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

CEDA IEEE Council on Electronic Design Automation · ifip · IEEE · Università di Verona Dipartimento di Informatica · 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		

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 (✓)

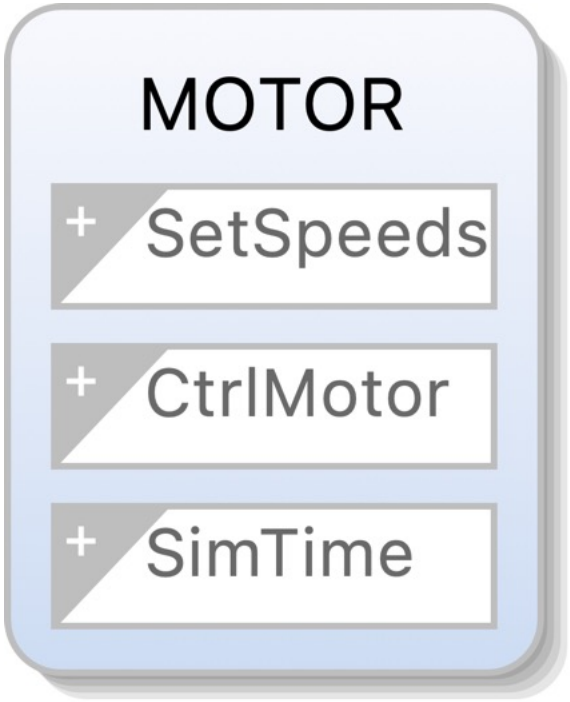
Documentation ✓

motor.sctx

```
/**
 Controller for stepper motor
 */
scchart MOTOR {
  output int currentUsec = 0
  output int wakeUsec
  input bool accel, decel
  input bool stop
  output bool motor = false
  output float v
  output int pMotorUsec
  int pSetSpeedsMinUsec = 500000
  int pSetSpeedsMaxUsec = 500000
  output int pUsec
  output int pMinUsec = pSetSpeedsMaxUsec
  float dV = 2
  float vMax = 20
  float cmPerHalfPeriod = 1

   /** Process user inputs, set the speed/motor period */
  region SetSpeeds:

  initial state SetSpeeds "" {
    bool clk

    region ProcessInputs:

    initial state Init
    immediate go to Running

    state Running {
      entry do v = 0

      region CalcV:

      initial state Pause
      if clk & accel & !decel go to Accel
      if clk & decel & !accel go to Decel

      state Accel
      immediate do v += dV go to CheckMax

      state Decel
      immediate do v -= dV go to CheckMin

      state CheckMax
      immediate if v <= vMax go to SetPeriod
      immediate do v = vMax go to SetPeriod

      state CheckMin
      immediate if v >= -vMax go to SetPeriod
      immediate do v = -vMax go to SetPeriod

      state SetPeriod
      immediate if v == 0 do pMotorUsec = 0 go to Pause
      immediate do pMotorUsec = 1000000 * cmPerHalfPeriod / v go to Pause
    }
    if clk & stop do pMotorUsec = 1000 abort to Running

    region GenClk:

    initial state GenClkState {
      int myWakeMinUsec, myWakeMaxUsec

      initial state Init
      immediate do clk = true;
        myWakeMinUsec = currentUsec + pSetSpeedsMinUsec;
        myWakeMaxUsec = currentUsec + pSetSpeedsMaxUsec
        go to AssertWakeTime

      connector state AssertWakeTime
      immediate do wakeUsec = myWakeMaxUsec go to Pause

      @Layout(LayerConstraint) LAST
      state Pause
      if currentUsec < myWakeMinUsec do clk = false go to AssertWakeTime
      go to Init
    }
  }

  region CtrlMotor:

  initial state CtrlMotor "" {
    bool clk

    region GenClk:

    initial state GenClkState "" {
      int myWakeUsec

      initial state Stopped
      immediate if pMotorUsec > 0
        do myWakeUsec = currentUsec + pMotorUsec;
        clk = true go to AssertWakeTime

      connector state AssertWakeTime
      immediate do wakeUsec min= myWakeUsec go to Running

      @Layout(LayerConstraint) LAST
      state Running
      do clk = false go to ResetClock

      connector state ResetClock
      immediate if pMotorUsec > 0 & currentUsec < myWakeUsec
        go to AssertWakeTime
      immediate go to Stopped
    }

    region Motor:

    initial state Low
    if clk do motor = true go to High

    state High
    if clk do motor = false go to Low
  }

  region SimTime:

  initial state SimTimeState "" {
     during do pUsec = currentUsec;
     currentUsec = pre(wakeUsec);
     pUsec = wakeUsec - pre(wakeUsec);
     pMinUsec min= pUsec
     /* during do pUsec = currentUsec;
     currentUsec = pre(wakeUsec);
     pUsec = currentUsec - pUsec;
     pMinUsec min= pUsec */
  }
}
```

