Language Engineering with Language Workbenches

Markus Voelter
independent/itemis
voelter@acm.org

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter

A DSL is a **focussed**, **processable language** for describing a specific **concern** when building a system in a specific **domain**. The **abstractions** and **notations** used are natural/suitable for the **stakeholders** who specify that particular concern.

Concepts (abstract syntax)

(concrete) Syntax

semantics (generators)

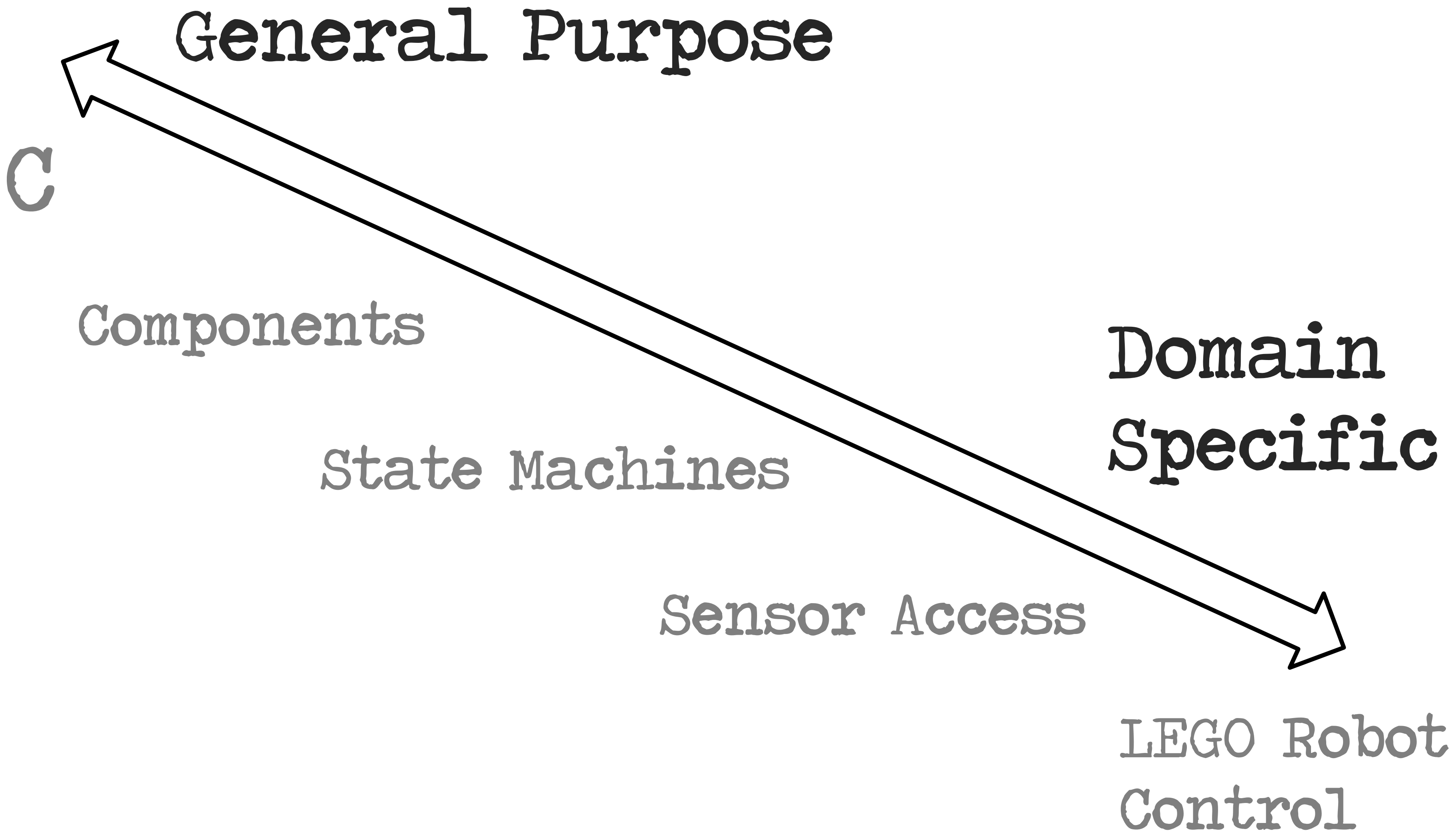
Tools and IDE

Shorter
Programs

More
Accessible
Semantics

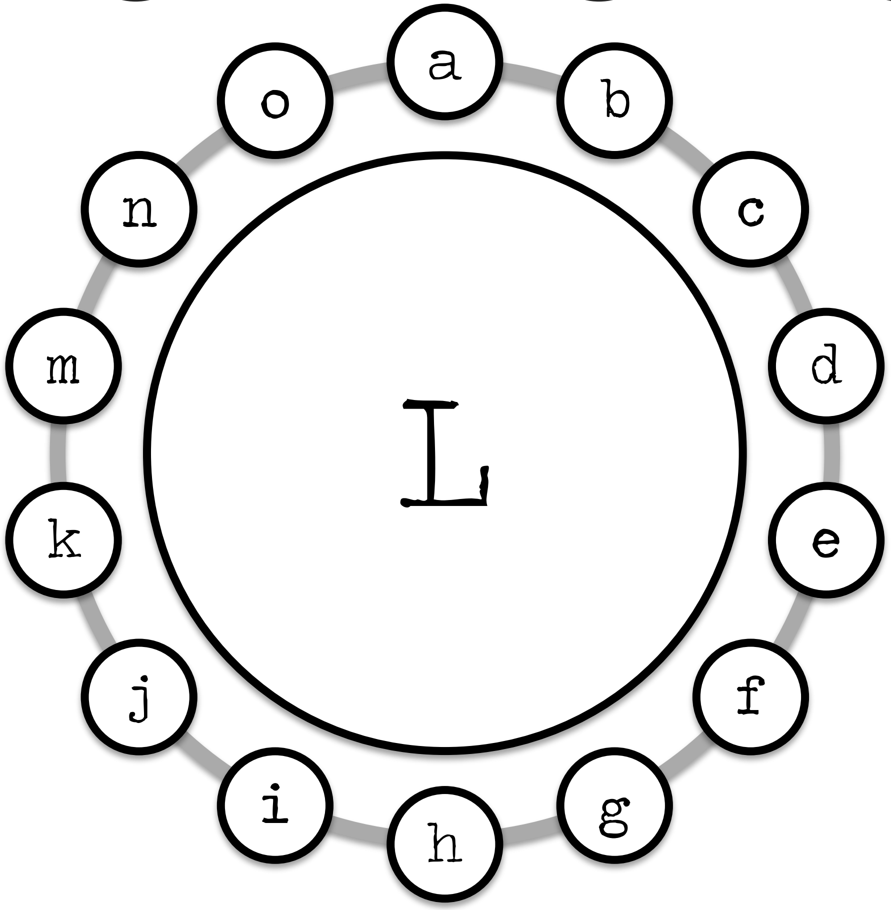
For a limited Domain!

