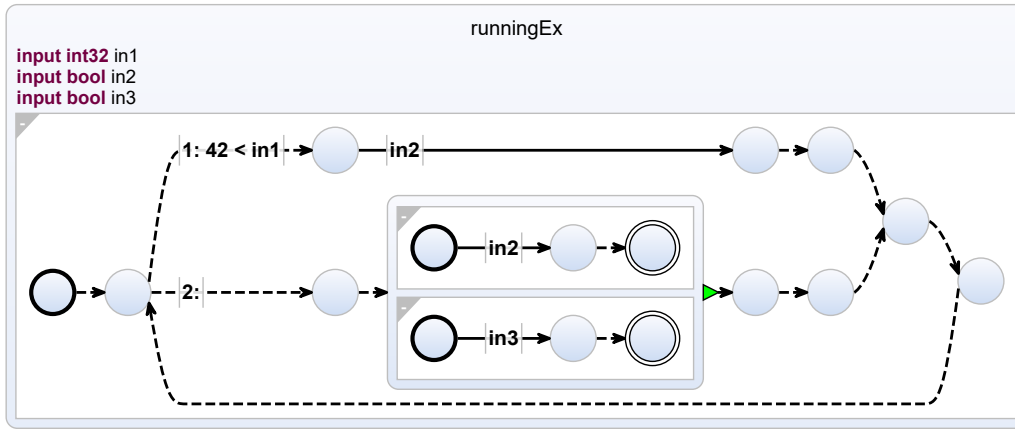
```
@@[cobegin="calculation label"]  
@@[branch="calculation A"]  
@@[branch="calculation B"]  
cobegin  
  // ...  
with  
  // ...  
end
```



Phase 3: Hierarchy Flattening

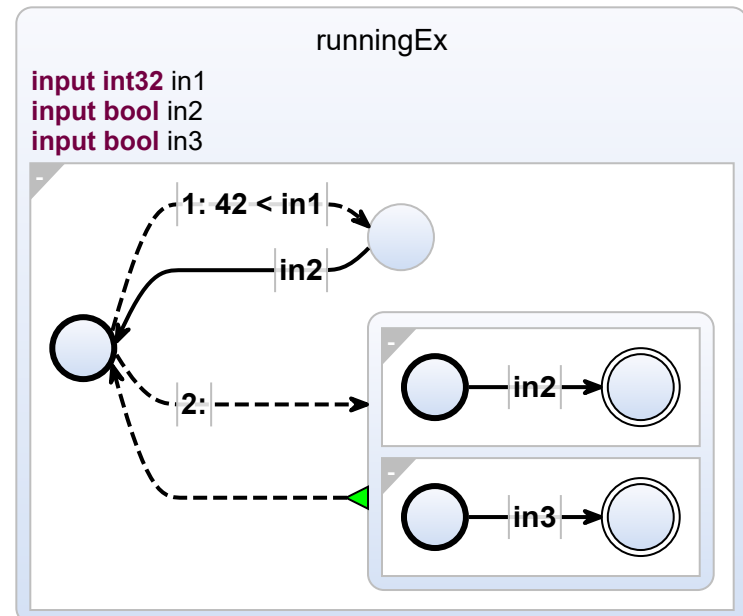


Phase 3: Transient State Elimination



Phase 3: Transient State Elimination

```
activity runningEx
(in1: int32, in2: bool, in3: bool)
repeat
  if in1 > 42 then
    await in2
  else then
    cobegin
      await in2
    with
      await in3
    end
  end
end
end
end
```





Documentation

[Getting Started](#)[User manual](#)[Language Evolution](#)[Blehc development](#)

Blech examples

[Virtual Safe Lock](#)[Blinker](#)[Stopwatch](#)[React](#)[DCF77 decoding](#)[Contributing](#)[Documentation](#) / [Blech examples](#)

Blech examples

Examples of Blech in use.

Virtual Safe Lock

Get a short introduction into the motivation behind Blech and its syntax. Write your first application that runs on actual hardware, the Bosch XDK.

Blinker

A case study from the automotive domain.

Stopwatch

Small example that shows the use of synchronous preemptions and uses an external C function.

React - a reaction test game

A Blech demo for the M5StickC programmed in the Arduino environment

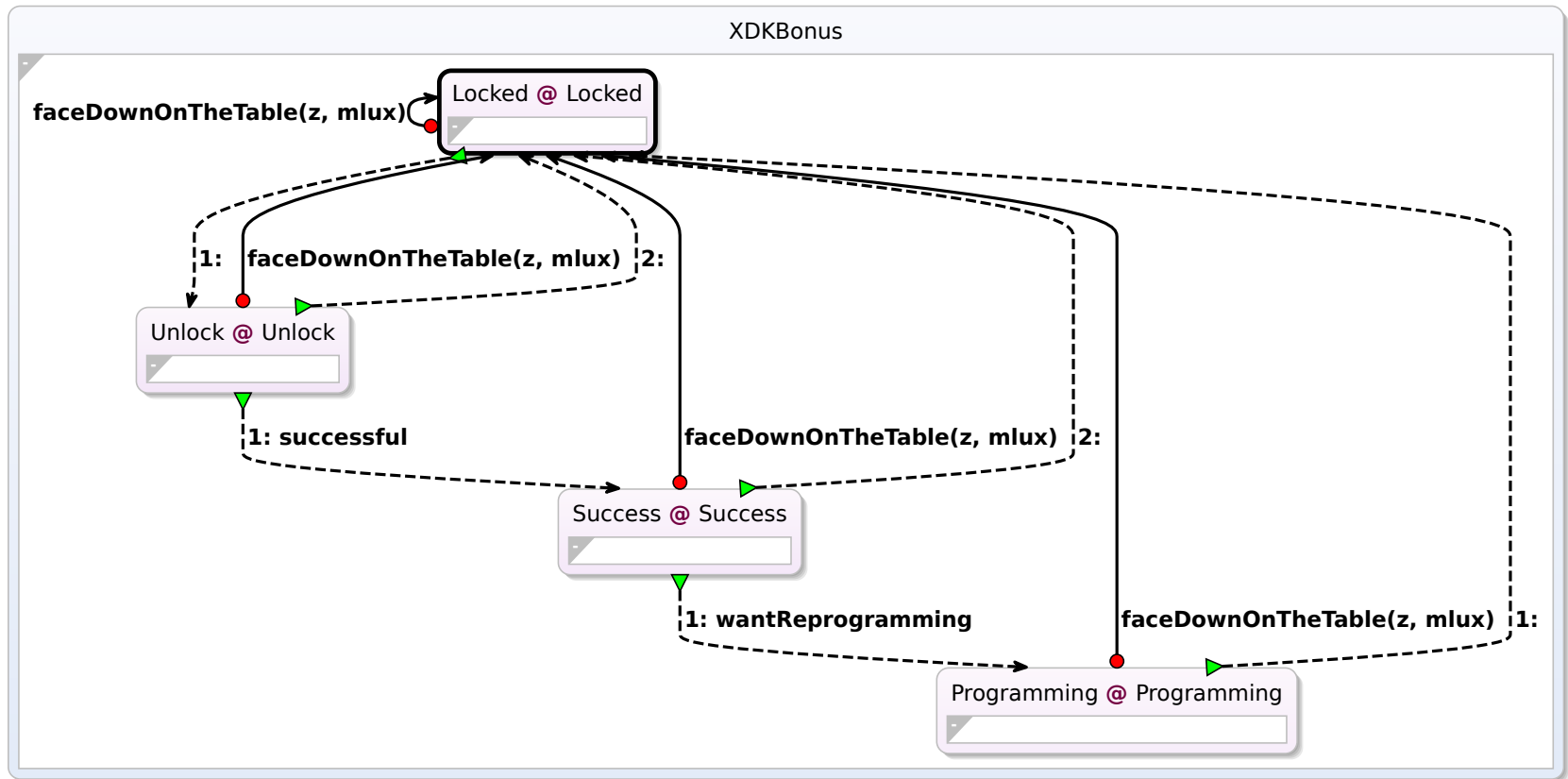
Decoding the DCF77 longwave time signal

A decoder for the DCF77 longwave time signal implemented in Blech on bare metal

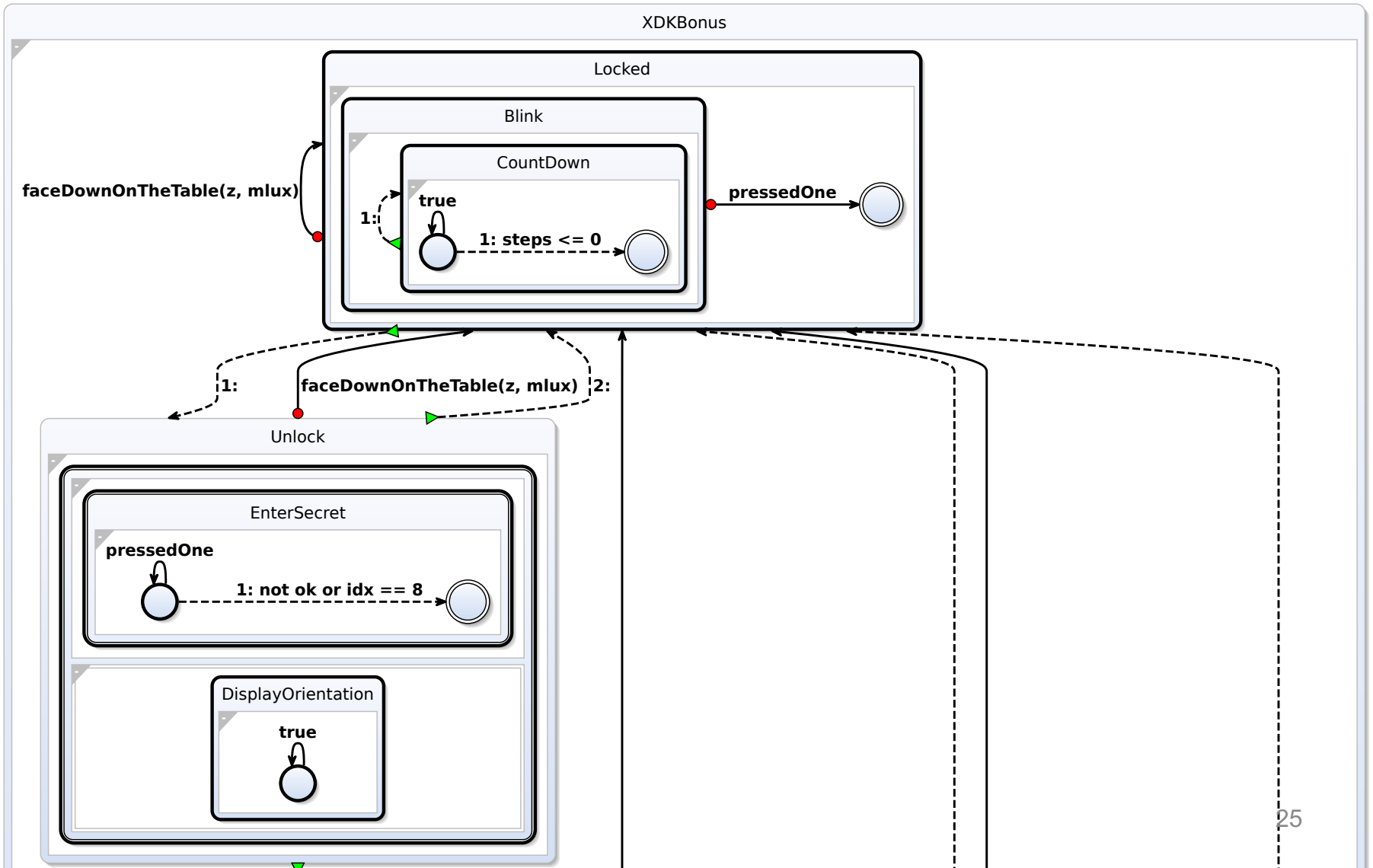
Bleach Code

```
/// This is the reference implementation of the process of locking the locked lock, keep blinking until the user presses button 1
/// initially written for the BLEACH 2019 (or in 2017, it is right to suppose that it is exact)
///
/// // abs(p-v) <= epsilon return pose % EXACT == 0 activity Unlock (secret: [MAXLEN]nat32, ledLeft: bool, ledMiddle: bool, ledRight: bool)
/// // See accompanying lecture notes for the specification. end bool) when pressedOne abort // Button 1: start unlocking
/// return p-v <= PosEpsilon // Determine proximity of given vector to South direction (ledLeft: bool, ledMiddle: bool, ledRight: bool)
/// (c) Robert Bosch GmbH 2019 function isSouthAligned (x: int32, y: int32) returns nat32 ***** returns bool ***** end *****
return v-p <= PosEpsilon then * Misc helper activities end
/***** */ return EXACT ***** var pose: nat32 ***** /
***** */ else if isRightOf(x, y) then var ok = false ***** // Lock has been successfully opened
* Global constant return RIGHTOF // Every led given accelerometer sensor badge var badge: bool ***** // Determine exclusively pressed button
/** */ elseif isLeftOf(x, y) then // sets LED to reflect the pose run DisplayOrientation(x, y, ledLeft, ledMiddle, ledRight, pose) pressedTwo: bool)
***** */ return LEFTOF * activity DisplayOrientation (x: int32, y: int32) (ledLeft: bool, ledMiddle: bool, ledRight: bool)
***** */ else Wkxckk (ledLeft: bool, ledMiddle: bool, ledRight: bool, secret: secret, pressedOne: bool)
const OneG: int32 = 4095 // acceleration value returns ENDEFPOS if OK else OK end successToLEDs((ledLeft, ledMiddle, ledRight))
// consider to be at least 1g (gravity force) NKOkxkXW pose = determineOrientation(x, y) return ok
const PositionEpsilon: int32 = 400 // 10% epsilon WKxdkKW poseToLED(pose)(ledLeft, ledMiddle, ledRight) await pressedOne and not pressedTwo
const Cos45xG: int32 = 2895 // cos(45°) * OneG = sin(45°) * OneG OK wait true or pressedTwo and not pressedOne // exactly one button is pressed
/// point symmetric to isEastAligned
/// We encode pose information by a primitive isSouthAligned (x: int32, y: int32) returns nat32 activity EnterNewSecret (pose: nat32, pressedOne: bool, pressedTwo:
/// For example: we represent the XDK standing upright (z=1) as yDk bool) return true // indicate that we want to reprogram the secret
/// NORTH * EXACT = 2 * 11 = 22 end NOxdkXW // When called, delays execution for a given number of ticks var secret: [MAXLEN]nat32 returns bool
const UNDEFPOS: nat32 = 1 NWxkxkON activity Countdown (ticks: nat32) var idx: nat32 = 0 return false // Button 1 leads back to start
HYPOUSE // Use to isSouthAligned by act steps ticks cobegin weak end
function isEastAligned (x: int32, y: int32) returns nat32 repeat end
const NORTH: nat32 = 2 return isEastAligned(-y, x) OK wait true await pressedOne and not pressedTwo
const EAST: nat32 = 3 end OxdDON ticks = steps - 1 if posesExact(pose) then
const SOUTH: nat32 = 5 WKkdkKNW unOdt steps <= 0 end newSecret[idx] = pose
const WEST: nat32 = 7 WkxkON symmetric to isEastAligned idx = idx + 1 /*****
const EXACT: nat32 = 11 NOxdkXW isWestAligned (x: int32, y: int32) returns nat32 end * Program starts here
const RIGHTOF: nat32 = 13 WKkxkKW return isEastAligned(-x, -y) OK // else inexact position, do not evaluate *****
const LEFTOF: nat32 = 17 XOxxON // Invert the status of all LEDs every half second added == MAXLEN end @[EntryPoint]
NOxkxKW activity Blink () (ledLeft: bool, ledMiddle: bool, ledRight: bool) activity XDKBonus (x: int32, y: int32, z: int32, pressedOne: bool, pressedTwo: bool,
/// The maximum length of the secret is MAXLEN // Determines the XDK position from the x and y values of the accelerometer and not pressedOne (ledLeft: bool, ledMiddle: bool, ledRight: bool)
const MAXLEN: nat32 = 8 NOxDON accelerometer okKvertLEDs((ledLeft, ledMiddle, ledRight)) var secret: [MAXLEN]nat32 = { EXACT * NORTH, EXACT * EAST, EXACT * WEST, EXACT * SOUTH }
WoxdkKW function determineOrientation (x: int32, y: int32) returns nat32 // at least one position has been established UNDEFPOS, UNDEFPOS, UNDEFPOS }
WKxxON // check every direction and take the first that gives a defined end repeat
/***** */ alignment ***** end OK ***** // abort when the device is put face down on the table
***** */ angle of alignment = isNorthAligned (x, y) // The process of setting a new secret when the device is on the table (z, mlux) abort
* Helpers WKkdxOX if UNDEFPOS != alignment then return alignment * NORTH end activity Programming (x: int32, y: int32, pressedOne: bool, pressedTwo:
NOoIXONXOXOKOX alignment = isEastAligned (x, y) ***** run locked (pressedOne)(ledLeft, ledMiddle, ledRight)
***** */ alignment = isSouthAligned (x, y) ***** ledRight: bool ***** // *****
***** */ alignment = isWestAligned (x, y) ***** ledRight: bool ***** // *****
/// invert LEDs' status values UNDEFPOS != alignment then return alignment * SOUTH end returns bool var successful = false
function invertLEDs () (ledLeft: bool, ledMiddle: bool, ledRight: bool) activity EnterSecret (secret: [MAXLEN]nat32, pressedOne: bool, pressedTwo: bool) return successful = Unlock(secret, x, y, pressedOne)(ledLeft, ledMiddle, ledRight)
ledRight = not ledMiddle function isExact (nearG: int32, z: int32) returns bool var newSecret: [MAXLEN]nat32 = UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS }
ledLeft = not ledMiddle return nearG >= OneG * PositionEpsilon var ok = true UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS }
ledMiddle = not ledMiddle and around(nearZero, 0) repeat UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS, UNDEFPOS }
end end await pressedOne var ok = false var wantReprogramming = false
/// Given a pose sets LED to reflect the right (pose) then var ok = false var wantReprogramming = Success(pressedOne, pressedTwo)(ledLeft, ledMiddle, ledRight)
function isRightOf (x: int32, y: int32) (ledLeft: bool, ledMiddle: bool, ledRight: bool) idx = idx + 1 run DisplayOrientation(x, y)(ledLeft, ledMiddle, ledRight)
function successToLEDs () (ledLeft: bool, ledMiddle: bool, ledRight: bool) if idx < MAXLEN and secret[idx] != UNDEFPOS then // guard array access
ledRight = true and opposite >= OneG else false if idx < MAXLEN and secret[idx] == UNDEFPOS then // guard array access
ledLeft = true and PositionEpsilon is false idx = MAXLEN // skip the rest run ok = EnterNewSecret(pose, pressedOne, pressedTwo)(newSecret)
ledMiddle = true and adjacent <= OneG false end pressedTwo)(newSecret) run _ = Programming(x, y, pressedOne, pressedTwo)(secret, ledLeft, ledRight)
end end if pose % EXACT == 0 then else end ledRight)
ledMiddle = true ok = false if ok then end
end end
```

Mode Chart – Top Level



Mode Chart – Expanded



Evaluation

- Implemented prototype based on VS Code and Eclipse Layout Kernel (ELK)
- Asked Blech developers at Bosch to evaluate
 - ⇒ Hierarchy flattening should be configurable
 - ⇒ Knowledge of visual syntax (SCCharts) needed to take full advantage of visualizations
 - ⇒ Visualizations considered helpful, in particular for discussions with people not familiar with code base

Take-Home Message

Presented approach to automatically extract mode diagrams from Blech code

To apply this to YOUR favorite language, you must be able to ...

1. Analyze programs written in your language

*the standard part
like an ordinary compiler*

2. Automatically synthesize graphical views

*the non-standard part
but there are solutions to that as well – talk to us 😊*

That's it – thanks!