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

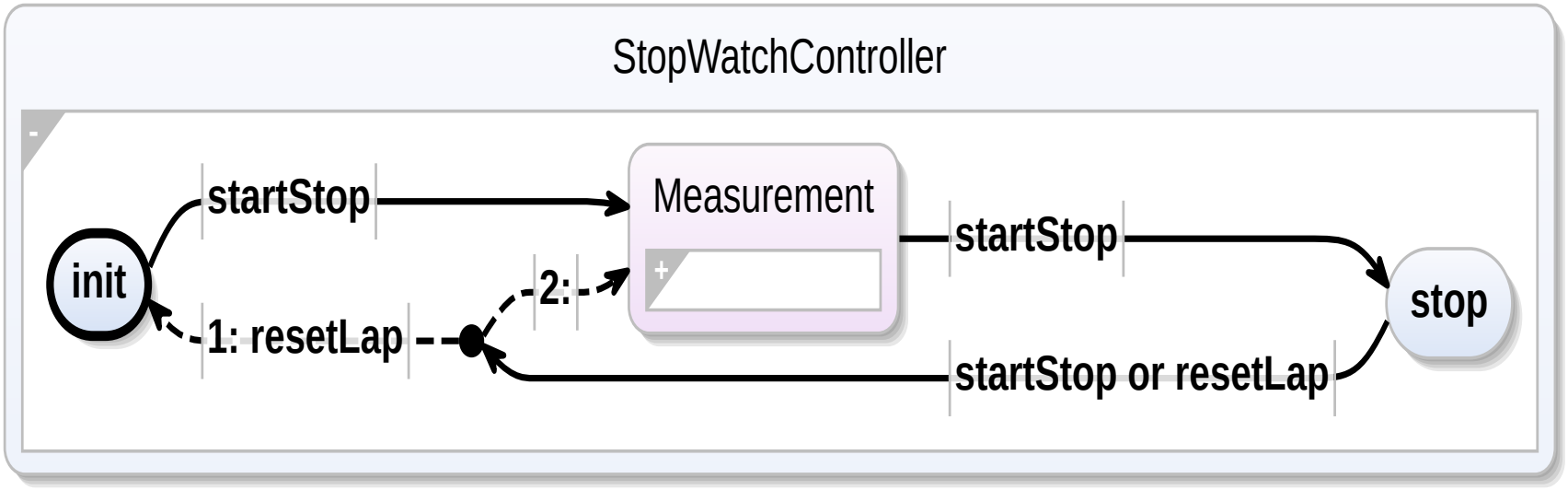
However, the general approach should be applicable to other languages as well!

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

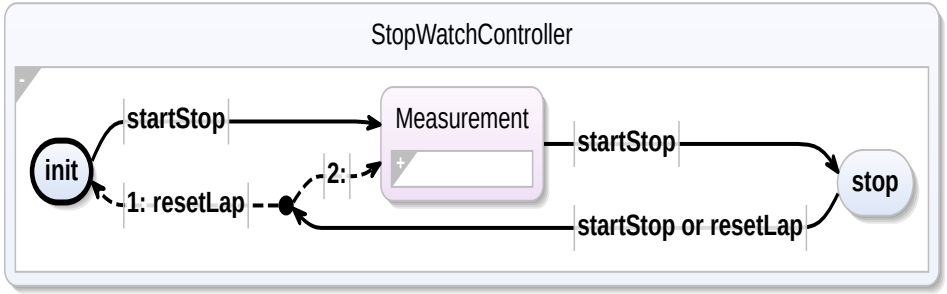
1. What are the states?
2. When do we change state?
3. What hierarchy is there?



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

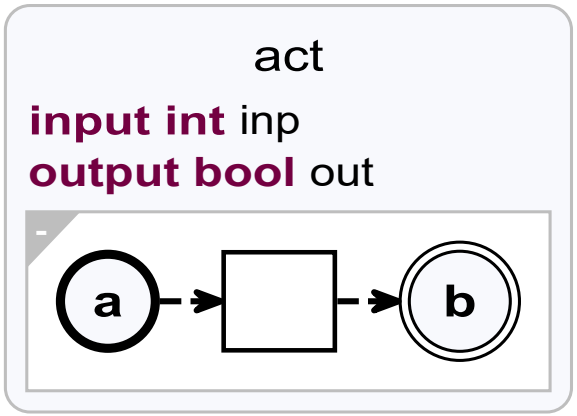
- State naming

Phase 3: Optimization

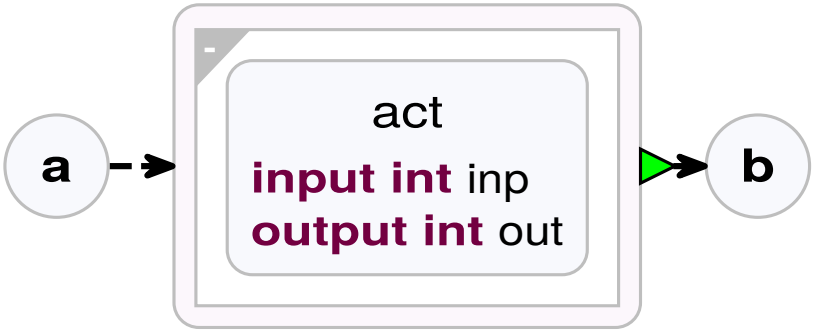
- Hierarchy Flattening
- Transient State Elimination