Domain Knowledge encapsulated in language


General Purpose
C
Components
State Machines
Sensor Access
Domain Specific
LEGO Robot Control

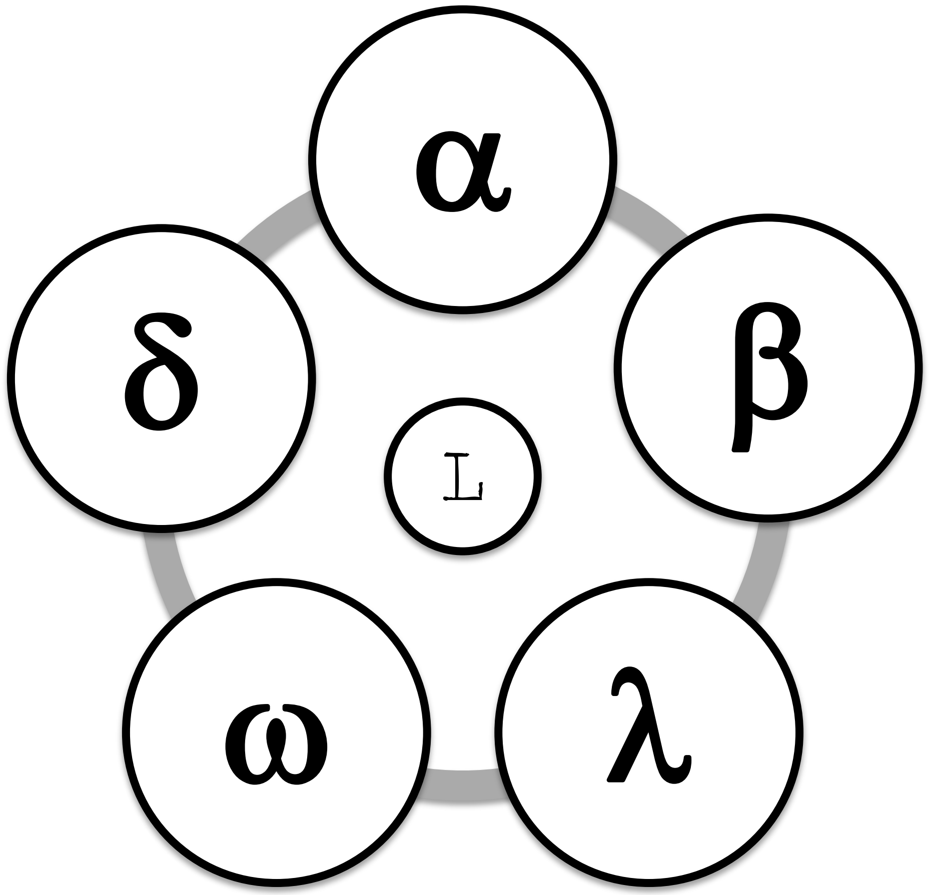
	more in GPLs	more in DSL
Domain Size	large and complex	smaller and well-defined
Designed by	guru or committee	a few engineers and domain experts
Language Size	large	small
Turing-completeness	almost always	often not
User Community	large, anonymous and widespread	small, accessible and local
In-language abstraction	sophisticated	limited
Lifespan	years to decades	months to years (driven by context)
Evolution	slow, often standardized	fast-paced
Incompatible Changes	almost impossible	feasible

Big Language



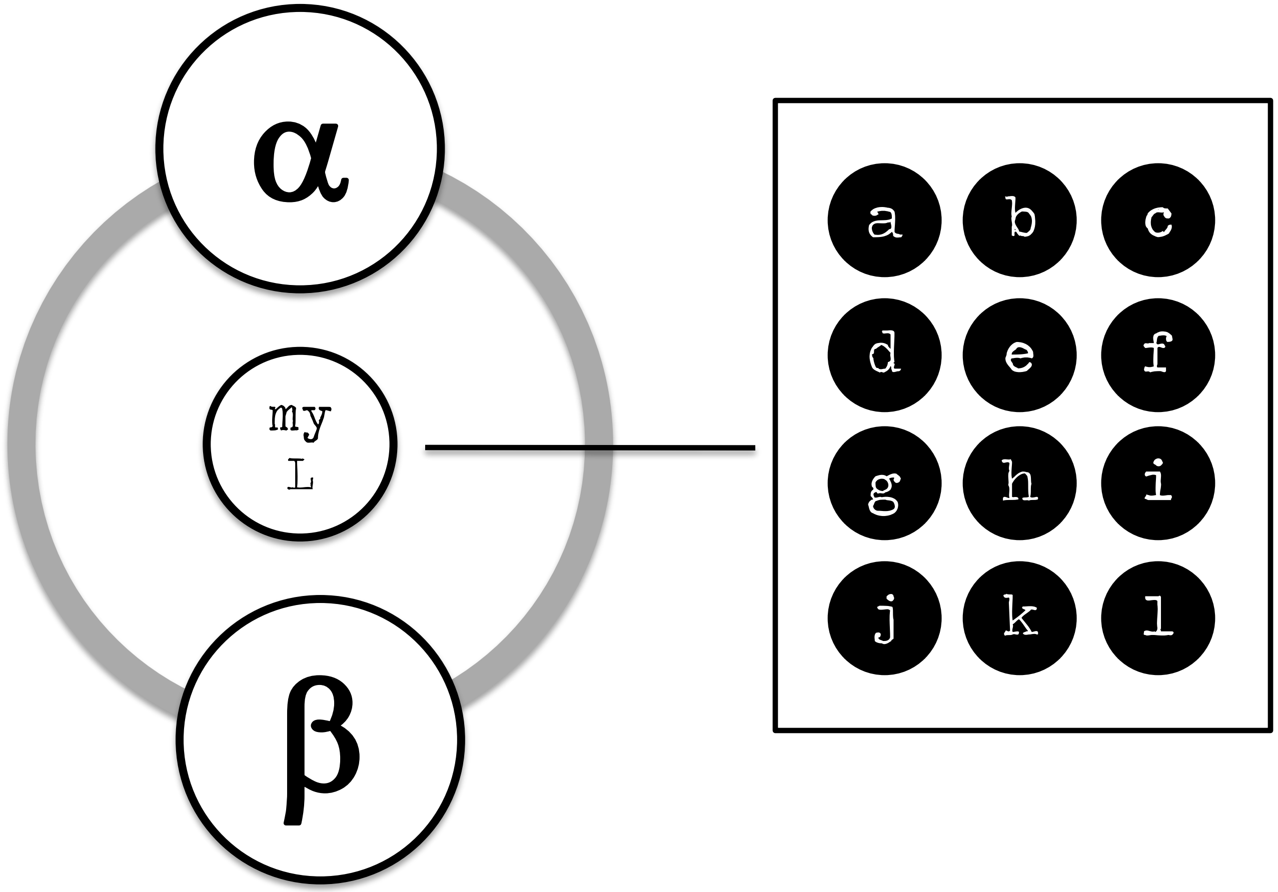
with **many** first class concepts!