Phase 1: Structural Translation

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



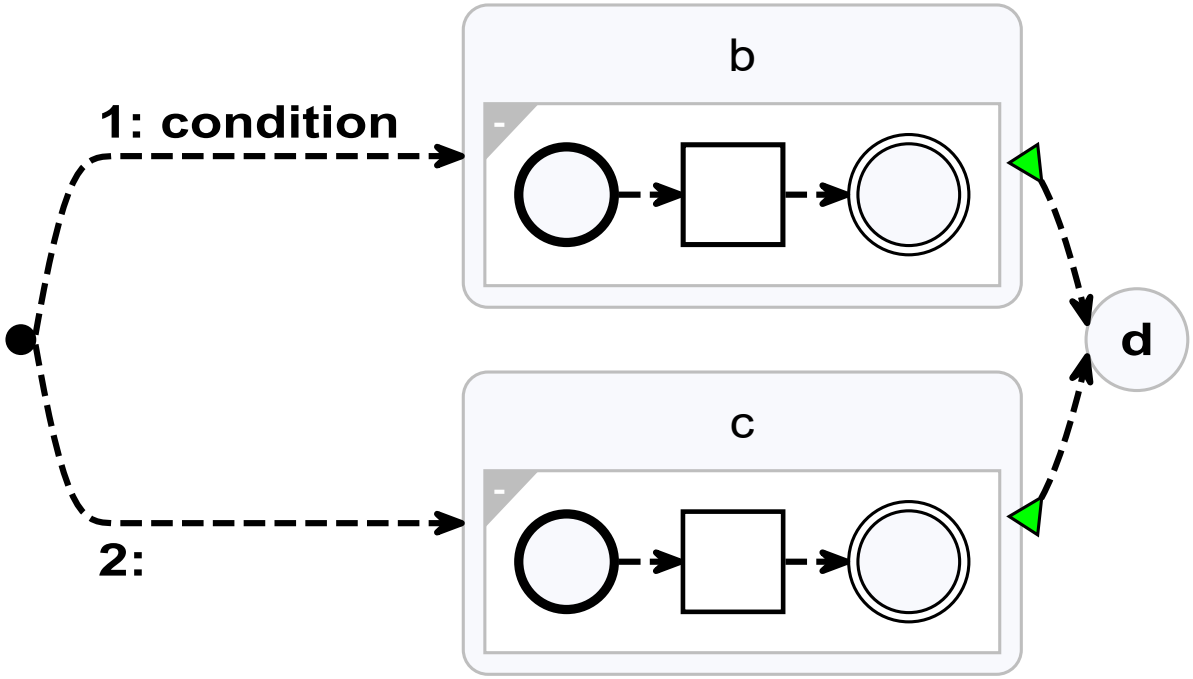

```
run act(inp)(out)
```



Phase 1: Structural Translation

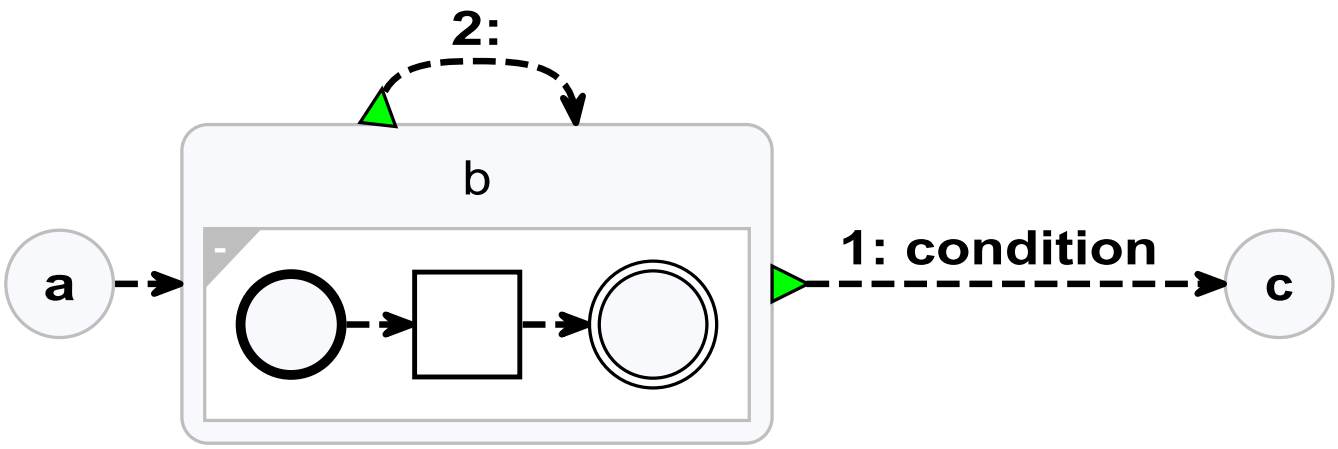
```
await condition
```


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

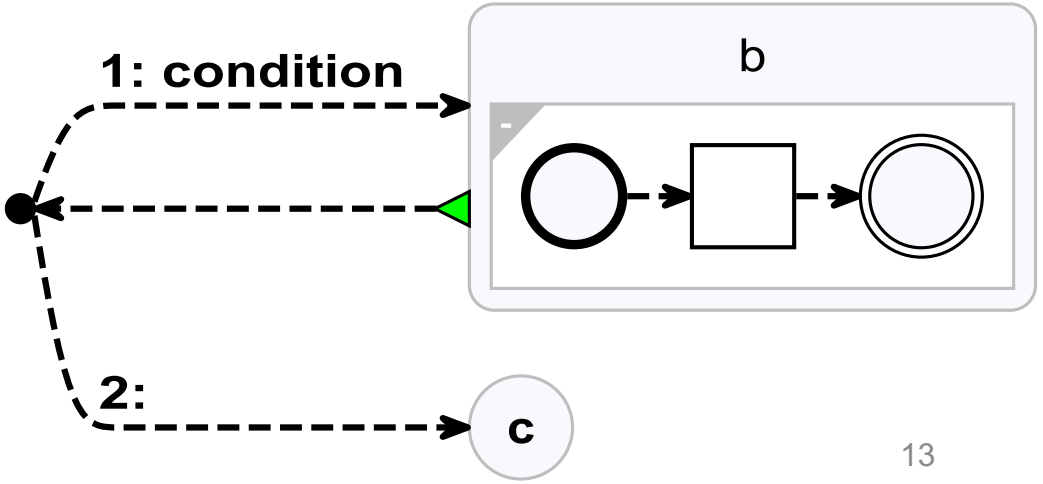


Phase 1: Structural Translation