Small Language



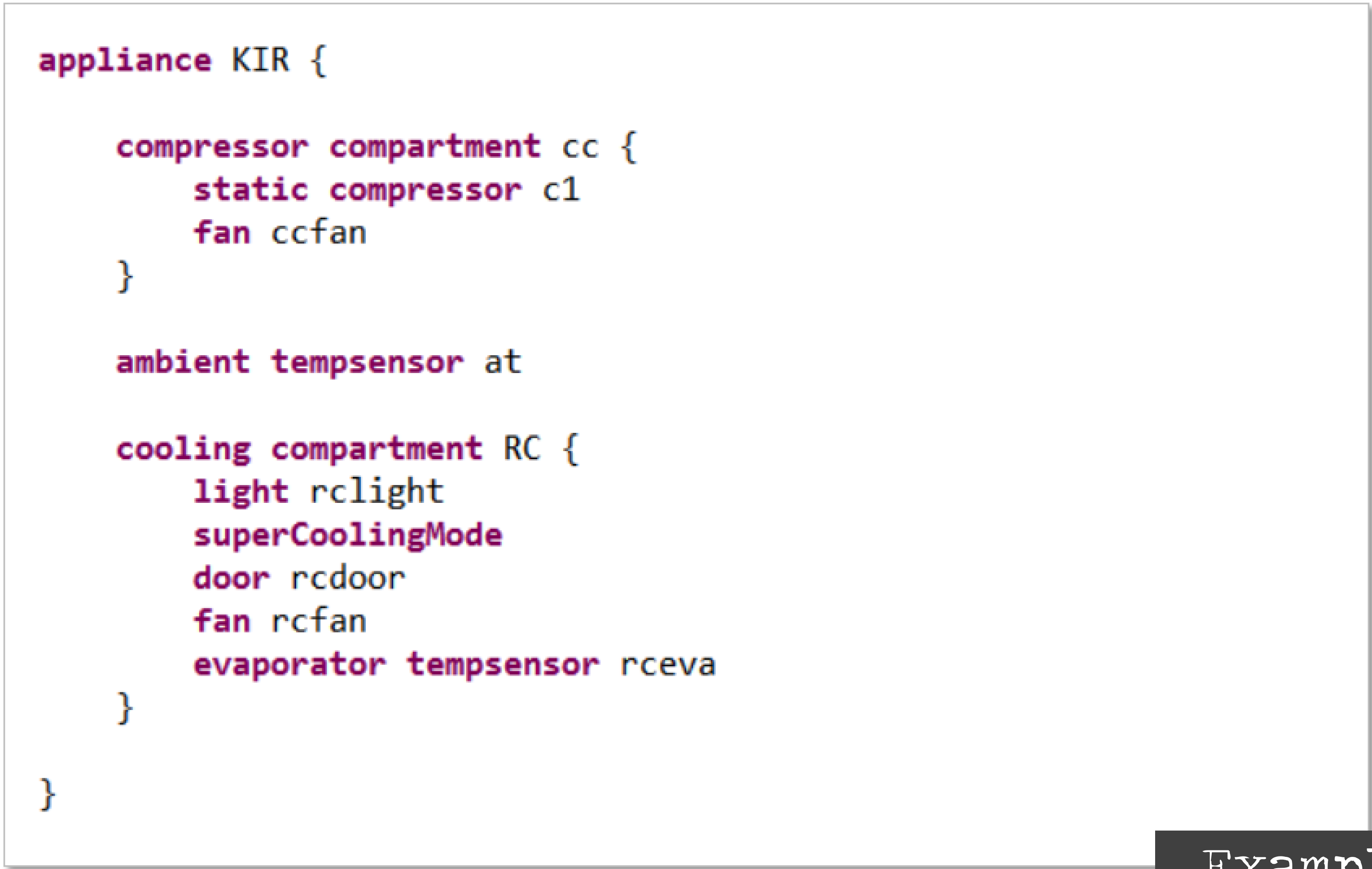
with a **few, orthogonal**
and **poweful** concepts

Modular Language



with many **optional, composable** modules

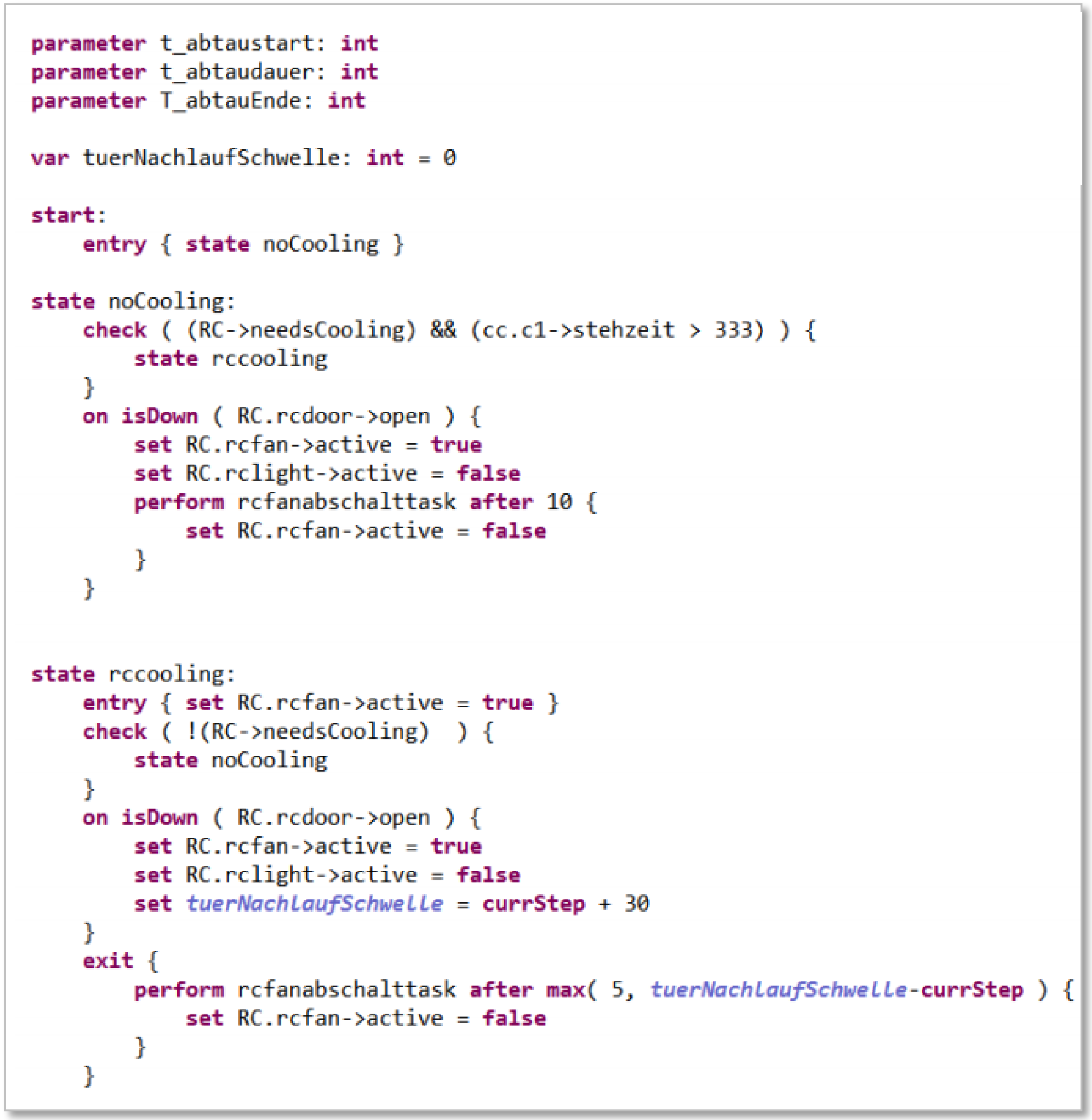
2 Examples



```
appliance KIR {

    compressor compartment cc {
        static compressor c1
        fan ccfan
    }

    ambient tempsensor at

    cooling compartment RC {
        light rclight
        superCoolingMode
        door rcdoor
        fan rcfan
        evaporator tempsensor rceva
    }

}
```