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

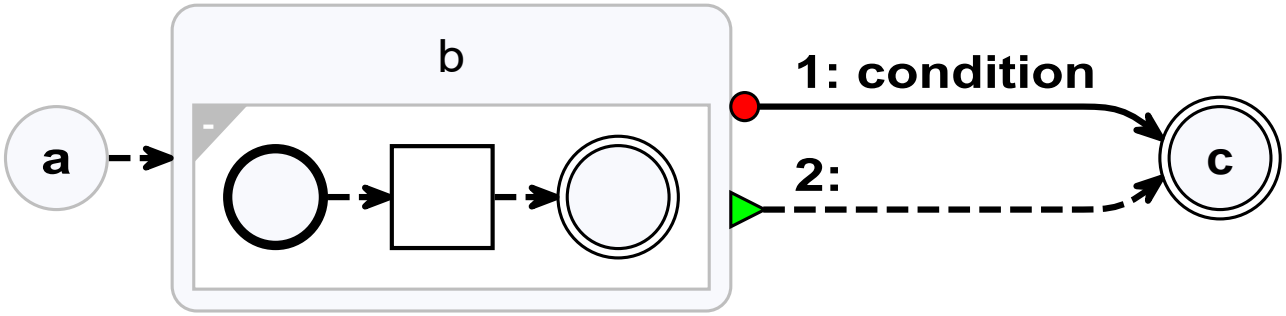



```
while condition repeat
    // ...
end
```

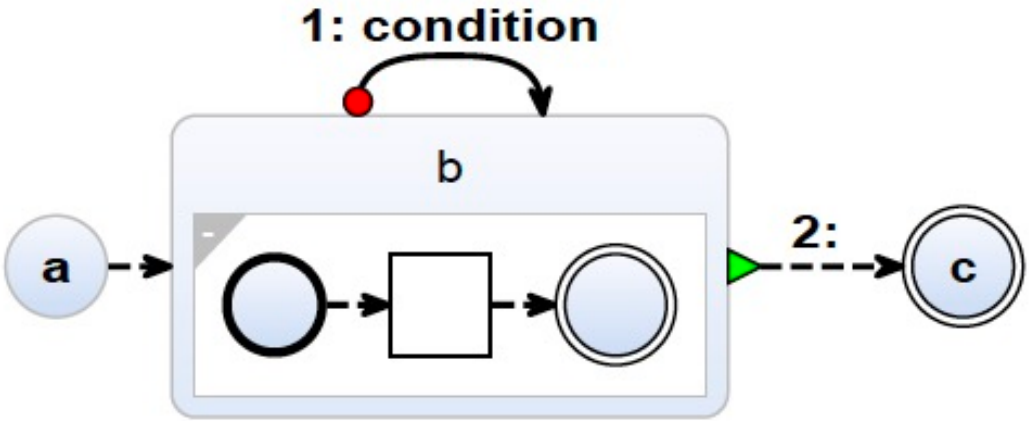


Phase 1: Structural Translation

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

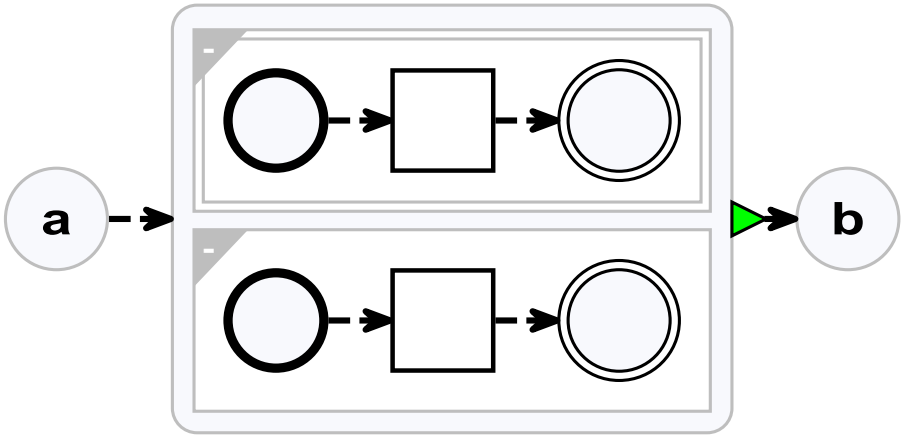



```
when condition reset
   // ...
end
```



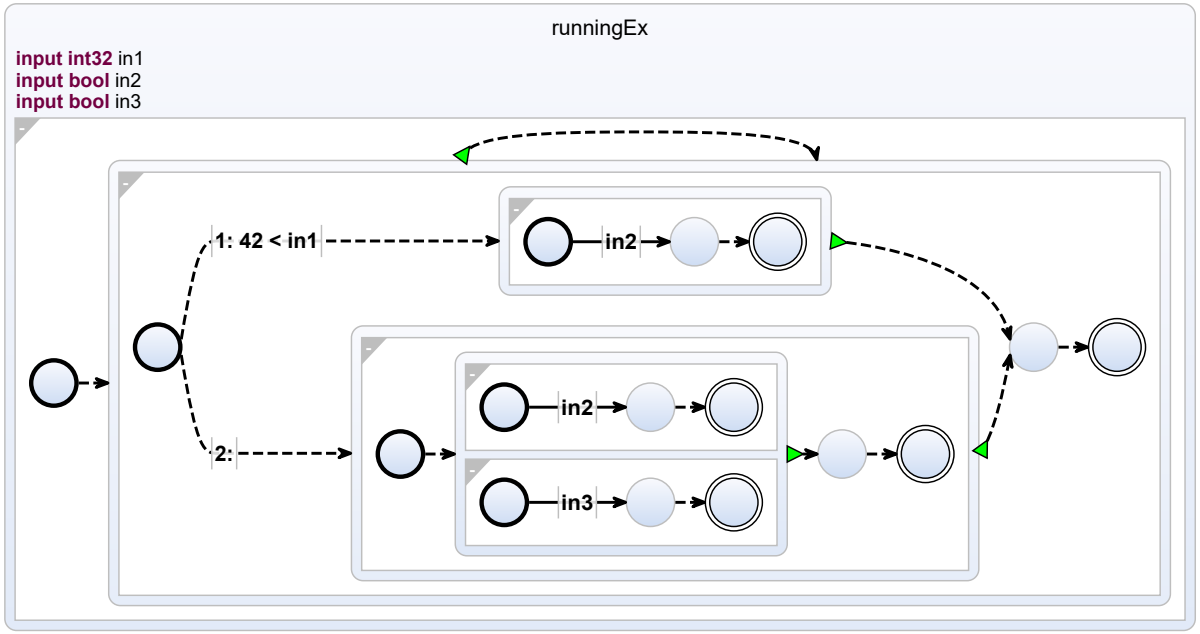
Phase 1: Structural Translation

```
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
```

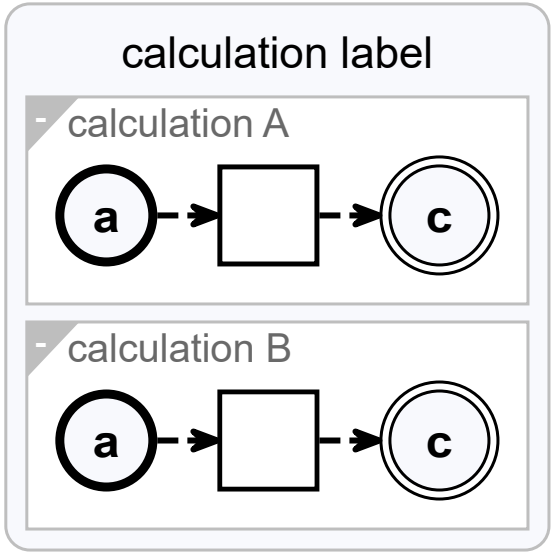


States are not named ⇒ Phase 2, Label Extraction
Bloated structure ⇒ Phase 3, Optimization

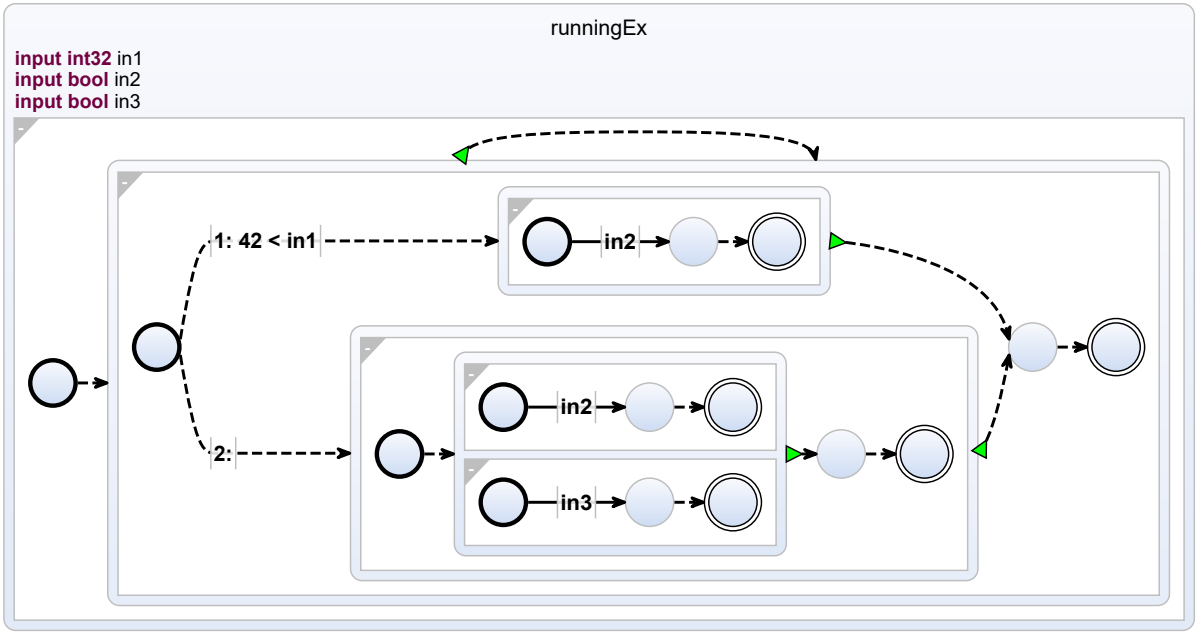
Phase 2: Label Extraction

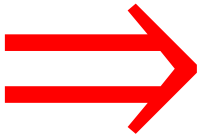
```
@@[label=”aLabel”]
await condition
```

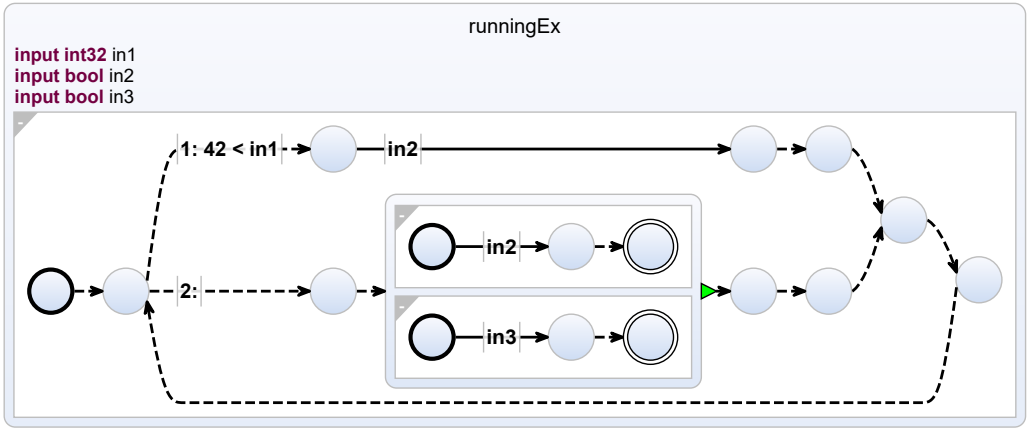

```
@@[cobegin=”calculation label”]
@@[branch= ”calculation A”]
@@[branch= ”calculation B”]
cobegin
   // ...
with
   // ...
end
```



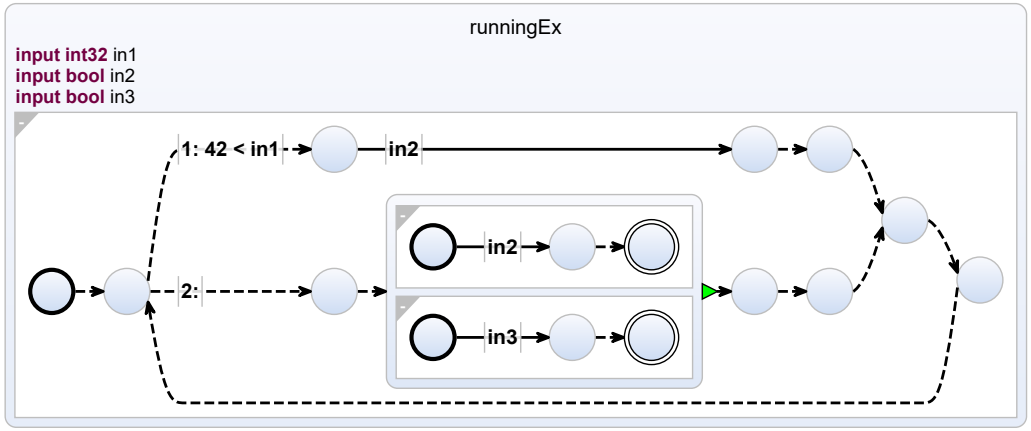
Phase 3: Hierarchy Flattening

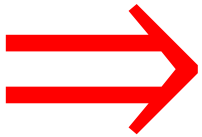


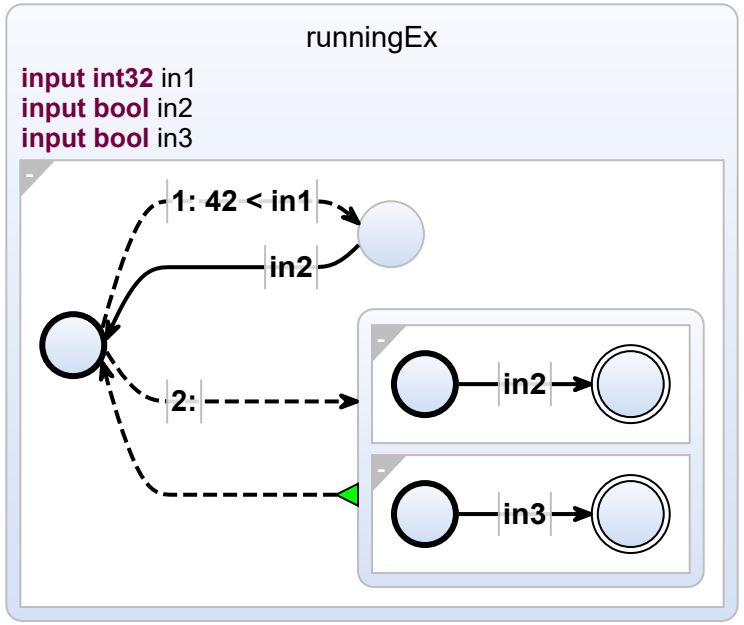




Phase 3: Transient State Elimination

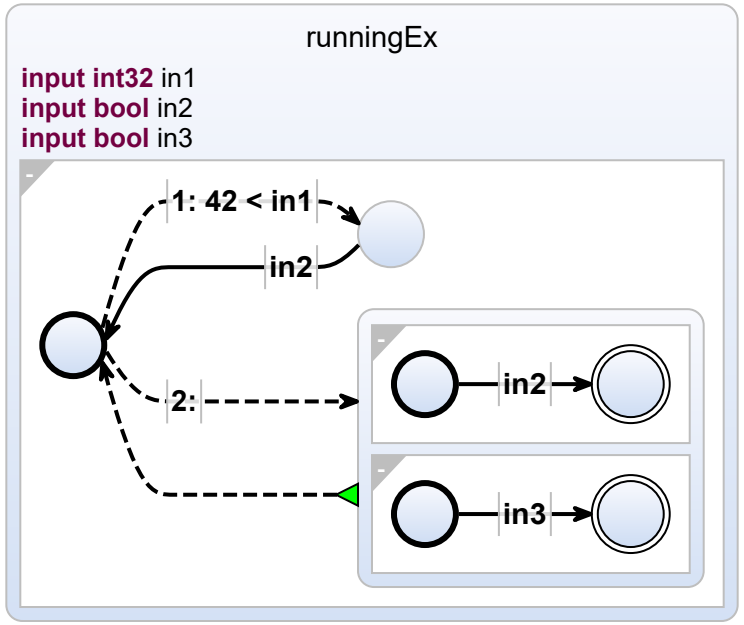






Phase 3: Transient State Elimination

```
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
```



Search this site...

Documentation

Getting Started
User manual
Language Evolution
Blechc development
Blech examples
Virtual Safe Lock
Blinker
Stopwatch
React
DCF77 decoding
Contributing

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

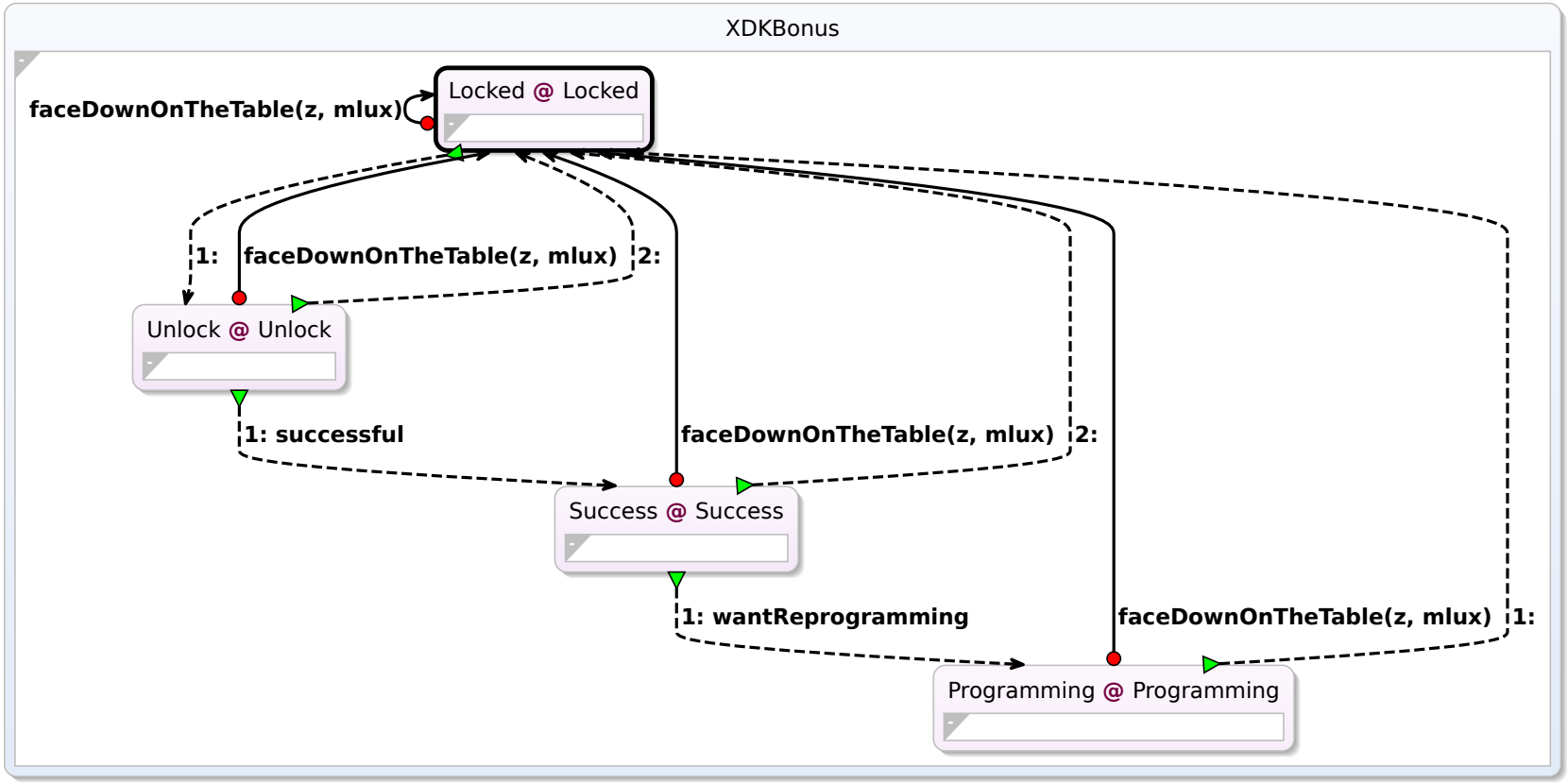
Decoding the DCF77 longwave time signal

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

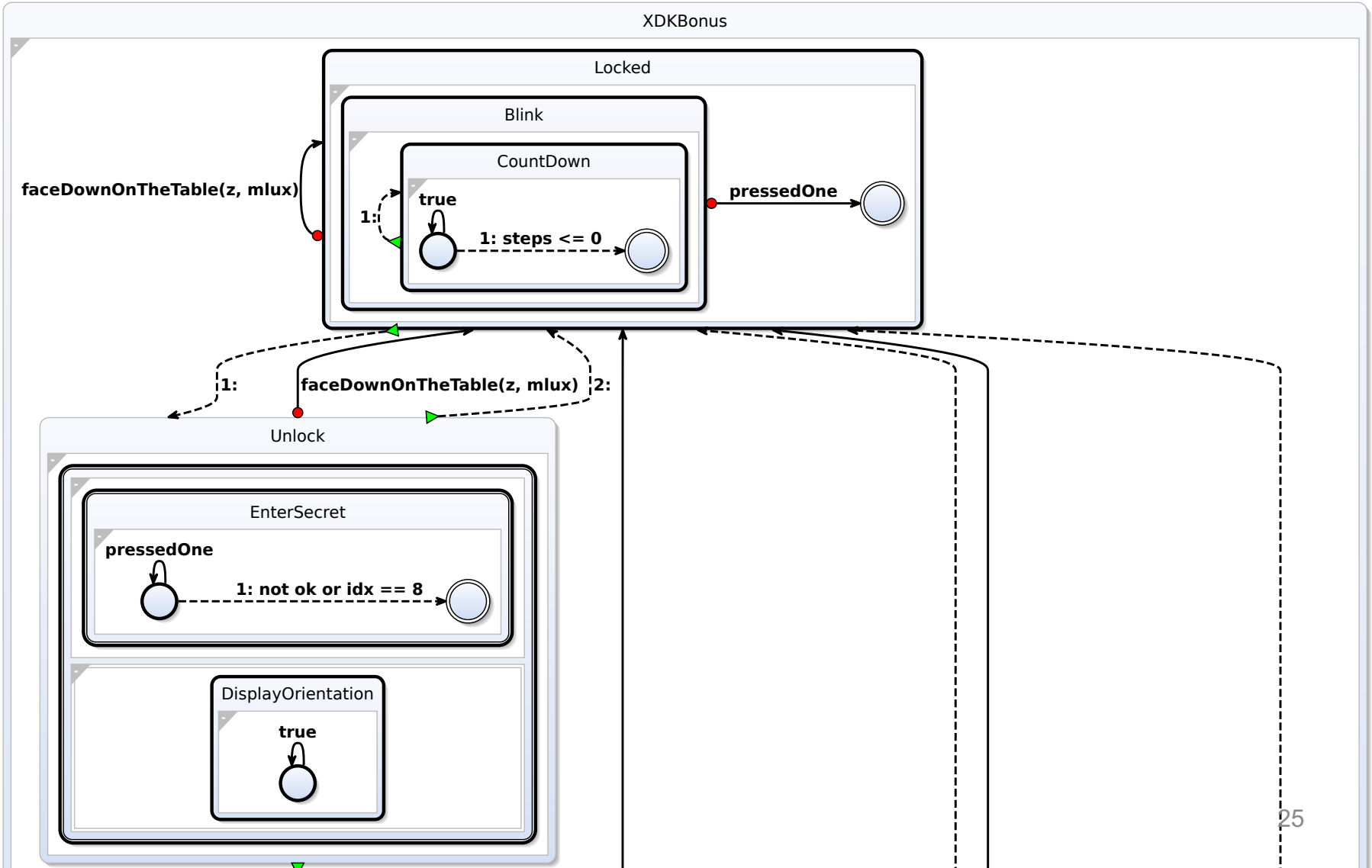
BlechCode

```
/// This is the reference implementation … 
/// initially written for the … CSE 2019 …
///
/// See accompanying lecture notes for the specification.
///
/// (c) Robert Bosch GmbH, 2019

/*************************
 * Global constants
 *************************/
const OneG: int32 = 4095 // acceleration value … 
            // consider to be at least 1g (gravitation force)
const PositionEpsilon: int32 = 400  // 10% epsilon
const Cos45xG: int32 = 2895 // cos(45°) * OneG = sin(45°) * OneG

/// We encode pose information by a prime number …
/// For example: we represent the XDK standing upright …
/// NORTH * EXACT = 2 * 11 = 22
const UNDEFPOS: nat32 = 1

const NORTH: nat32 = 2
const EAST: nat32 = 3
const SOUTH: nat32 = 5
const WEST: nat32 = 7

const EXACT: nat32 = 11
const RIGHTOF: nat32 = 13
const LEFTOF: nat32 = 17

/// The maximum length of the secret
const MAXLEN: nat32 = 8

/*************************
 * Helpers
 *************************/
/// invert LEDs' status values
function invertLEDs () (ledLeft: bool, ledMiddle: bool, ledRight: bool)
  ledRight = not ledMiddle
  ledLeft = not ledMiddle
  ledMiddle = not ledMiddle
end

/// indicate succesfully entered secret
function successToLEDs () (ledLeft: bool, ledMiddle: bool, ledRight: bool)
  ledRight = true
  ledLeft = true
  ledMiddle = true
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!