Example

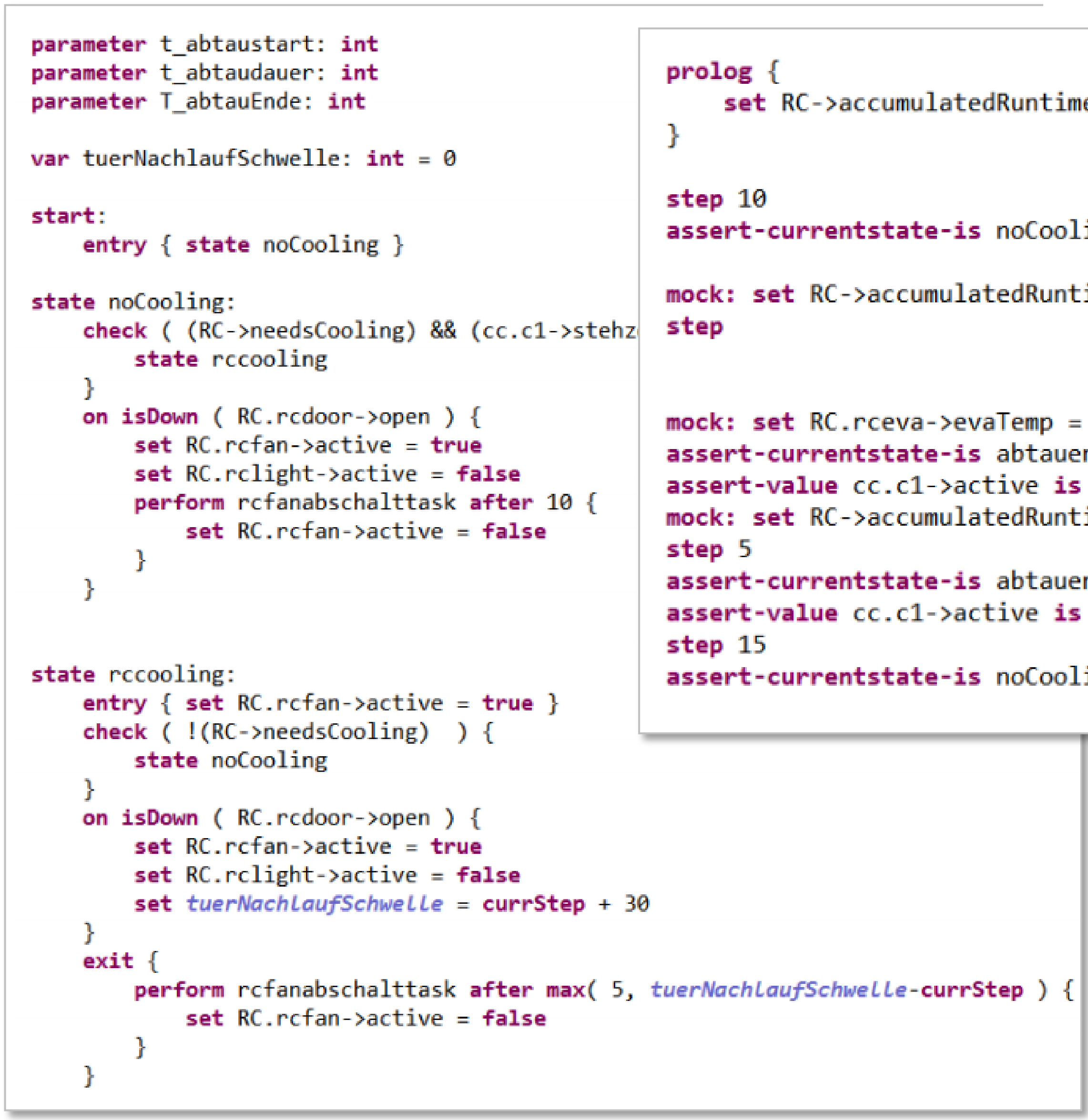
Refrige
rators



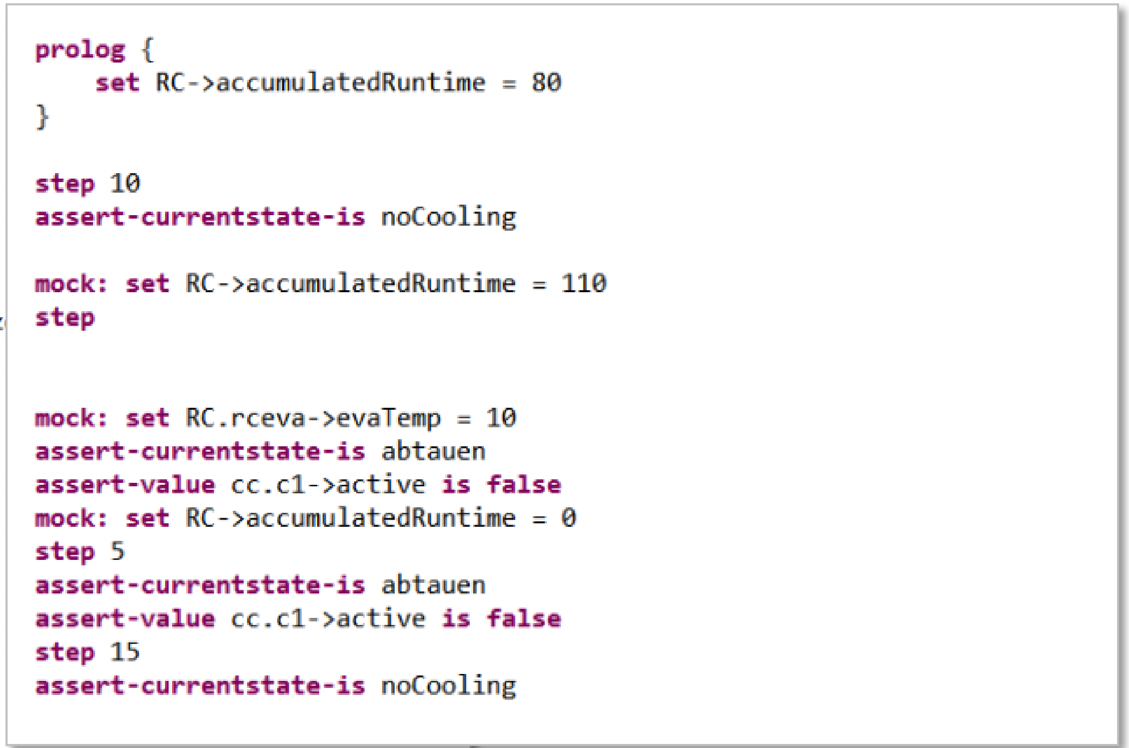
```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehzeit > 333) ) {
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

Example

Refrigerators



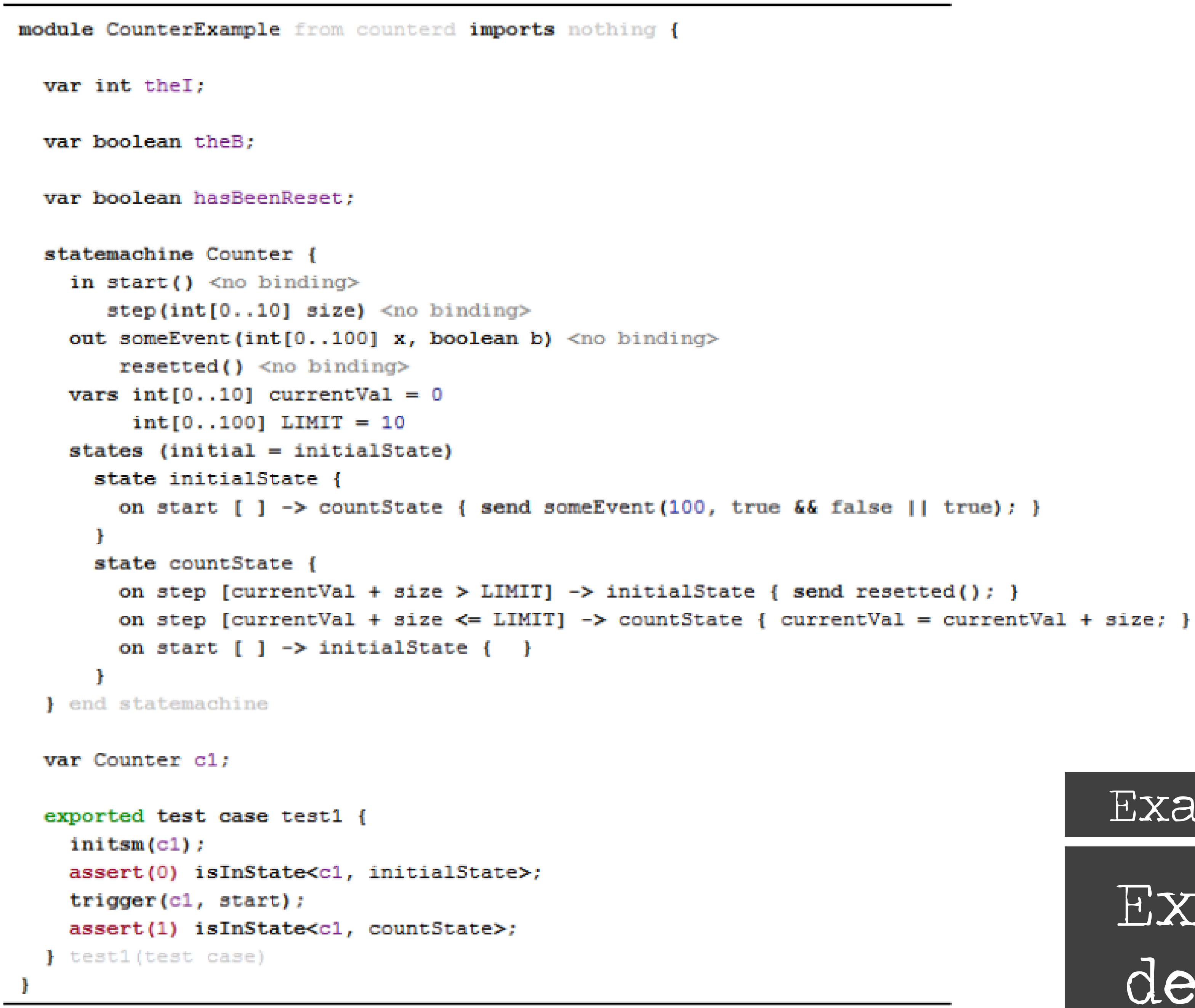
```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehz
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```



```
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauen
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtauen
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
```

Example

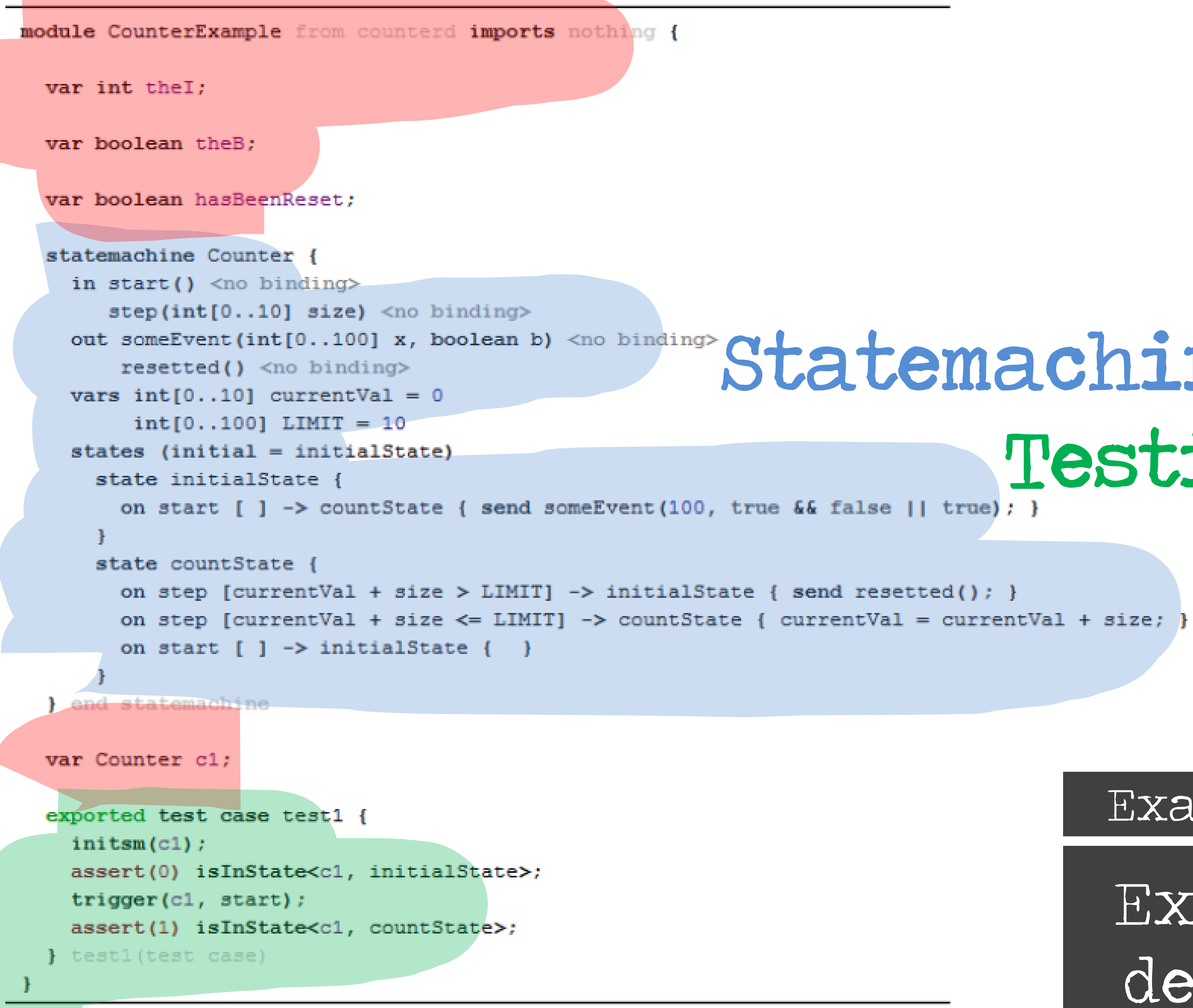
Refrigerators



```
module CounterExample from counterd imports nothing {

  var int theI;

  var boolean theB;

  var boolean hasBeenReset;

  statemachine Counter {
    in start() <no binding>
       step(int[0..10] size) <no binding>
    out someEvent(int[0..100] x, boolean b) <no binding>
        resetted() <no binding>
    vars int[0..10] currentVal = 0
         int[0..100] LIMIT = 10
    states (initial = initialState)
      state initialState {
        on start [ ] -> countState { send someEvent(100, true && false || true); }
      }
      state countState {
        on step [currentVal + size > LIMIT] -> initialState { send resetted(); }
        on step [currentVal + size <= LIMIT] -> countState { currentVal = currentVal + size; }
        on start [ ] -> initialState {   }
      }
  } end statemachine

  var Counter c1;

  exported test case test1 {
    initsm(c1);
    assert(0) isInState<c1, initialState>;
    trigger(c1, start);
    assert(1) isInState<c1, countState>;
  } test1(test case)
}
```

Example

Exten ded C



```
module CounterExample from counterd imports nothing {

  var int theI;

  var boolean theB;

  var boolean hasBeenReset;

  statemachine Counter {
    in start() <no binding>
       step(int[0..10] size) <no binding>
    out someEvent(int[0..100] x, boolean b) <no binding>
        resetted() <no binding>
    vars int[0..10] currentVal = 0
         int[0..100] LIMIT = 10
    states (initial = initialState)
      state initialState {
        on start [ ] -> countState { send someEvent(100, true && false || true); }
      }
      state countState {
        on step [currentVal + size > LIMIT] -> initialState { send resetted(); }
        on step [currentVal + size <= LIMIT] -> countState { currentVal = currentVal + size; }
        on start [ ] -> initialState {   }
      }
  } end statemachine

  var Counter c1;

  exported test case test1 {
    initsm(c1);
    assert(0) isInState<c1, initialState>;
    trigger(c1, start);
    assert(1) isInState<c1, countState>;
  } test1(test case)
}
```

C

Statemachines

Testing

Example

Extended C

2 Tools

Xtext

MPS

Meta Programming System

DEMO

The End.

This material is part of my upcoming (early 2013) book **DSL Engineering**. Stay in touch, it may become a free eBook ☺

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter