

DSL Design

A conceptual framework
for building good DSLs

Markus Voelter
independent/itemis
voelter@acm.org

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

based on material
from a paper written
with Eelco Visser

E.Visser@tudelft.nl
<http://eelcovisser.org/>

DSL Engineering

*Designing, Implementing and Using
Domain-Specific Languages*



Markus Voelter

with Sebastian Benz
Christian Dietrich
Birgit Engelmann
Mats Helander
Lennart Kats
Eelco Visser
Guido Wachsmuth

www.dslbook.org

This material is part of my
upcoming (Feb. 2013) book

DSL Engineering

Designing, Implementing and Using
Domain-Specific Languages

Stay in touch; it will be cheap :-)

<http://dslbook.org>

Introduction

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.

General Purpose

C

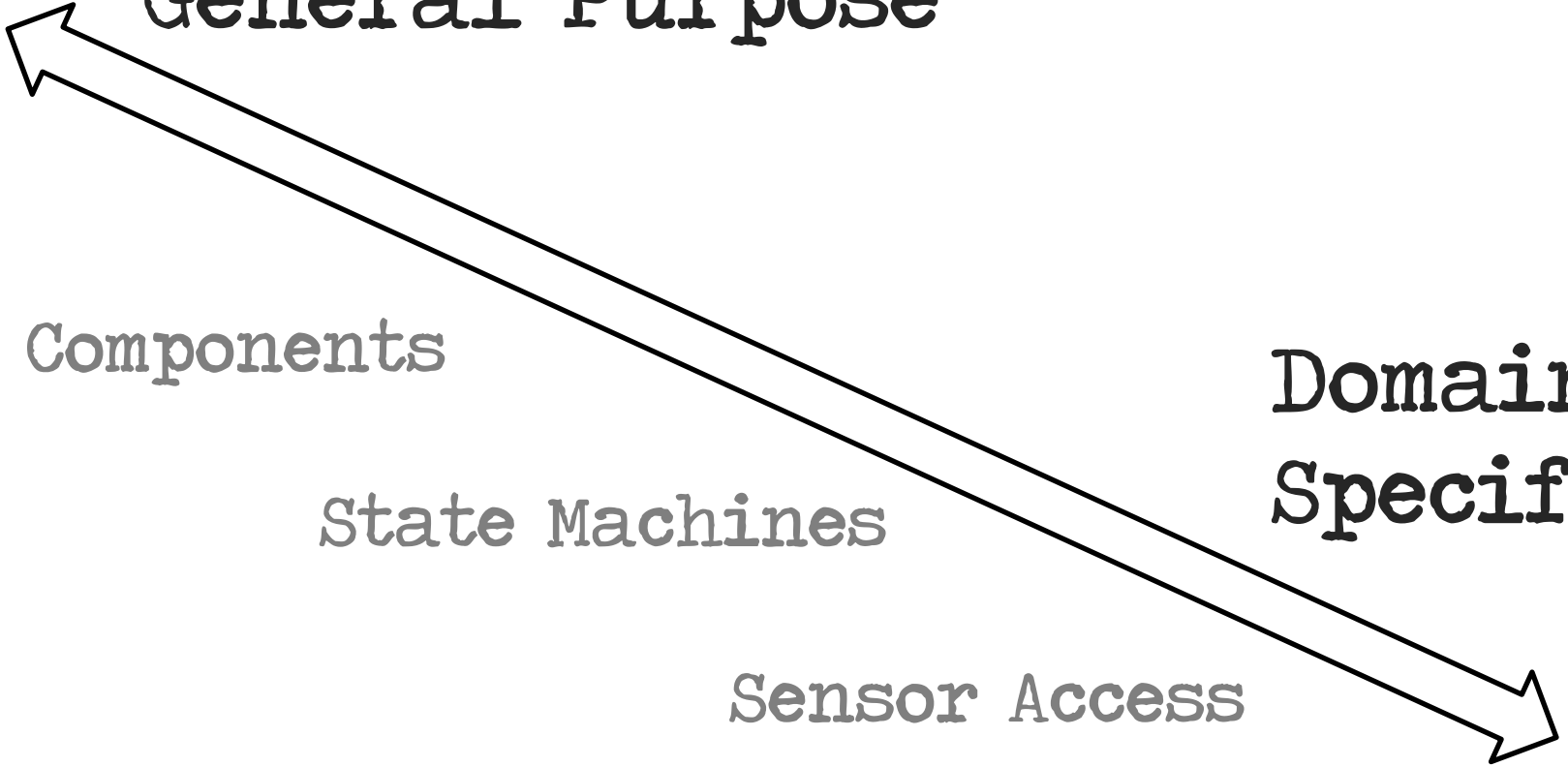
Components

State Machines

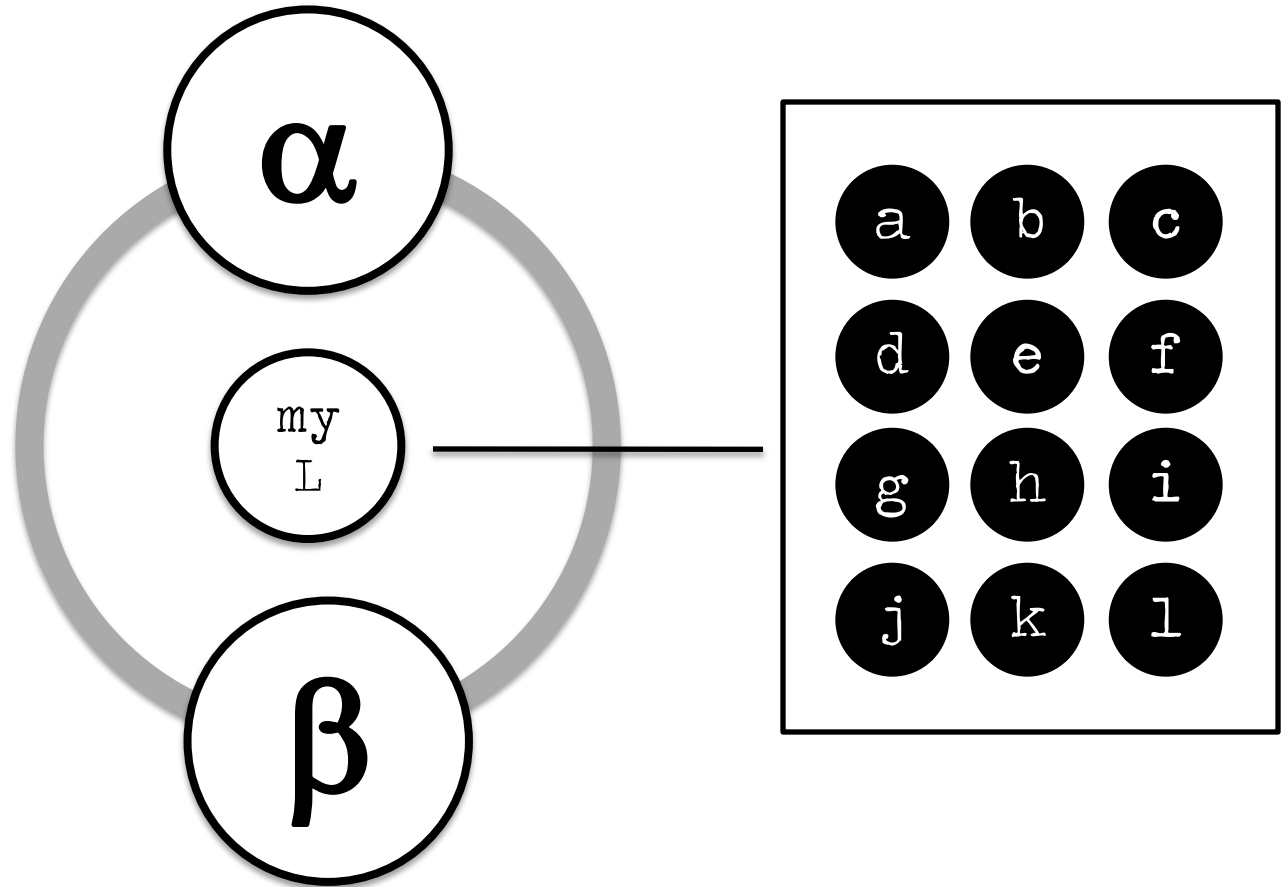
Sensor Access

**Domain
Specific**

LEGO Robot
Control



Modular Language



with many optional,
composable modules

Case Studies

```
namespace com.mycompany {
  namespace datacenter {
    component DelayCalculator {
      provides aircraft: IAircraftStatus
      provides console: IManagementConsole
      requires screens[0..n]: IInfoScreen
    }
    component Manager {
      requires backend[1]: IManagementConsole
    }
    public interface IInfoScreen {
      message expectedAircraftArrivalUpdate
        (id: ID, time: Time)
      message flightCancelled(flightID: ID)
    }
    public interface IAircraftStatus ...
    public interface IManagementConsole ...
  }
}
```

Example

Compo
nents

```
namespace com.mycompany.test {  
  system testSystem {  
    instance dc: DelayCalculator  
    instance screen1: InfoScreen  
    instance screen2: InfoScreen  
    connect dc.screens to  
      (screen1.default, screen2.default)  
  }  
}
```

Example

Compo
nents

```
appliance KIR {  
  
    compressor compartment cc {  
        static compressor c1  
        fan ccfan  
    }  
  
    ambient tempensor at  
  
    cooling compartment RC {  
        light rclight  
        superCoolingMode  
        door rcdoor  
        fan rcfan  
        evaporator tempensor 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

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->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

Refrige
rators

```
module main imports OsekKernel, EcAPI, BitLevelUtilities {
```

```
    constant int WHITE = 500;
```

```
    constant int BLACK = 700;
```

```
    constant int SLOW = 20;
```

```
    constant int FAST = 40;
```

```
    statemachine linefollower {
```

```
        event initialized;
```

```
        initial state initializing {
```

```
            initialized [true] -> running
```

```
        }
```

```
        state running { }
```

```
    }
```

```
    initialize {
```

```
        ecrobot_set_light_sensor_active
```

```
            (SENSOR_PORT_T::NXT_PORT_S1);
```

```
        event linefollower:initialized
```

```
    }
```

```
    terminate {
```

```
        ecrobot_set_light_sensor_inactive
```

```
            (SENSOR_PORT_T::NXT_PORT_S1);
```

```
    task run cyclic prio = 1 every = 2 {
```

```
        stateswitch linefollower
```

```
            state running
```

```
                int32 light = 0;
```

```
                light = ecrobot_get_light_sensor
```

```
                    (SENSOR_PORT_T::NXT_PORT_S1);
```

```
                if ( light < ( WHITE + BLACK ) / 2 ) {
```

```
                    updateMotorSettings(SLOW, FAST);
```

```
                } else {
```

```
                    updateMotorSettings(FAST, SLOW);
```

```
                }
```

```
            default
```

```
                <noop>;
```

```
        }
```

```
    void updateMotorSettings( int left, int right ) {
```

```
        nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_C, left, 1);
```

```
        nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, right, 1);
```

```
    }
```

Example

Extended C

Capgemini Pension Workbench

File Edit Projection Navigation Search Format Tools Dev Generate Pension Team NN

NNLCPA-14w2-21112008 * x

Table of Contents * x

Library

- Documentation
- Foundation
 - Value sets
 - Value set Groottebepalingmethode
 - Value set member Salaris-diensttijd
 - Value set member Verzekerde bedragen
 - Value set member Afgeleide toezegging
 - Value set Salaris-diensttijd
 - Value set member Middelloon
 - Value set member Eindloon
 - Value set Verzekerde bedragen
 - Value set member Vast bedrag
 - Value set member Percentage van gron
 - Value set member Percentage van gron
 - Value set member Opgegeven bedrag
 - Value set member ANW-hiaat
 - Value set member AOP bedrag
 - Value set Indicatie Opbouw / Risico
 - Value set member Opbouw
 - Value set member Risico
 - Value set Deelnemerstatus
 - Value set member Aspirant
 - Value set member Actief
 - Value set member Premievrij
 - Value set member Slapend
 - Value set member Uitkerend
 - Value set member Overleden
 - Value set member Vervallen
 - Tag definitions
 - Tag Basisberekening
 - Tag Ouderdomspensioen
 - Tag Partnerpensioen
 - Tag Wezenpensioen
 - Tag ANW extra
 - Tag WIA excedent AOV

All

3.3 Commutatietgetallen op 1 leven¶

$$D_x = v^x * \frac{l_x}{100} \quad \text{¶6 Dec (3) ¶}$$

Implemented in ¶ [V940](#) ¶

$$\omega - x$$

$$N_x = \sum_{t=0}^{\omega - x} D_{x+t} \quad \text{¶7 Dec (3) ¶}$$

3.6 Contante waarde 1 leven/ 2 levens¶

$$E_{n|x} = \frac{D_{x+n}}{D_x} \quad \text{¶19 Dec (4) ¶}$$

$$a_x = \ddot{a}_x - 1 \quad \text{¶21 Dec (3) ¶}$$

$$\bar{a}_x = \ddot{a}_x - 0,5 \quad \text{¶22 Dec (3) ¶}$$

$$\ddot{a}_{x:n]} = \frac{N_x - N_{x+n}}{D_x} \quad \text{¶23 Dec (3) ¶}$$

$$\bar{a}_{x:n]} = \ddot{a}_{x:n]} - 0,5 + 0,5 * E_{n|x} \quad \text{¶25 Dec (3) ¶}$$

4 BN(_ris) koopsommen¶

Section • title • Paragraph : Text Dev
Doc | Splitter | Pension | PensionDecorated | IAM

Example

Pension
Plans

Cappgemini Pension Workbench

File Edit Projection Navigation Search Format Tools Dev Generate Pension Team NN

NNLCPA-14w2-21112008 * x

Table of Contents x

All

Library

- Documentation
- Foundation
 - Value sets...
 - Tag definitions
 - Tag Basisberekening
 - Tag Ouderdomspensioen
 - Tag Partnerpensioen
 - Tag Wezenpensioen
 - Tag ANW extra
 - Tag WIA exceedent AOV
 - Tag Eindkapitaal
- Shared
 - Elements...
 - Rules
 - Rule Bereken Mutatieperiode**
 - Rule Bereken Salaris ontwikkeling
 - Rule Bereken Pensioengrondslag
 - Rule Bereken Pensioengrondslag
 - Rule Bereken Bedrag jaaropbouw
 - Rule Bereken Bedrag jaaropbouw
 - Rule Bereken Bedrag jaaropbouw
 - Rule Bereken Delta deelaanspraak uit door
 - Rule Bereken Deelaanspraak opgebouwd
 - Rule Bereken Toekomstige dienstjaren
 - Rule Bereken Deelaanspraak uitzicht
 - Rule Bereken Verzekerd bedrag
 - Rule Bereken Verzekerd bedrag
 - Rule Bereken Verzekerd bedrag
 - Rule Bereken Verkoopkosten
 - Rule Bereken Netto Eindwaarde
 - Rule Bereken einddatum opbouw
 - Rule Bereken premie
 - Rule Bereken IS-Opslag
 - Rule Bereken administratiekosten

Elements...

Rules

Rule Bereken Mutatieperiode

Result:

Mutatieperiode

Name:

Bereken Mutatieperiode

Documentation:

Het vaststellen van de periode tussen de huidige en de vorige mutatie in dagen.

De mutatieperiode kan niet meer dan 360 dagen bedragen omdat elk jaar een begin- en eindmutatie kent i.v.m. het openen en sluiten van het verslagjaar.

Dit wordt niet afgevangen omdat het uitvoeren van de begin- en eindmutatie verantwoordelijkheid zijn van de pensioenadministratie.

Tags:

Basisberekening

Algorithm:

if maximum(Mutaties per datum) == 1 then daysof(duration(valid(Mutaties per datum))) else 0

Test cases:

| Name | Valid time | Transaction time | Fixture | Product | Element | Expected value | Actual value |
|----------------|------------|------------------|---|---------|---------|----------------|--------------|
| Gelijke datums | 03/01/2008 | | Mutatieperiode - Mutatiedatum = Mutatiedatum Vorig | | | 3 | 0 |
| Periode < 30 | 03/01/2008 | | Mutatieperiode - Mutatiedatum > Mutatiedatum Vorig (binnen 1 maand) | | | 15 | 15 |
| Periode > 30 | 03/01/2008 | | Mutatieperiode - Mutatiedatum > Mutatiedatum Vorig (meerdere maanden) | | | 60 | 60 |

Bereken Mutatieperiode • Test cases • Unit test: Gelijke datums : ^Place Dev

Doc | Splitter | Pension | PensionDecorated | AM

Example

Pension
Plans

```

entity Post {
  key      :: String (id)
  blog     → Blog
  urlTitle :: String
  title    :: String (searchable)
  content  :: WikiText (searchable)
  public   :: Bool (default=false)
  authors  → Set<User>
  function isAuthor(): Bool {
    return principal() in authors
  }
  function mayEdit(): Bool {
    return isAuthor();
  }
  function mayView(): Bool {
    return public || mayEdit();
  }
}

```

access control rules

```

  rule page post(p: Post, title: String) {
    p.mayView()
  }
  rule template newPost(b: Blog) {
    b.isAuthor()
  }
section posts
  define page post(p: Post, title: String) {
    title{ output(p.title) }
    bloglayout(p.blog){
      placeholder view { postView(p) }
      postComments(p)
    }
  }
  define permalink(p: Post) {
    navigate post(p, p.urlTitle) { elements }
  }

```

Example

Web
DSL

Terms
&
Concepts

Domain

deductive
top down



body of
knowledge
in the real
world

Domain

Example

Example

Penion
Plans

Refrige
rators

Domain

existing
software
(family)



inductive
bottom up

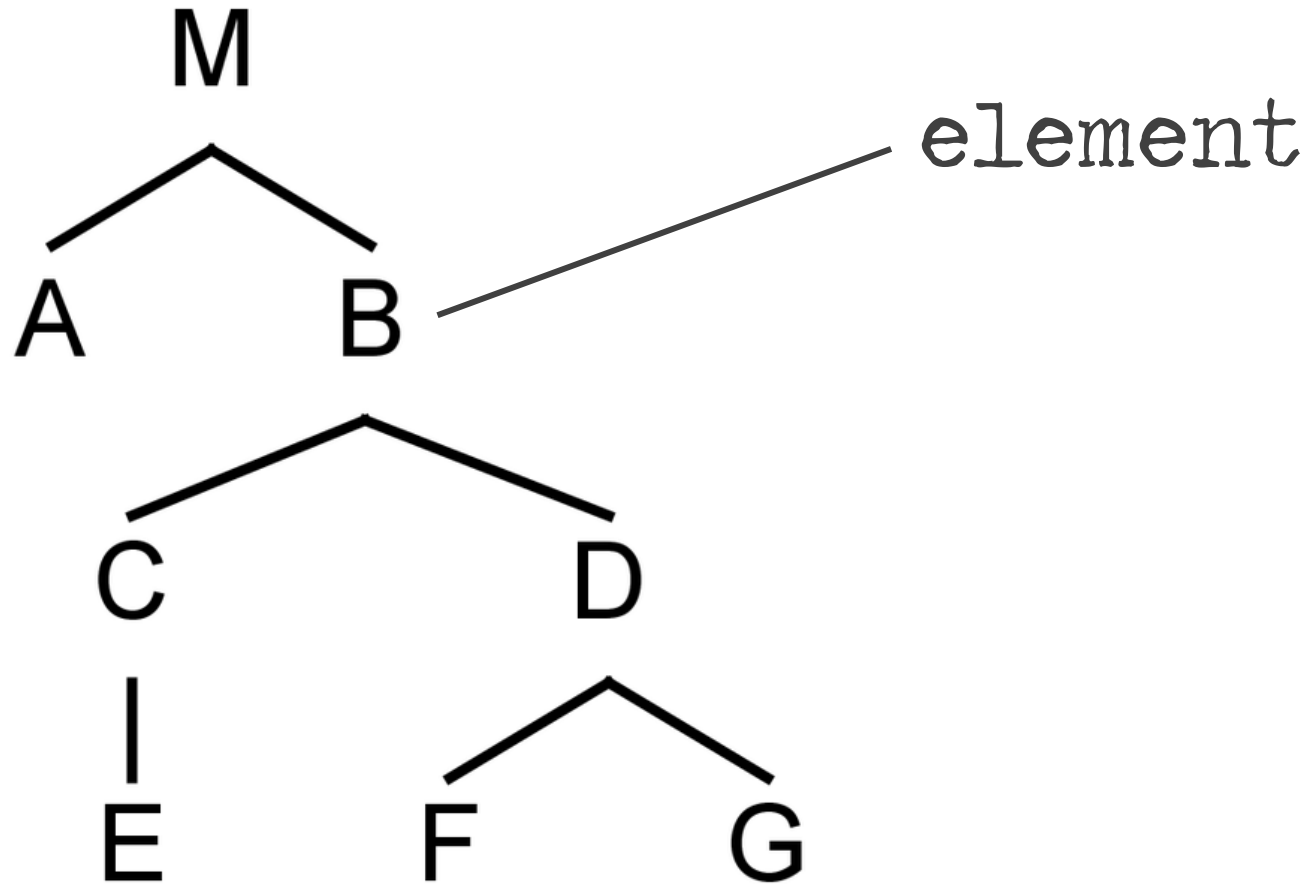
Example

Example

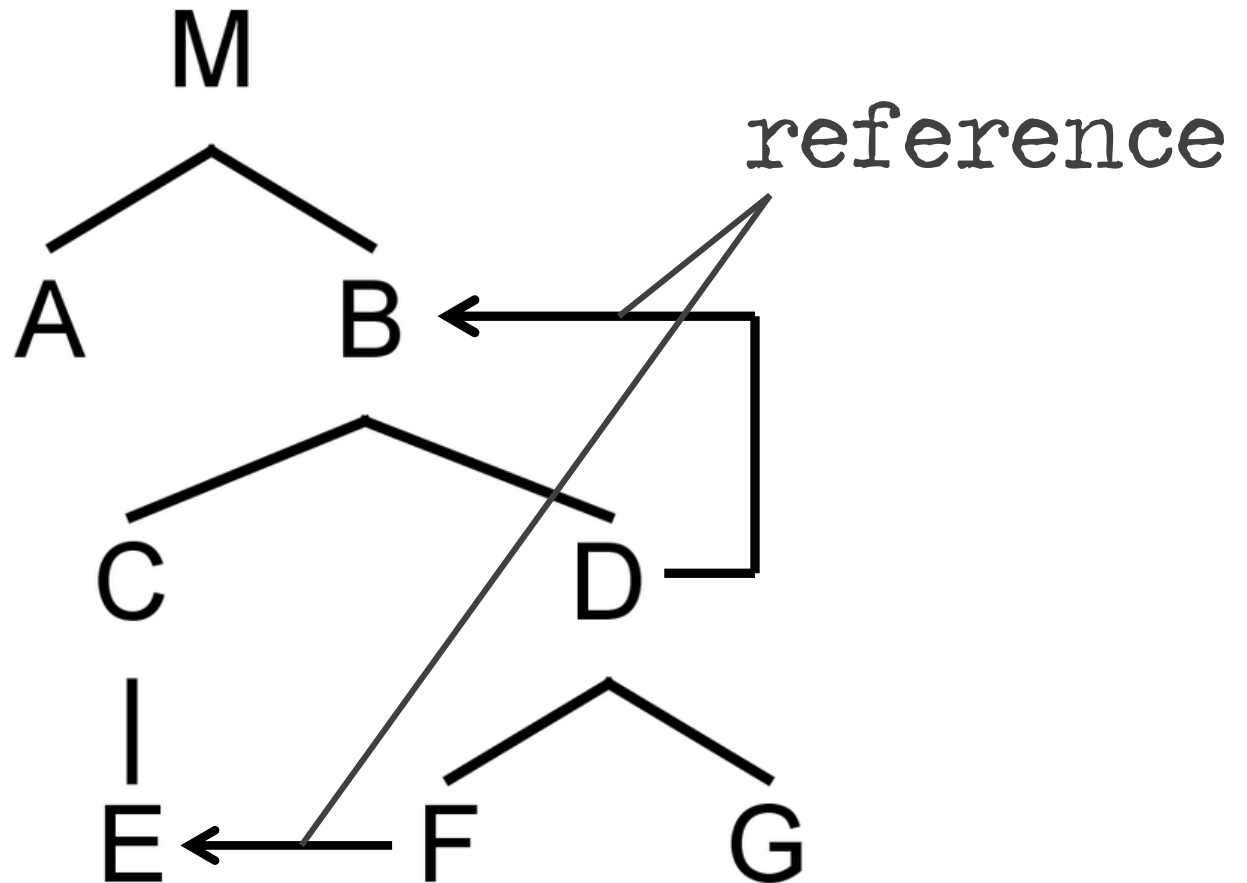
Exten
ded C

Compo
netns

Programs
are trees



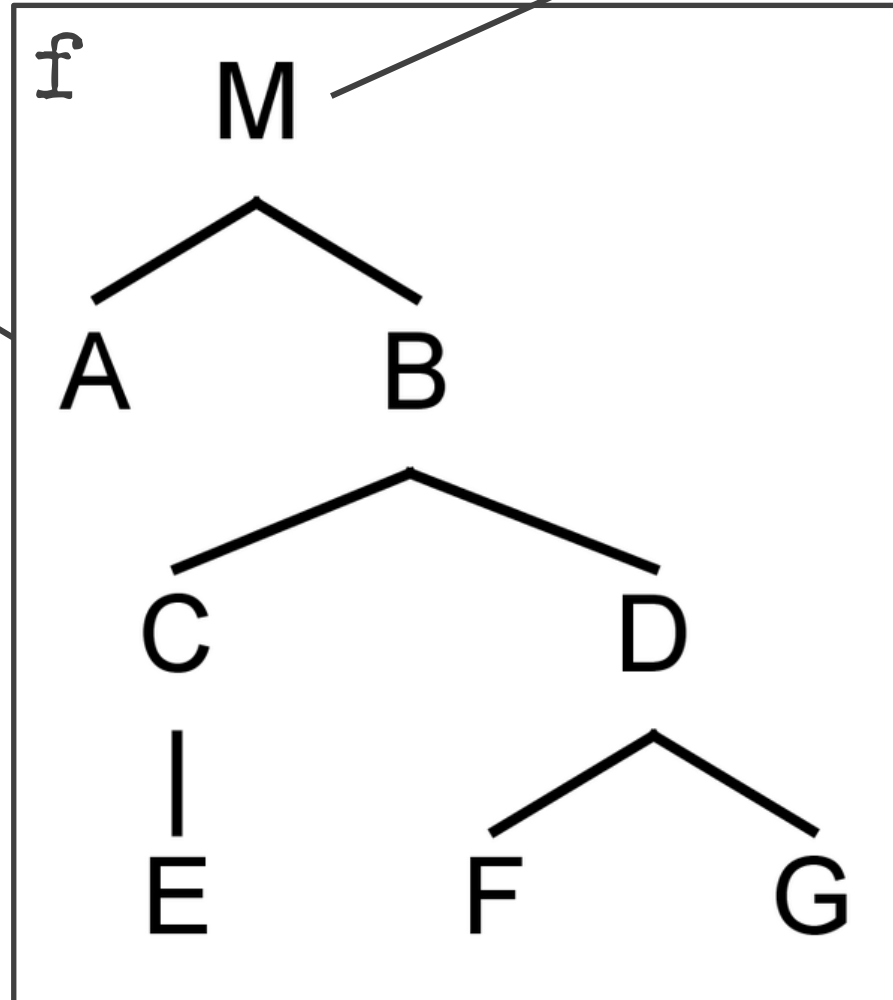
Programs are
graphs, really.



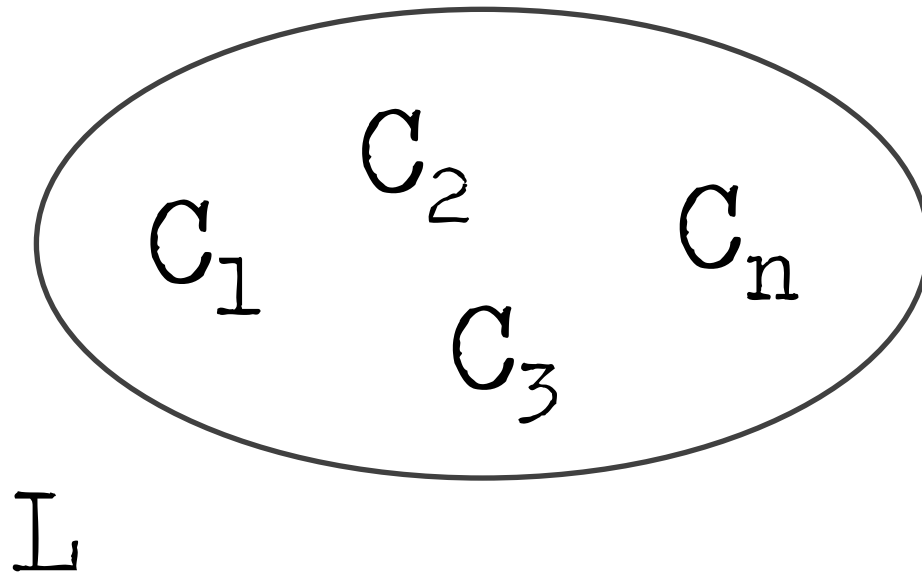
Fragments are
subtrees w/ root

fragment root

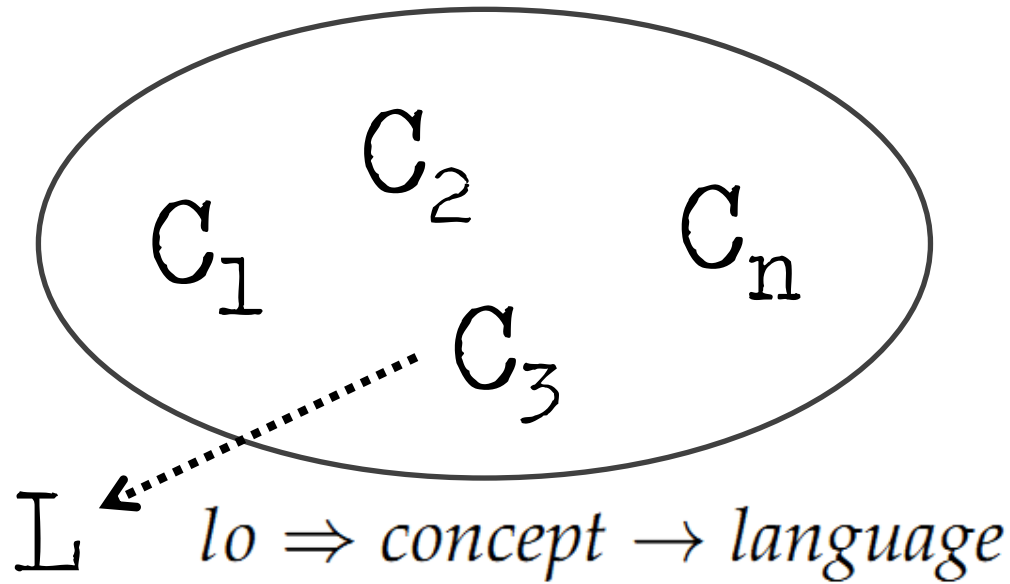
fragment



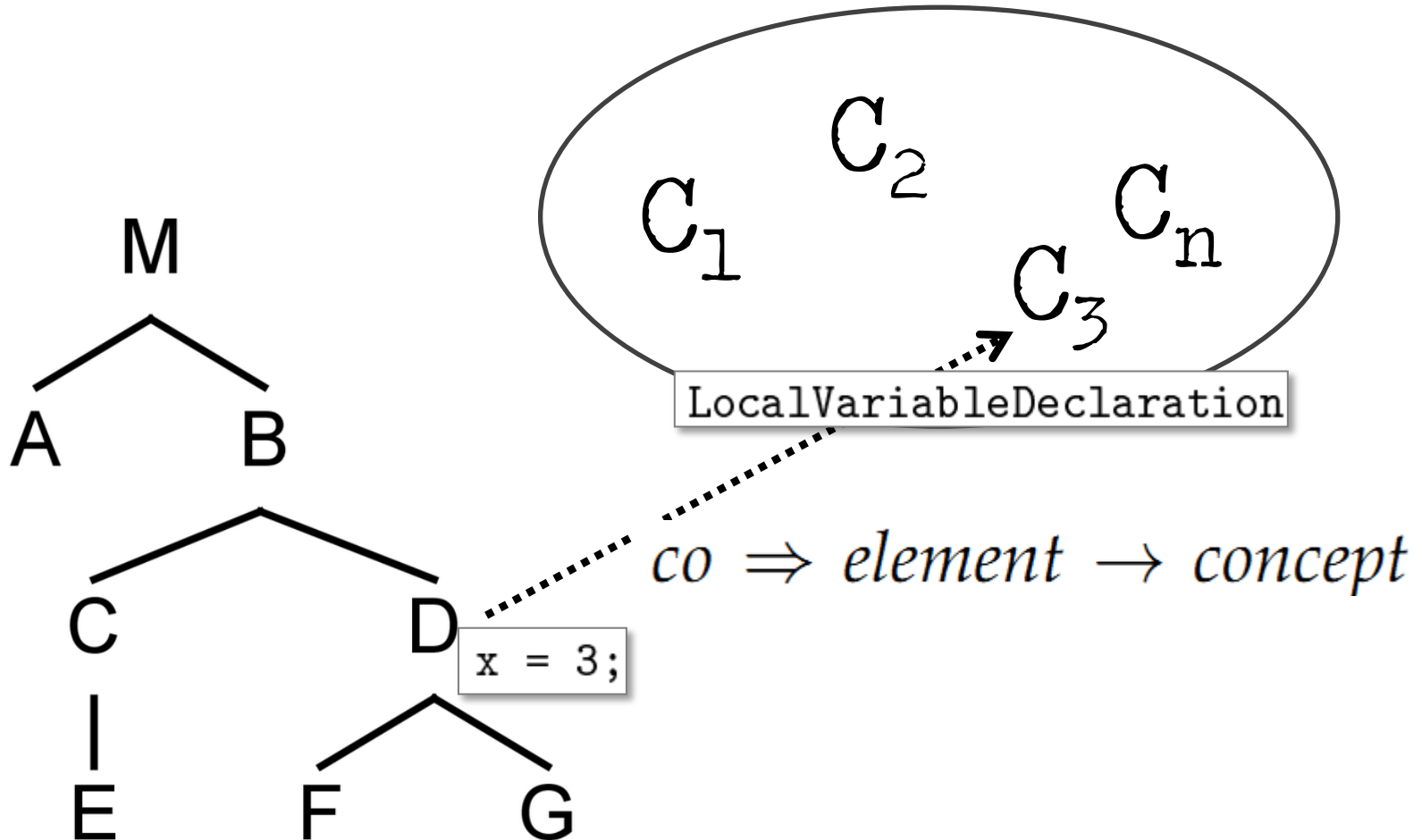
Languages are
sets of concepts



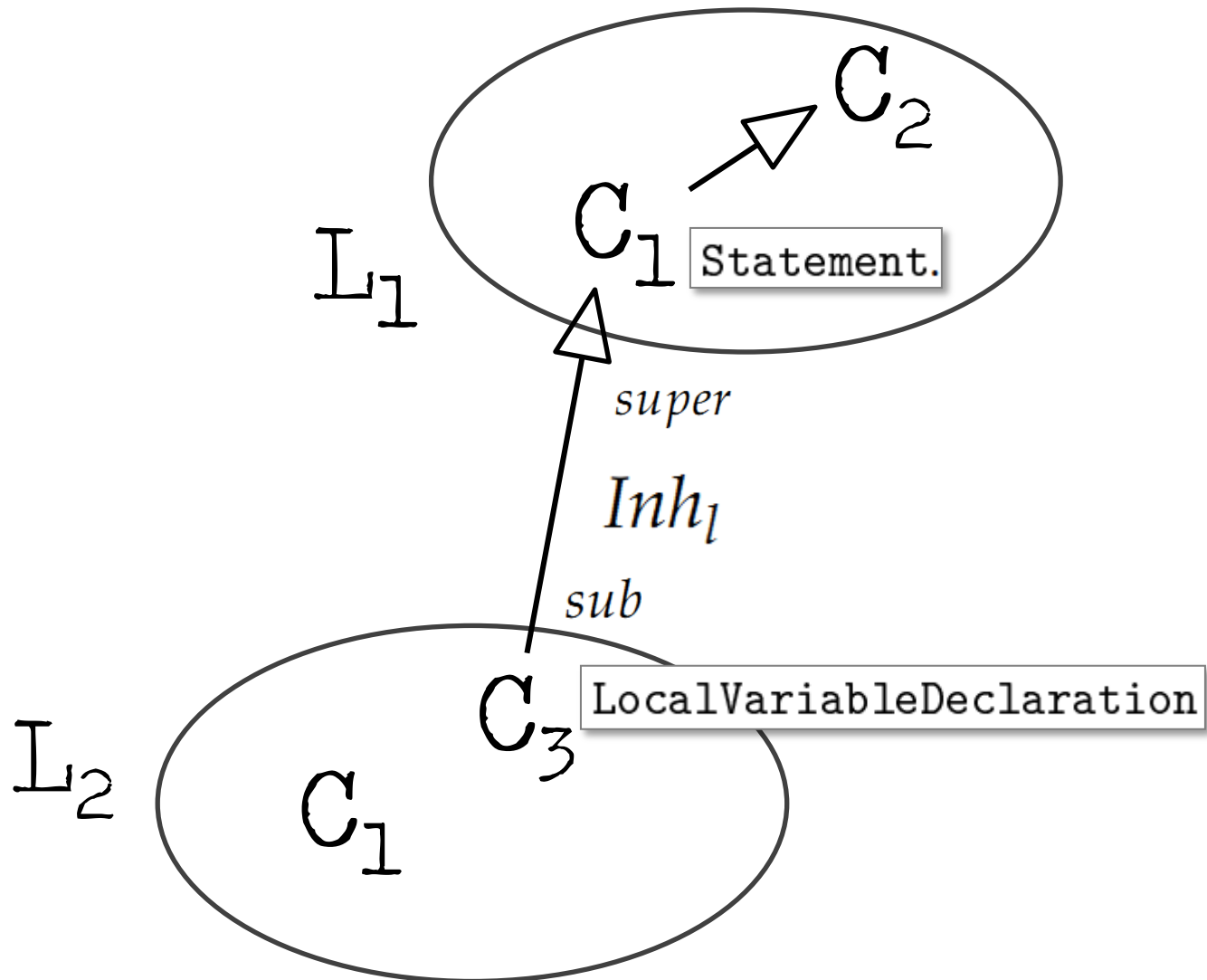
Languages are
sets of concepts



Programs and languages



Language: concept inheritance



Language

does not depend on
any other language

$$\forall r \in Refs_l \mid lo(r.to) = lo(r.from) = l$$

$$\forall s \in Inh_l \mid lo(s.super) = lo(s.sub) = l$$

$$\forall c \in Cdn_l \mid lo(c.parent) = lo(c.child) = l$$

Independence

Fragment

does not depend on
any other fragment

$$\forall r \in Refs_f \mid fo(r.to) = fo(r.from) = f$$

$$\forall e \in E_f \mid lo(co(e)) = l$$

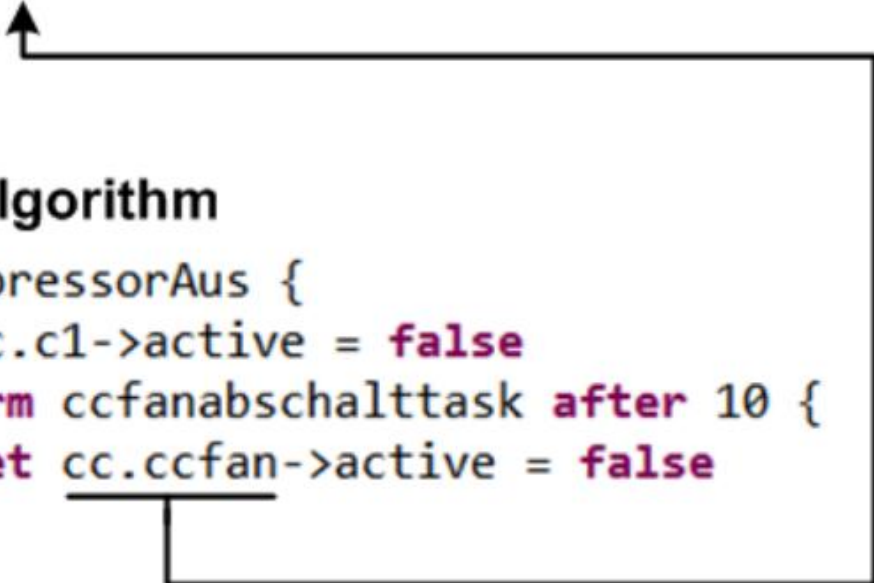
Independence

Hardware:

```
compressor compartment cc {  
  static compressor c1  
  fan ccfan  
}
```

Cooling Algorithm

```
macro kompressorAus {  
  set cc.c1->active = false  
  perform ccfanabschalttask after 10 {  
    set cc.ccfan->active = false  
  }  
}
```



Example

Refrige
rators

Homogeneous

Fragment

everything expressed
with one language

$$\forall e \in E_f \mid lo(e) = l$$

$$\forall c \in Cdn_f \mid lo(c.parent) = lo(c.child) = l$$

```
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 {
```

```
    initism(c1);
```

```
    assert(0) isInState<c1, initialState>;
```

```
    trigger(c1, start);
```

```
    assert(1) isInState<c1, countState>;
```

```
  } test1(test case)
```

```
}
```

Heterogeneous

C

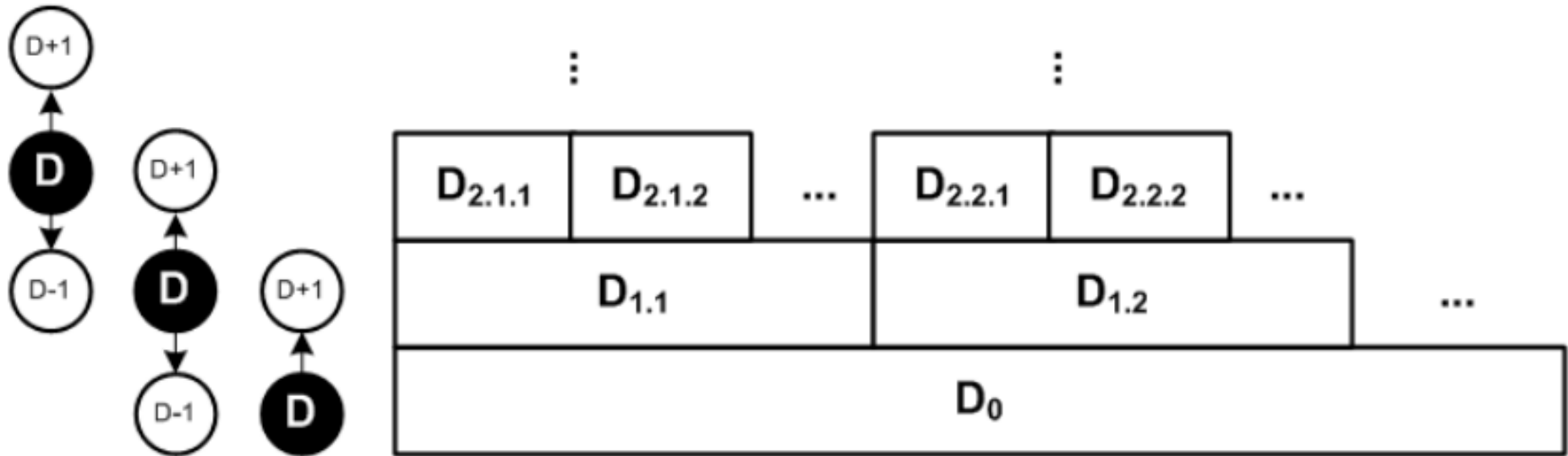
Statemachines

Testing

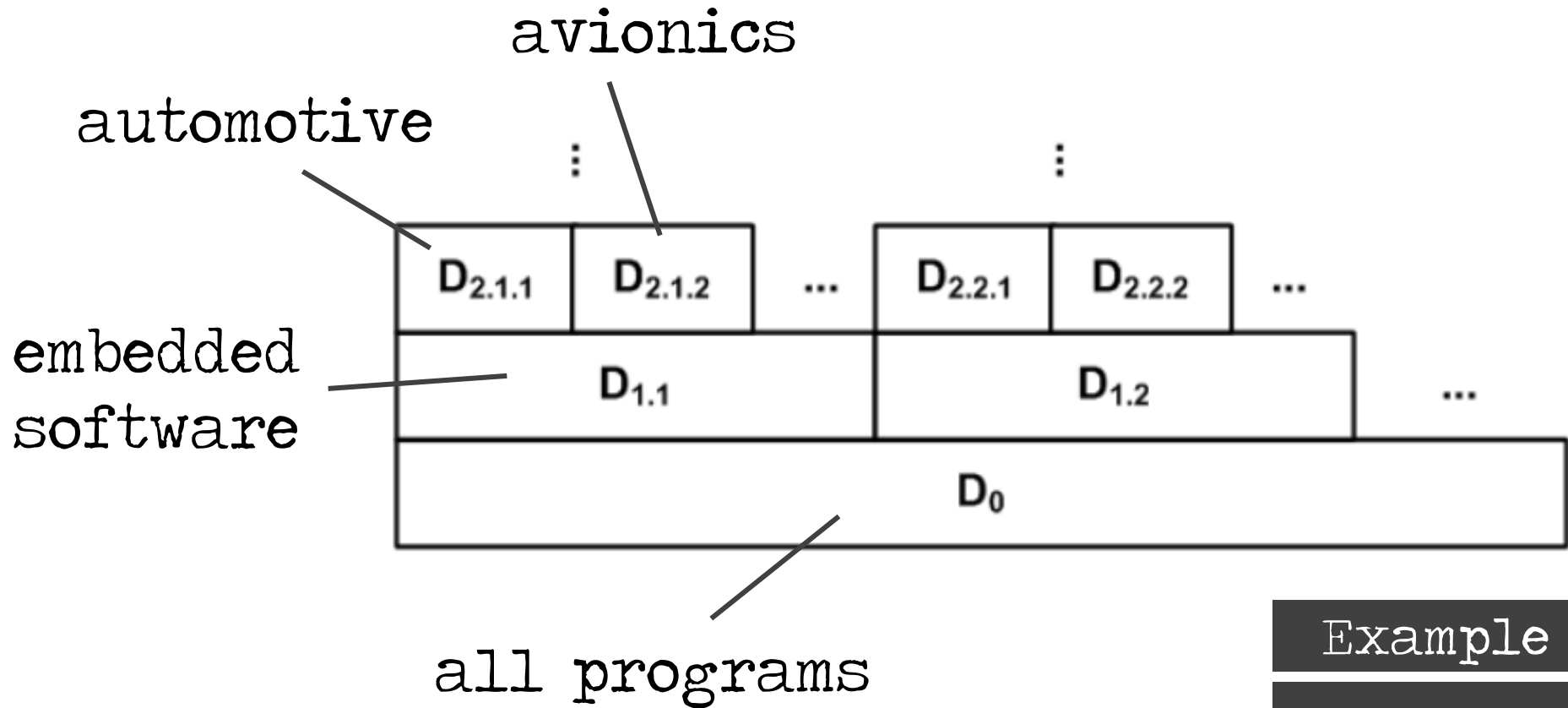
Example

Extended C

Domain Hierarchy



Domain Hierarchy



Example

Extended C

Design Dimensions

expressivity
coverage
semantics
separation of
concerns

completeness
paradigms
modularity
concrete
syntax

process

Expressivity

expressivity

coverage

semantics

separation of

concerns

completeness

paradigms

modularity

concrete

syntax

process

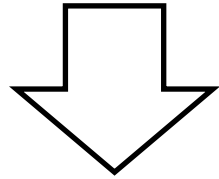
Shorter Programs

More
Accessible
Semantics

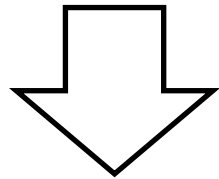
For a limited
Domain!

Domain Knowledge
encapsulated in
language

Smaller Domain



More Specialized
Language



Shorter Programs

The
do-what-I-want
language

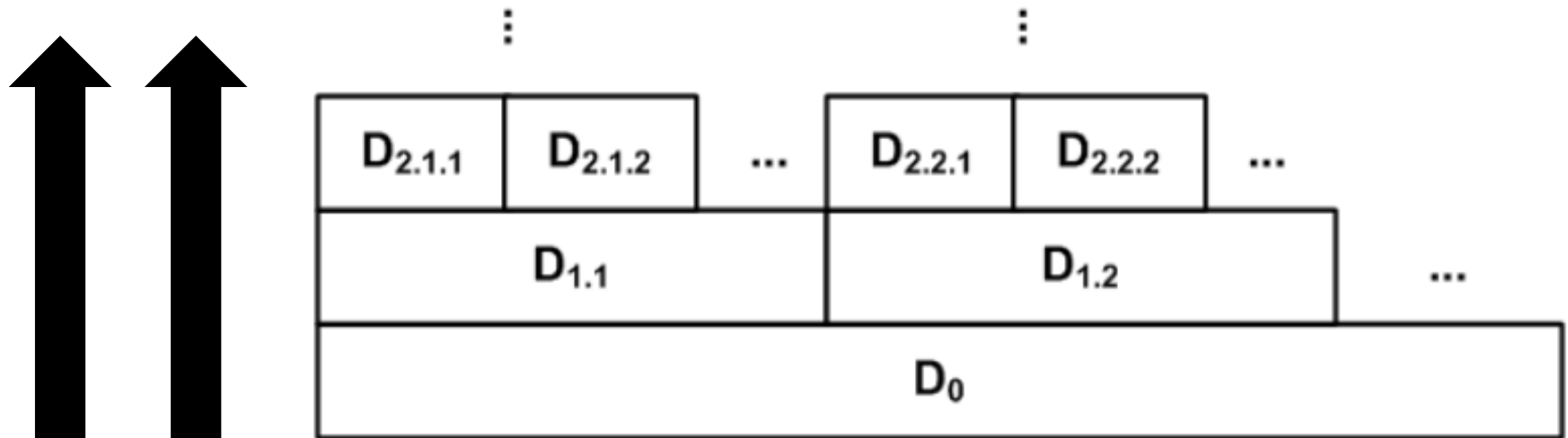
Ψ

Single Program
vs. Class/Domain

No Variability!

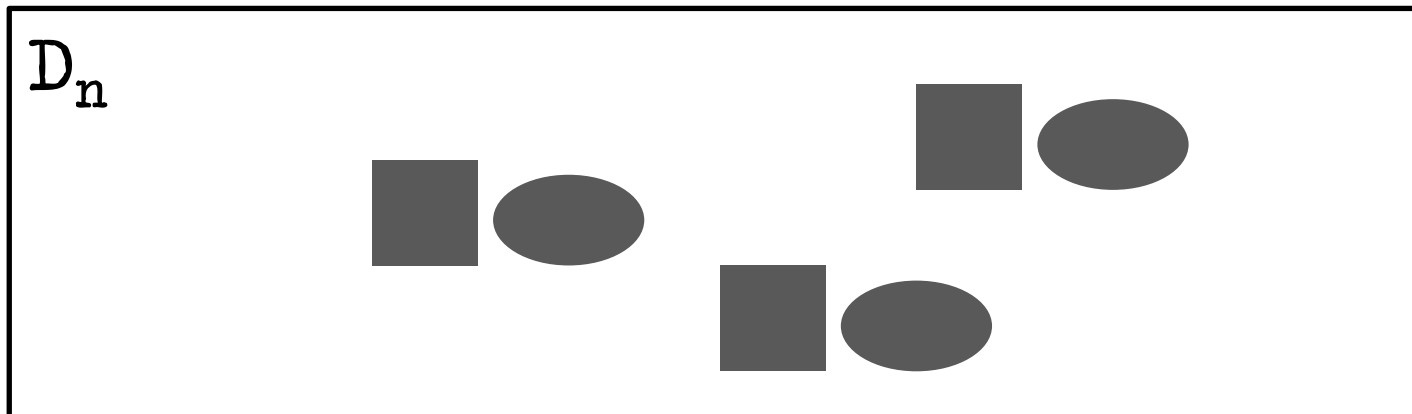
Ψ

Domain Hierarchy

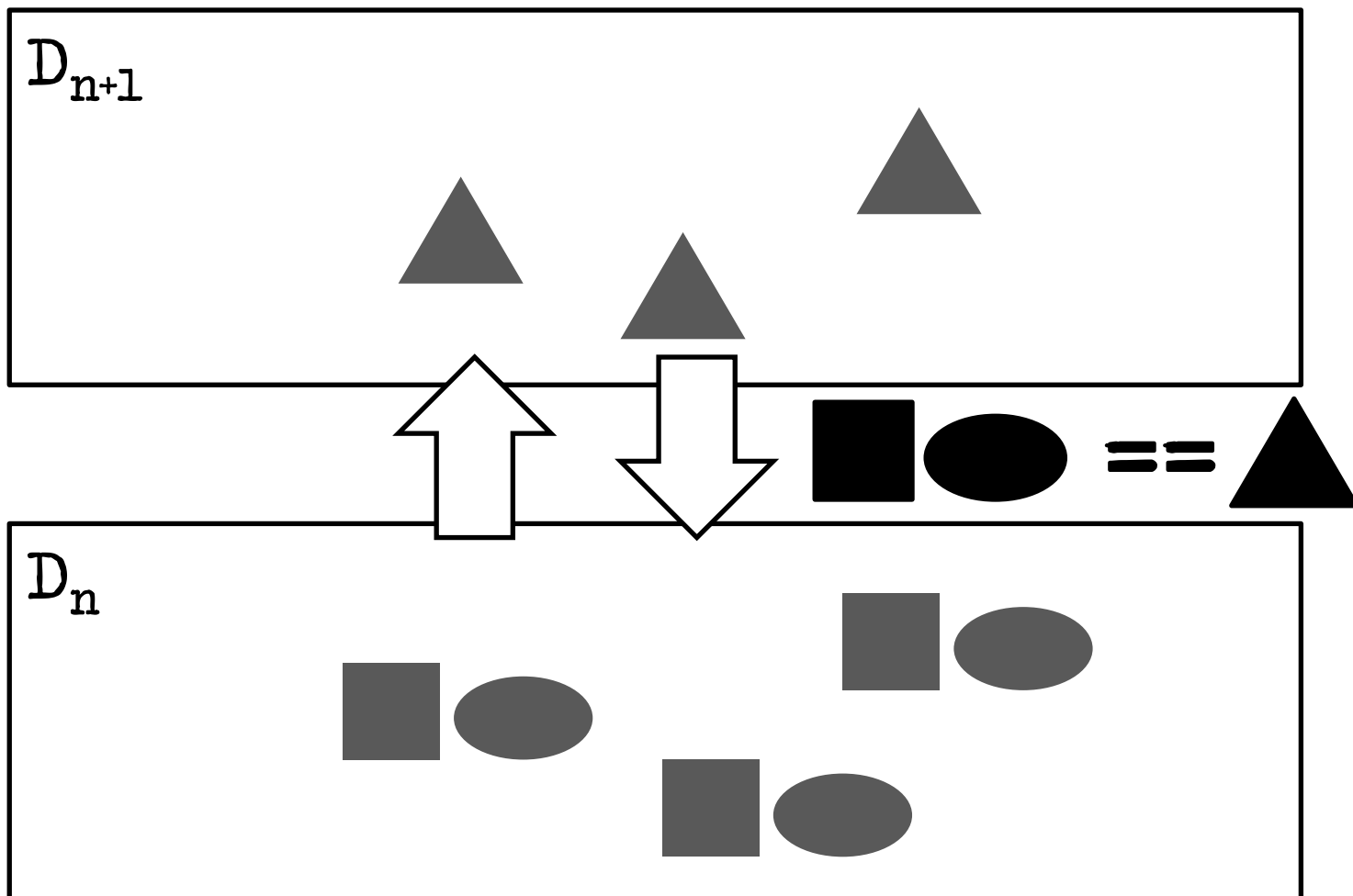


more specialized domains
more specialized languages

Reification

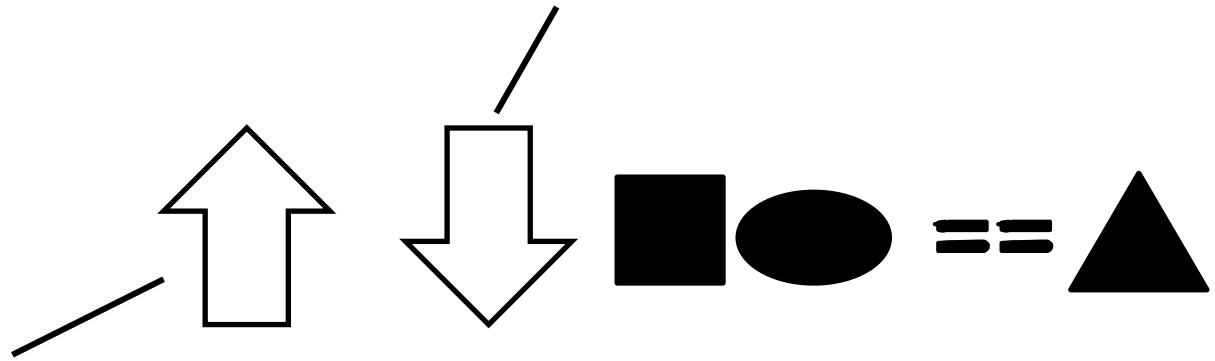


Reification



Reification

Transformation/
Generation



Language
Definition

```
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

```
int[] arr = ...
List<int> l = ...
for (int i=0; i<arr.size(); i++) {
    l.add( arr[i] );
}
```



Overspecification! Requires Semantic Analysis!

```
int[] arr = ...  
for (int i=0; i<arr.size(); i++) {  
    sum += arr[i];  
}
```

```
int[] arr = ...  
List<int> l = ...  
for (int i=0; i<arr.size(); i++) {  
    l.add( arr[i] );  
}
```



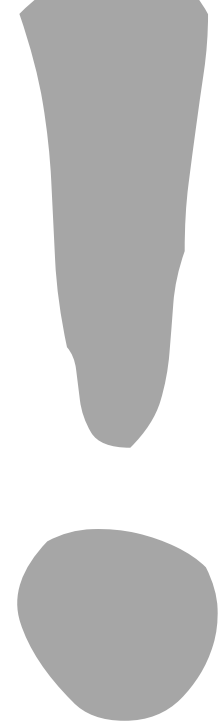
Linguistic Abstraction

```
for (int i in arr) {  
    sum += i;  
}
```

Declarative!

Directly represents Semantics.

```
seqfor (int i in arr) {  
    l.add( arr[i] );  
}
```



Def: DSL

A DSL is a **language** at D that provides **linguistic abstractions** for **common patterns and idioms** of a language at $D-1$ when used within the domain D .

Def: DSL cont'd

A good DSL does **not** require the use of patterns and idioms to express **semantically**

interesting concepts in D.

Processing tools do not have to do "semantic recovery" on D programs.

Declarative!

Semantics & Execution

expressivity
coverage
semantics
separation of
concerns

completeness
paradigms
modularity
concrete
syntax

process

Static Semantics

Execution Semantics

Static Semantics

Execution Semantics

Static Semantics

Constraints

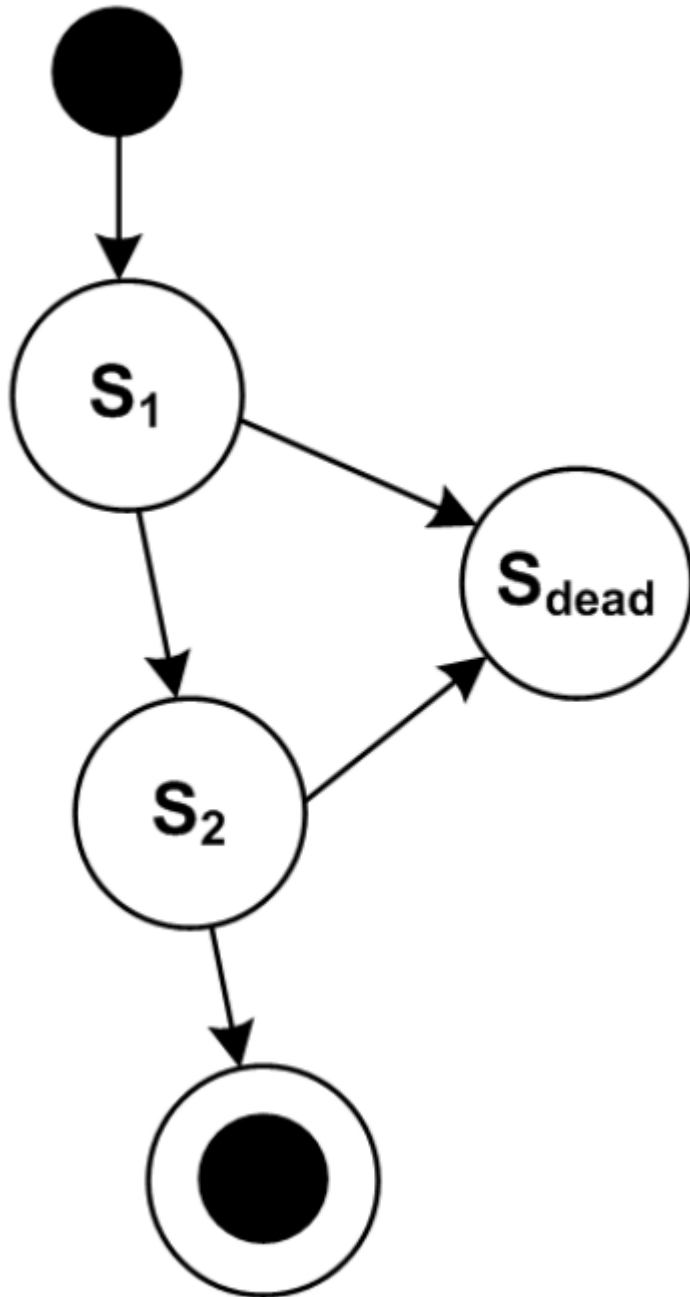
Type Systems

Unique State Names

Unreachable States

Dead End States

...



Example

Extended C

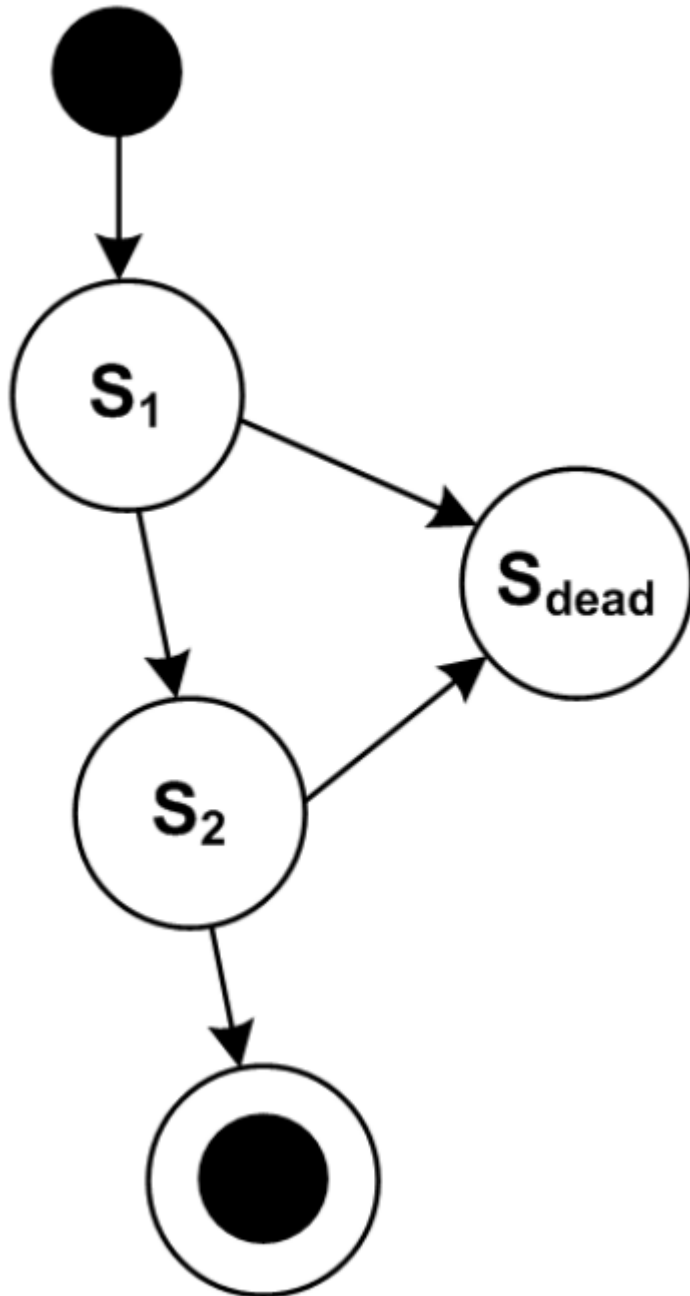
Unique State Names

Unreachable States

Dead End States

...

Easier to do on a
declarative Level!



Example

Extended C

Unique State Names

Unreachable States

Dead End States

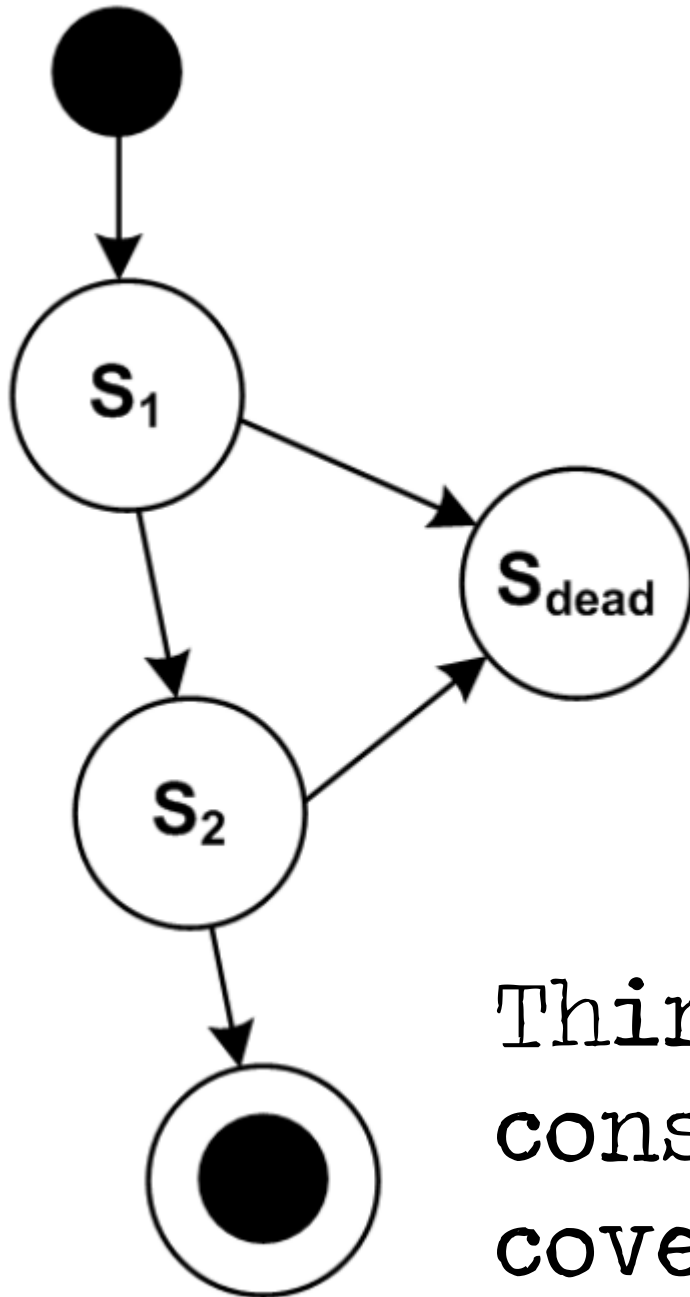
...

Easier to do on a
declarative Level!

Thinking of all
constraints is a
coverage problem!

Example

Extended C




```
var int x = 2 * someFunction(sqrt(2));
```

Assign fixed types

Derive Types

Calculate Common Types

Check Type Consistency

What does a type system do?

Intent +
Check

Derive

Intent +
Check

```
var int x = 2 *  
    someFunction(sqrt(2));
```

More code

Better error
messages

Better Performance

Derive

Intent + Check

```
var int x = 2 *  
    someFunction(sqrt(2));
```

More code

Better error
messages

Better Performance

Derive

```
var x = 2 * some  
    Function(sqrt(2));
```

More convenient

More complex
checkers

Harder to
understand for
users

What does it
all mean?

Execution Semantics

Def: Semantics

... via mapping to lower level

$$\mathit{semantics}(p_{L_D}) := q_{L_D-1}$$

$$\textit{where } OB(p_{L_D}) == OB(q_{L_D-1})$$

OB: Observable Behaviour (Test Cases)

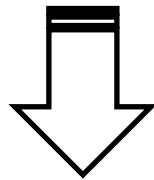
Def: Semantics

... via mapping to lower level

$$\text{semantics}(p_{L_D}) := q_{L_{D-1}}$$

$$\text{where } OB(p_{L_D}) == OB(q_{L_{D-1}})$$

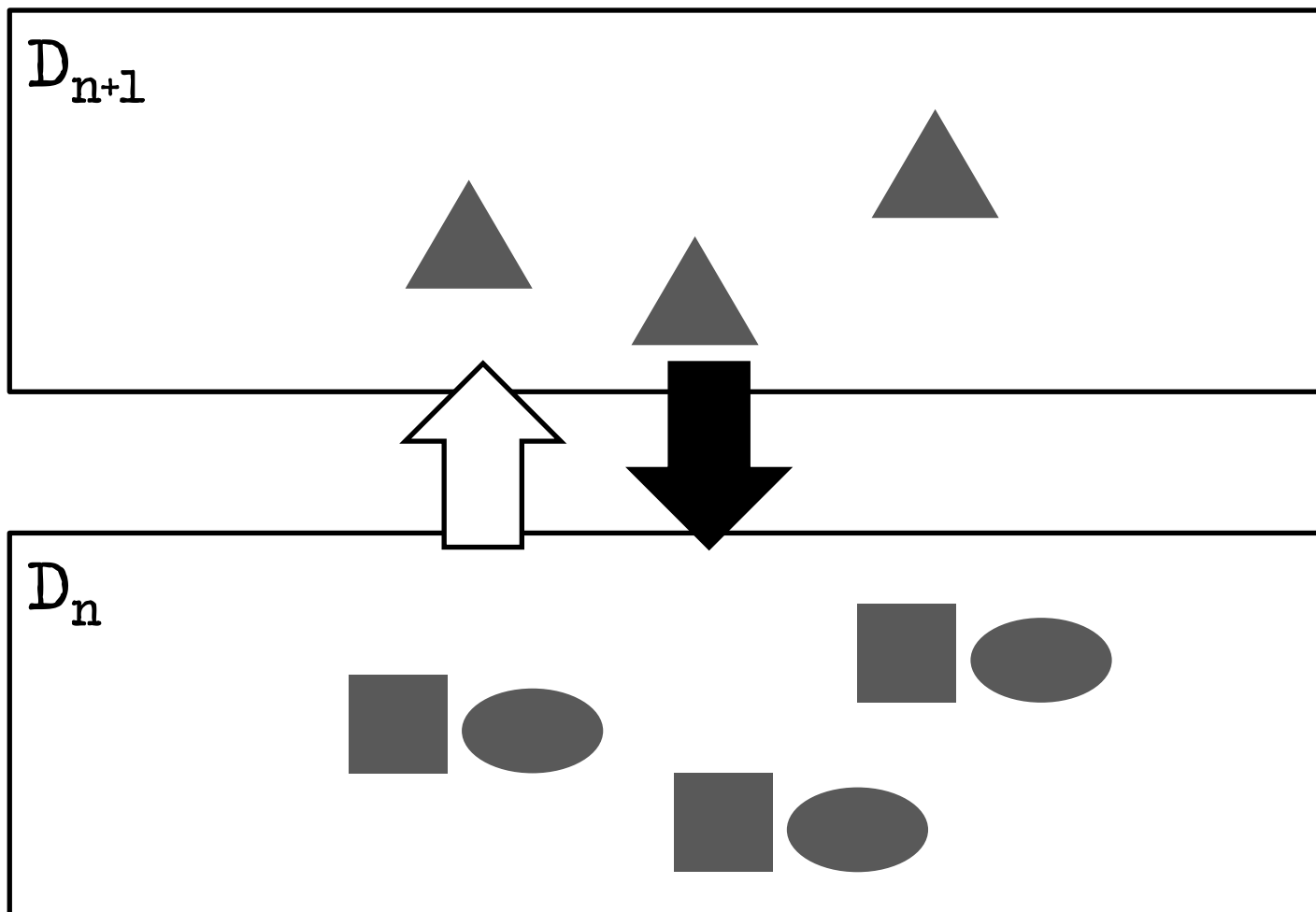
L_D



Transformation
Interpretation

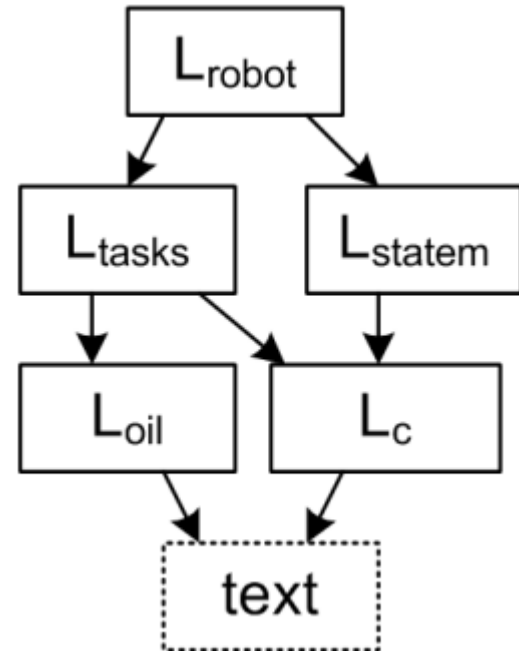
L_{D-1}

Transformation



Transformation

```
module impl imports <<imports>> {  
  
  int speed( int val ) {  
    return 2 * val;  
  }  
  
  robot script stopAndGo  
    block main on bump  
      accelerate to 12 + speed(12) within 3000  
      drive on for 2000  
      turn left for 200  
      decelerate to 0 within 3000  
      stop  
  
}
```

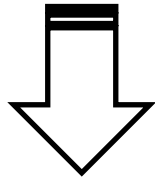


Example

Extended C

Transformation

L_D



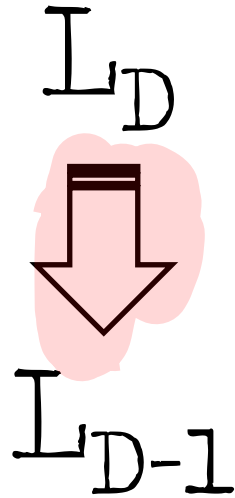
Transformation

L_{D-1}

Known Semantics!

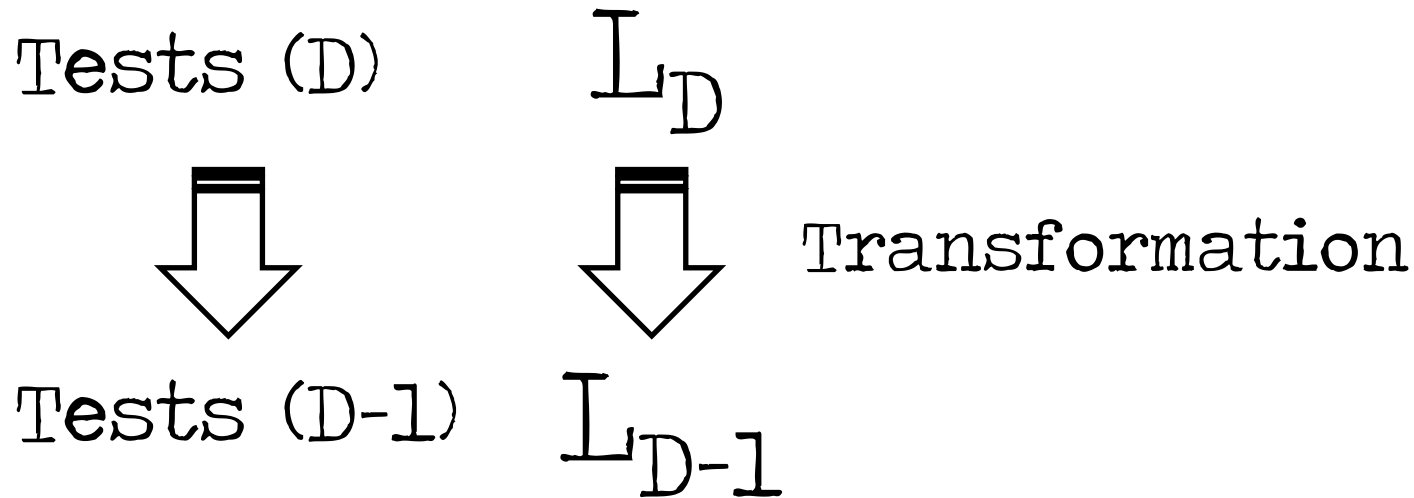
Transformation

Correct!?



Transformation

Transformation



Run tests on both levels; all pass.
Coverage Problem!

```

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

Refrige
rators

Transformation

| Name | Documentation | Tags | Valid time | Transaction time | Fixture | Product | Element | Expected value | Actual value |
|------------------------------|---------------|------|------------|------------------|---------------|-----------------|------------------------------------|----------------|--------------|
| Accrued right at retireme | | | 2006-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued right | 761.0402 | 761.0402 |
| Accrued Right last final pay | | | 2004-1-1 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued right | 705.0589 | 705.0589 |
| premium last year | | | 2006-1-1 | 2007-9-24 | Jan De Jong | Old Age Pension | Premium old age pension | 329.0625 | 329.0625 |
| Accrued right at retireme 2) | | | 2006-12-31 | 2007-9-24 | Piet Van Dijk | Old Age Pension | Accrued right | 740.94 | 724.7658 |
| | | | 1985-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued Right in service period | 73.661 | 73.661 |
| | | | 1985-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Years of service in service period | 3.7534 | 3.7534 |
| | | | 1987-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Pension base average FP | 7750 | |
| | | | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued Right in service period | 387.7449 | 387.7449 |
| | | | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Years of service in service period | 10.8082 | |
| | | | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Pension base average FP | 8250 | |

Example

Pension Plans

Simulation View

Status

Current Test: KIRAbtauen

Current State: -

Current Step: -

Control

Autorun Single Step

10 Enable Breakpoints

Property Values

| Property | Value |
|----------------------|-------|
| RC.accumulatedRun... | 80 |
| RC.needsCooling | false |
| c1.active | false |
| ccfan.active | false |
| rcdoor.open | false |
| rceva.evaTemp | 20 |
| rcfan.active | false |

Queue

| Event | Data |
|-------|------|
| | |

Commands

| St... | Command |
|-------|---------|
| | |

Variable Values

| Variable | Value |
|-----------------------|-------|
| tuerNachlaufSchwel... | 0 |

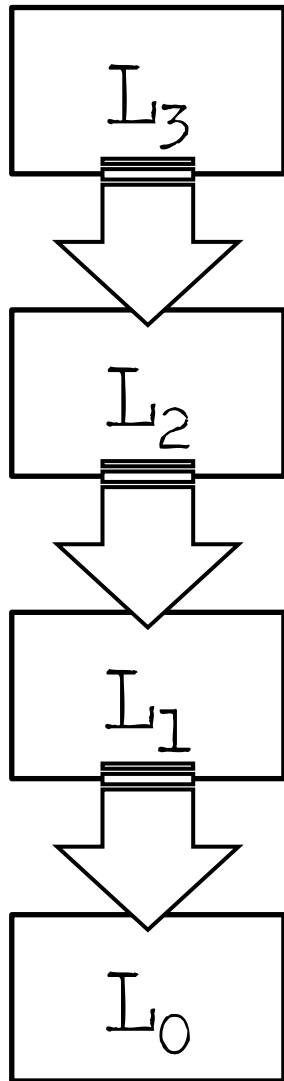
Running Tasks

| Task | Sinc... |
|------|---------|
| | |

Example

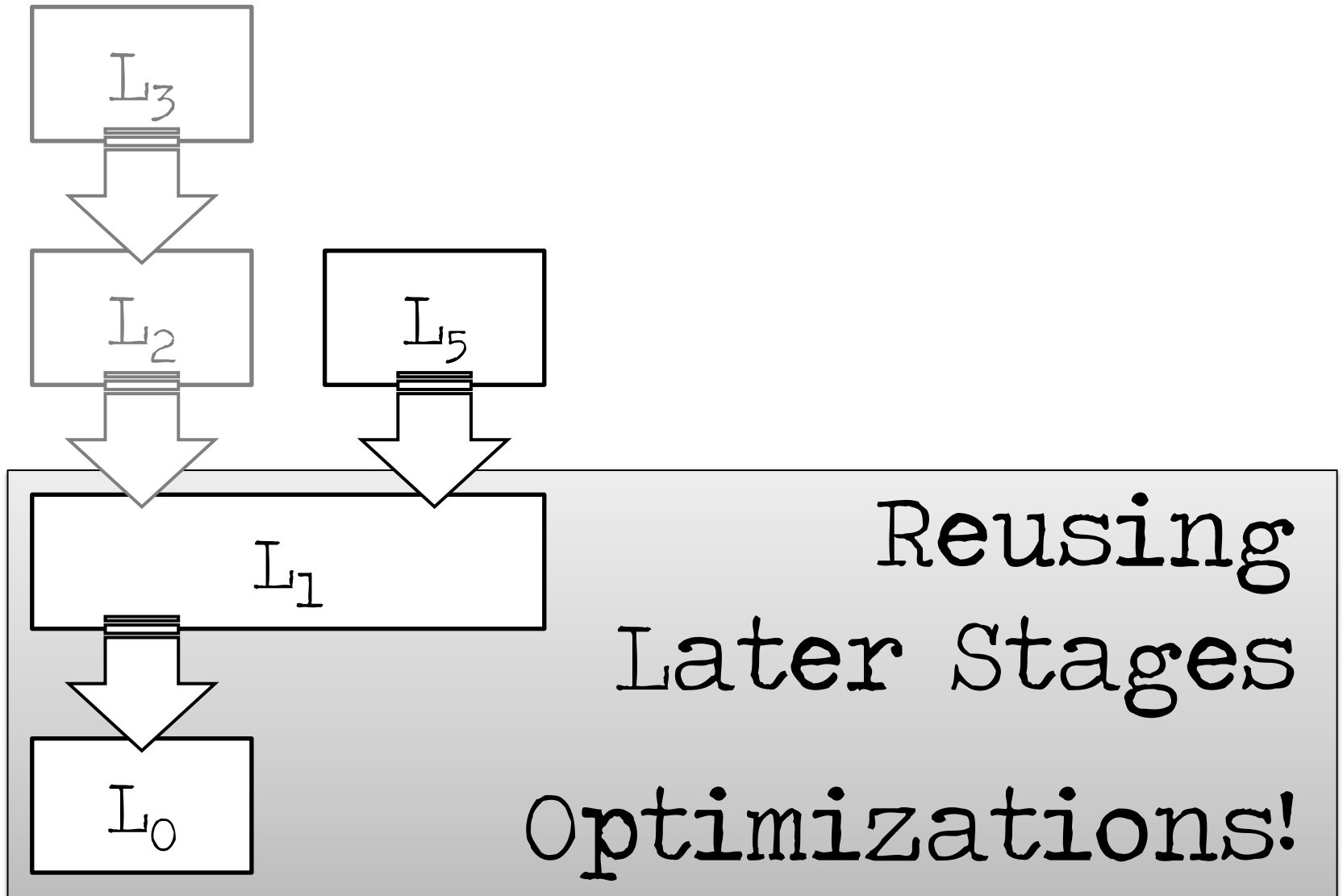
Refrige
rators

Multi-Stage

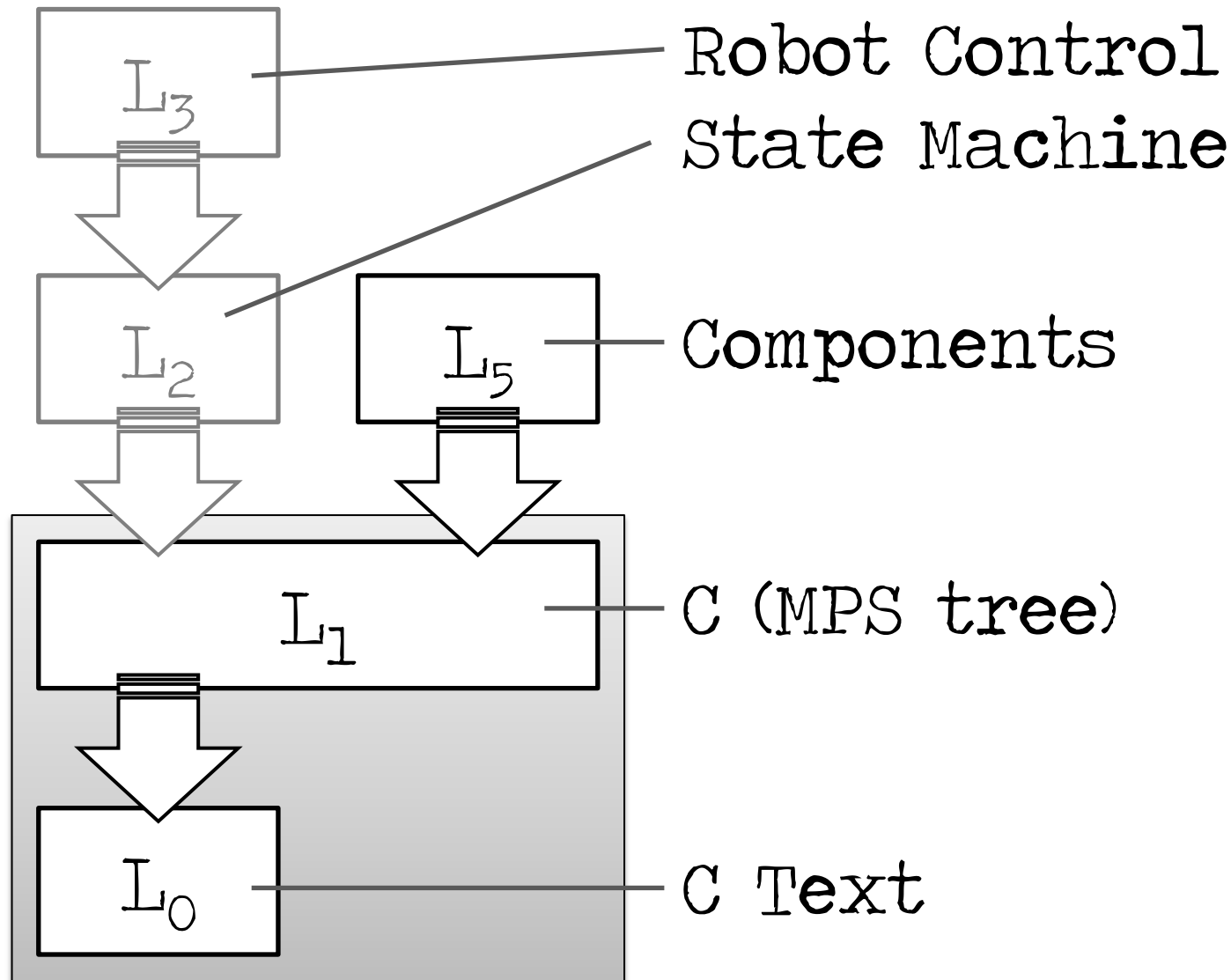


Modularization

Multi-Stage: Reuse



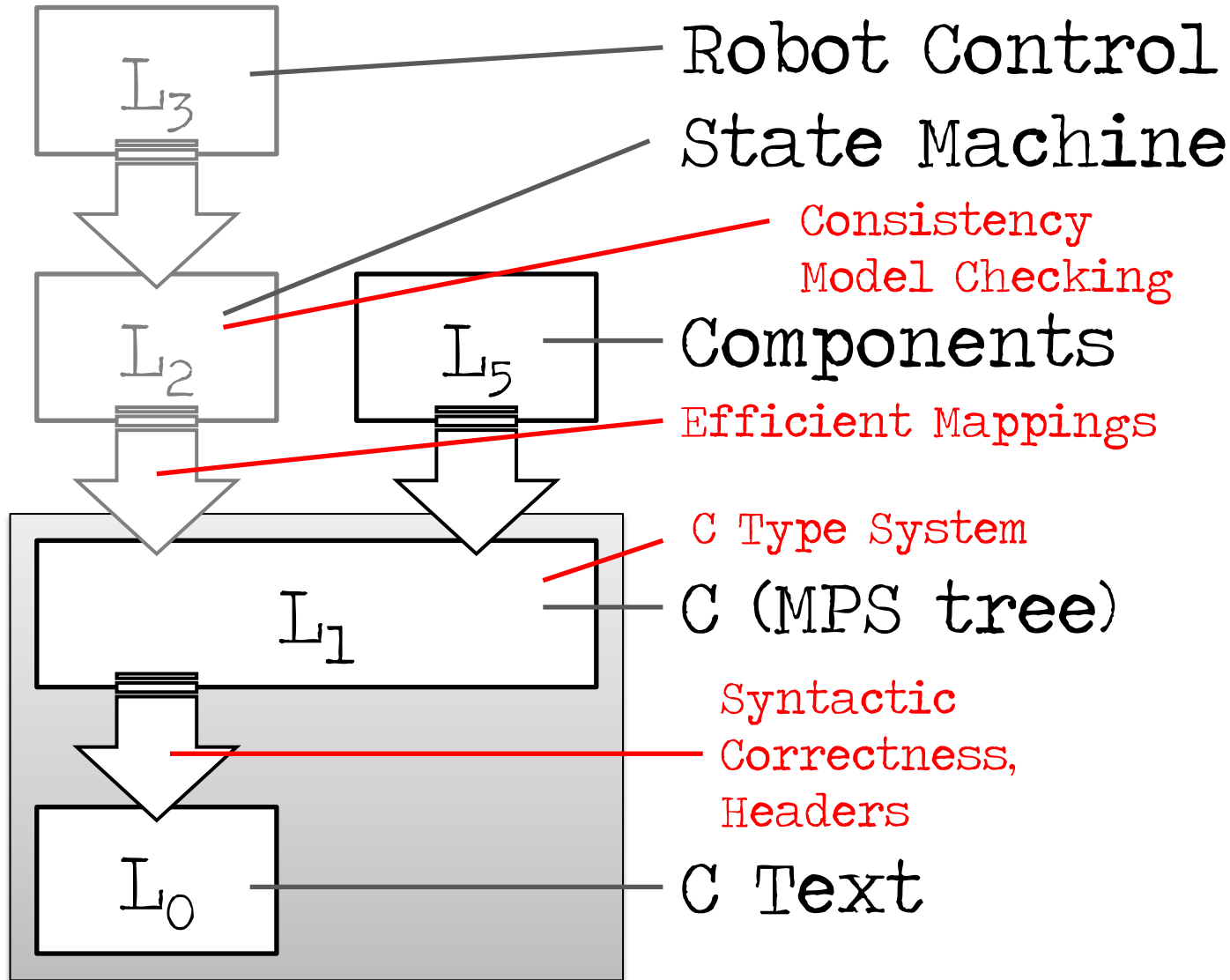
Multi-Stage: Reuse



Example

Extended C

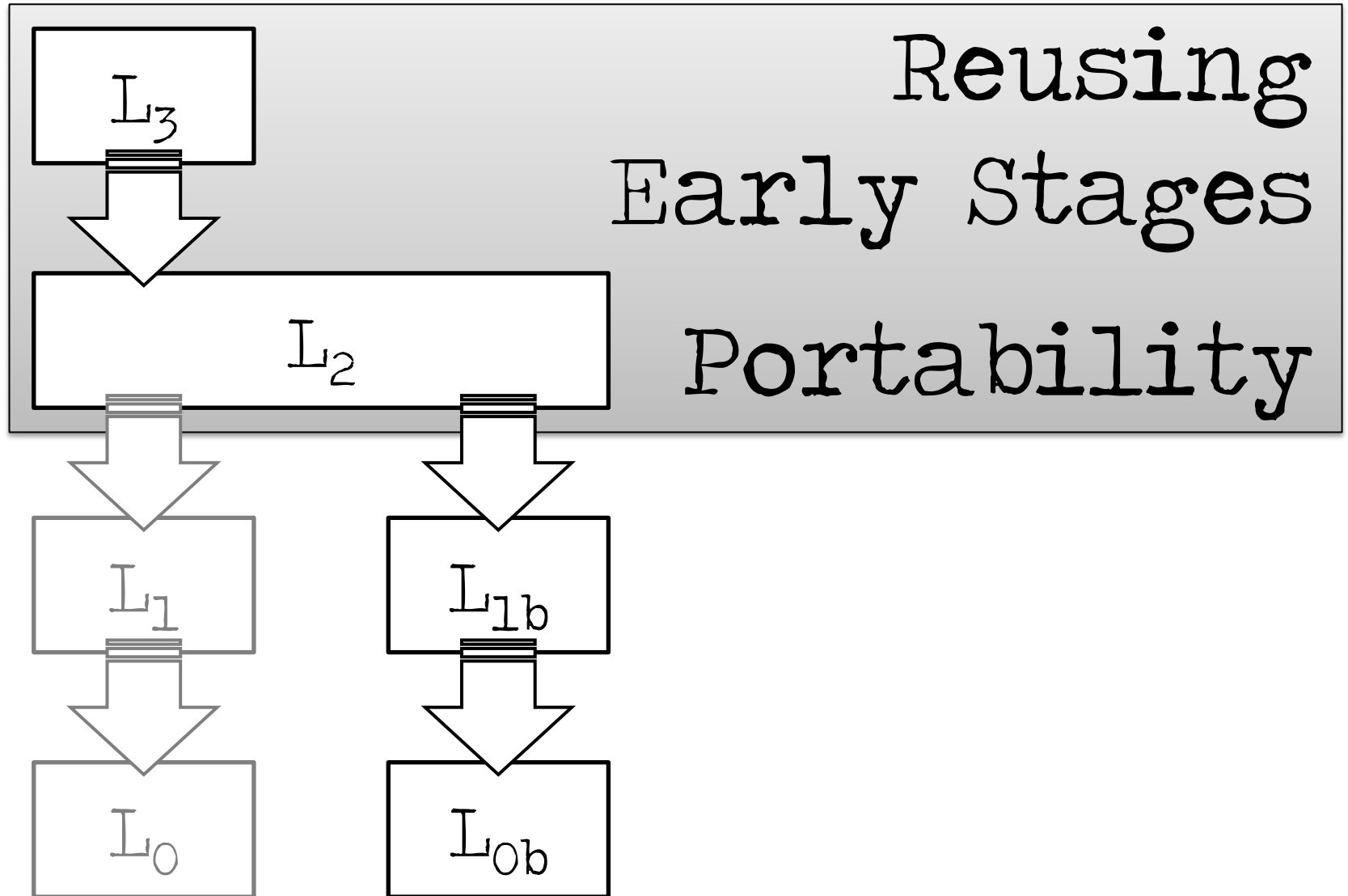
Multi-Stage: Reuse



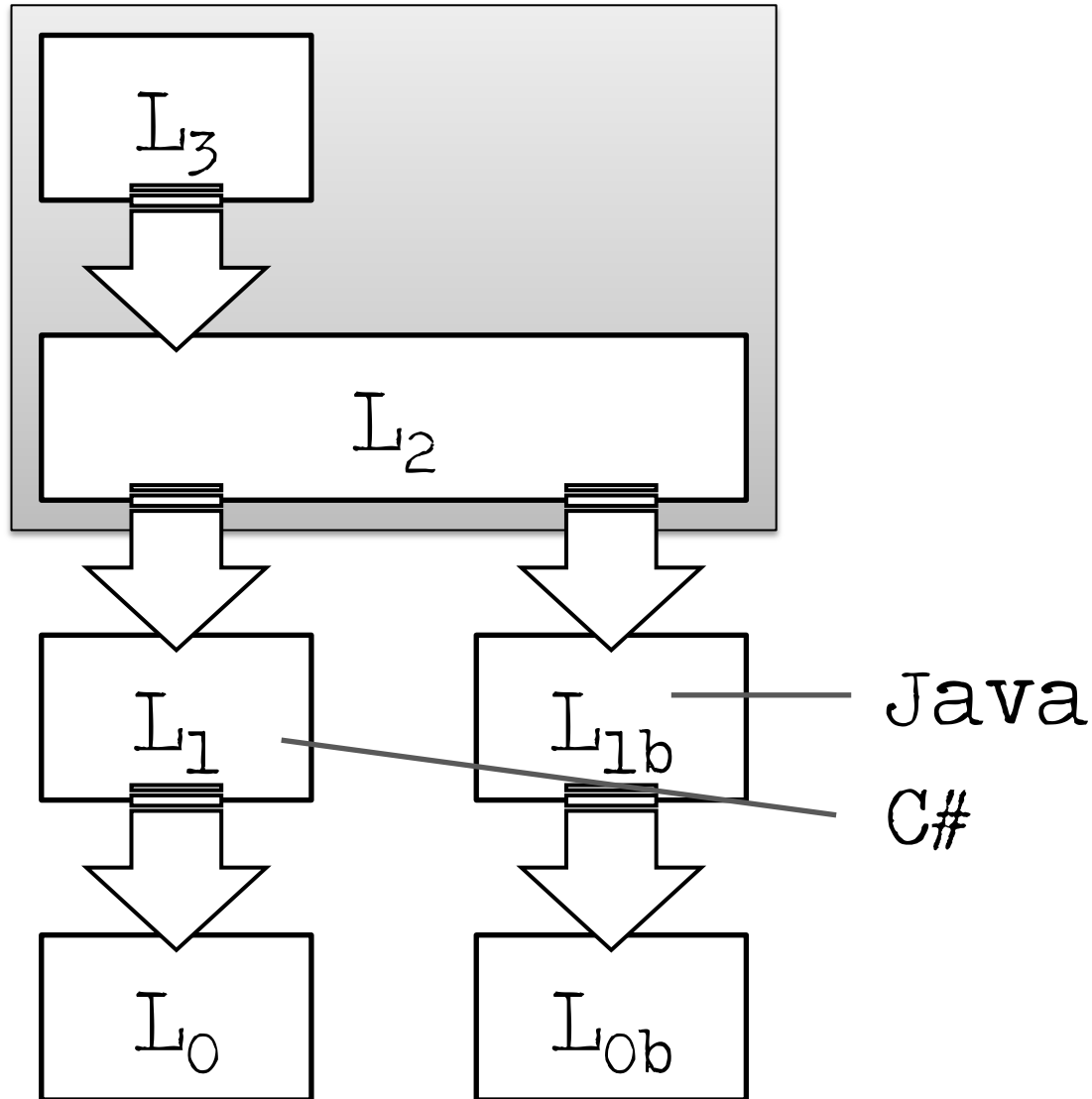
Example

Extended C

Multi-Stage: Reuse



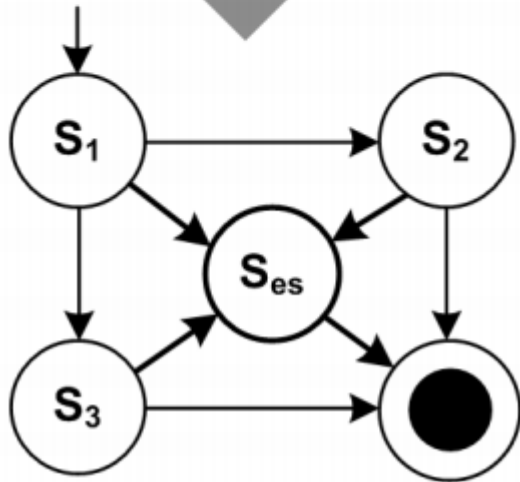
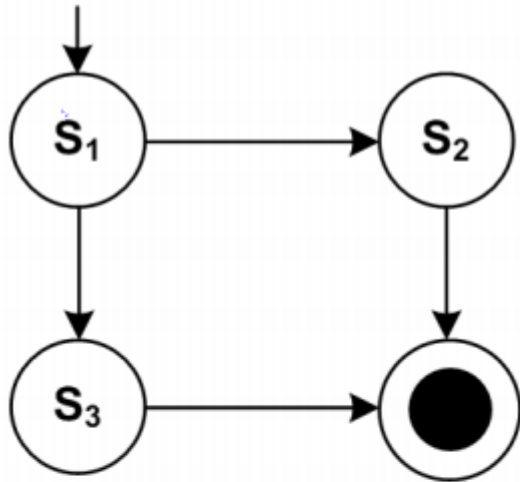
Multi-Stage: Reuse



Example

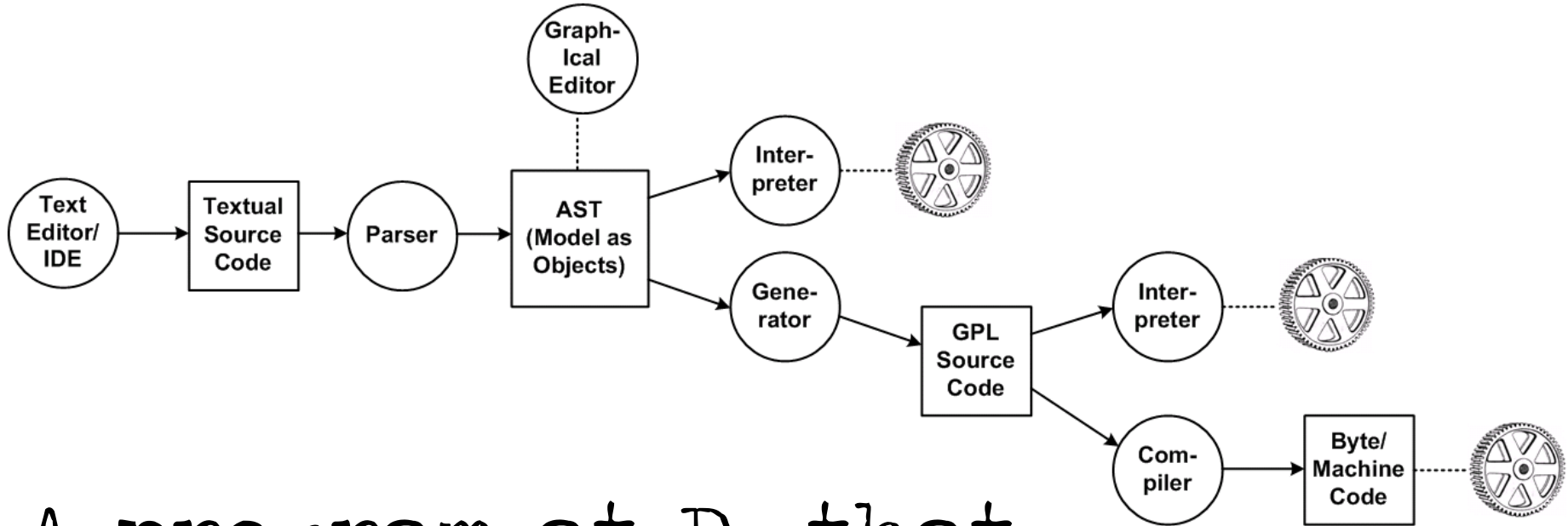
Extended C

Multi-Stage: Preprocess



Adding an
optional, modular
emergency
stop feature

Interpretation



A program at D_0 that acts on the structure of an input program at $D_{>0}$

Interpretation

A program at D_0 that
acts on the structure
of an input program at D_{x_0}

imperative \rightarrow step through

functional \rightarrow eval recursively

declarative \rightarrow ? solver ?

| Name | Documentation | Tags | Valid time | Transaction time | Fixture | Product | Element | Expected value | Actual value |
|------------------------------|---------------|------|------------|------------------|---------------|-----------------|------------------------------------|----------------|--------------|
| Accrued right at retireme | | | 2006-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued right | 761.0402 | 761.0402 |
| Accrued Right last final pay | | | 2004-1-1 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued right | 705.0589 | 705.0589 |
| premium last year | | | 2006-1-1 | 2007-9-24 | Jan De Jong | Old Age Pension | Premium old age pension | 329.0625 | 329.0625 |
| Accrued right at retireme 2) | | | 2006-12-31 | 2007-9-24 | Piet Van Dijk | Old Age Pension | Accrued right | 740.94 | 724.7658 |
| | | | 1985-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued Right in service period | 73.661 | 73.661 |
| | | | 1985-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Years of service in service period | 3.7534 | 3.7534 |
| | | | 1987-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Pension base average FP | 7750 | 7750 |
| | | | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued Right in service period | 387.7449 | 387.7449 |
| | | | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Years of service in service period | 10.8082 | 10.8082 |
| | | | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Pension base average FP | 8250 | 8250 |

Example

Pension
Plans

```

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

Refrige
rators

Simulation View

Status

Current Test: KIRAbtauen

Current State: -

Current Step: -

Control

Autorun Single Step

10 Enable Breakpoints

Property Values

| Property | Value |
|----------------------|-------|
| RC.accumulatedRun... | 80 |
| RC.needsCooling | false |
| c1.active | false |
| ccfan.active | false |
| rcdoor.open | false |
| rceva.evaTemp | 20 |
| rcfan.active | false |

Queue

| Event | Data |
|-------|------|
| | |

Commands

| St... | Command |
|-------|---------|
| | |

Variable Values

| Variable | Value |
|-----------------------|-------|
| tuerNachlaufSchwel... | 0 |

Running Tasks

| Task | Sinc... |
|------|---------|
| | |

Example

Refrige
rators

Transformation

Interpretation

Transformation

+ Code Inspection

Interpretation

Transformation

+ Code Inspection
OSGi Generators

Interpretation

Example

Components

Transformation

- + Code Inspection
- + Debugging

Interpretation

Transformation

- + Code Inspection
- + Debugging
- Platform Interactions

Interpretation

Example

Refrigerators

Transformation

- + Code Inspection
- + Debugging
- + Performance & Optimization

Interpretation

Transformation

+ Code Inspection

+ Debugging

+ Performance &
Optimization

Efficiency for
Real-Time S/w

Memory Use

Interpretation

Example

Extended C

Transformation

- + Code Inspection
- + Debugging
- + Performance & Optimization
- + Platform Conformance

Interpretation

Transformation

- + Code Inspection
- + Debugging
- + Performance & Optimization
- + Platform Conformance

Web Frameworks
and Standards

Interpretation

Example

Web
DSL

Transformation

- + Code Inspection
- + Debugging
- + Performance & Optimization
- + Platform Conformance

Interpretation

- + Turnaround Time

Transformation

- + Code Inspection
- + Debugging
- + Performance & Optimization
- + Platform Conformance

Interpretation

- + Turnaround Time
Testing

Example

Pension
Plans

Example

Refrige
rators

Transformation

- + Code Inspection
- + Debugging
- + Performance & Optimization
- + Platform Conformance

Interpretation

- + Turnaround Time
- + Runtime Change

Transformation

- + Code Inspection
- + Debugging
- + Performance & Optimization
- + Platform Conformance

Interpretation

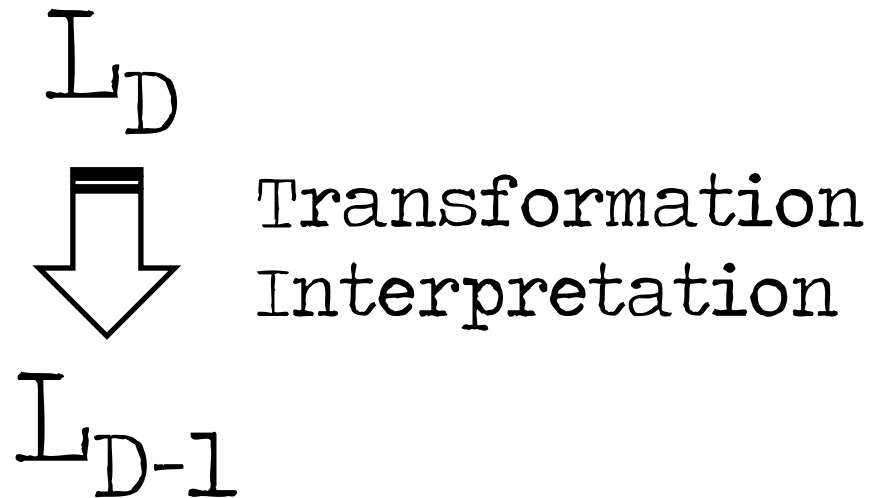
- + Turnaround Time
- + Runtime Change
Change Business
Rules without
Redeployment

Example

Pension
Plans

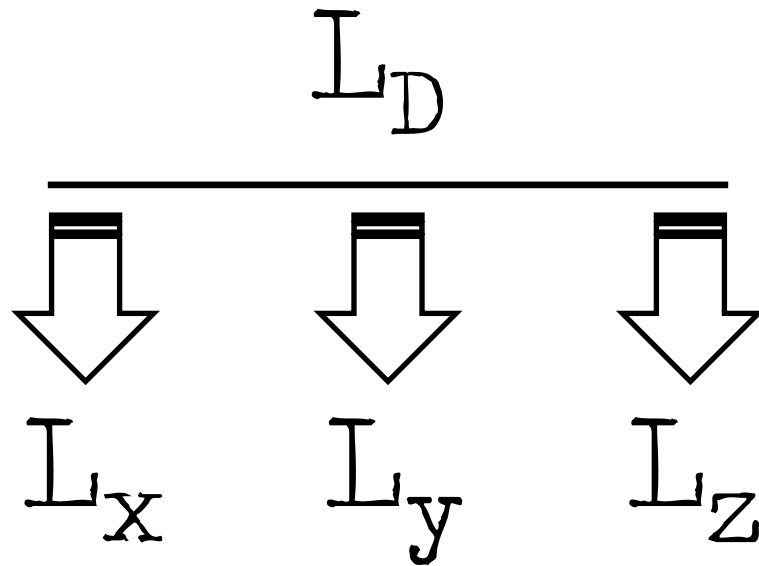
Def: Semantics

... via mapping to lower level



Multiple Mappings

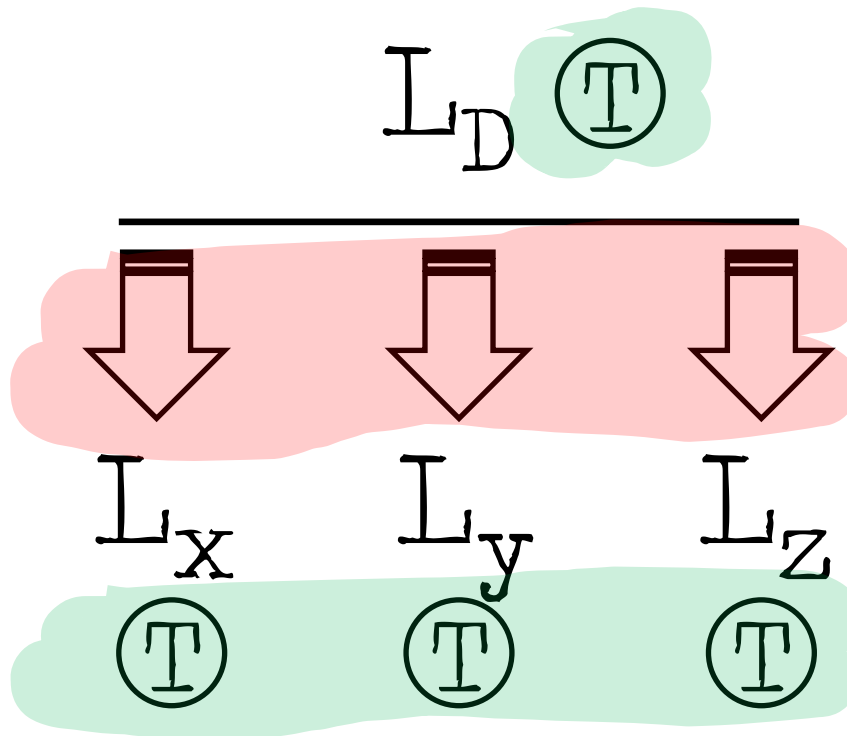
... at the same time



Similar
Semantics?

Multiple Mappings

... at the same time



Similar
Semantics?

all green!

| Name | Documentation | Tags | Valid time | Transaction time | Fixture | Product | Element | Expected value | Actual value |
|------------------------------|---------------|------|------------|------------------|---------------|-----------------|------------------------------------|----------------|--------------|
| Accrued right at retireme | | | 2006-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued right | 761.0402 | 761.0402 |
| Accrued Right last final pay | | | 2004-1-1 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued right | 705.0589 | 705.0589 |
| premium last year | | | 2006-1-1 | 2007-9-24 | Jan De Jong | Old Age Pension | Premium old age pension | 329.0625 | 329.0625 |
| Accrued right at retireme 2) | | | 2006-12-31 | 2007-9-24 | Piet Van Dijk | Old Age Pension | Accrued right | 740.94 | 724.7658 |
| | | | 1985-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued Right in service period | 73.661 | 73.661 |
| | | | 1985-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Years of service in service period | 3.7534 | 3.7534 |
| | | | 1987-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Pension base average FP | 7750 | 7750 |
| | | | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued Right in service period | 387.7449 | 387.7449 |
| | | | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Years of service in service period | 10.8082 | 10.8082 |
| | | | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Pension base average FP | 8250 | |

Example

Pension Plans

```

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

Refrige
rators

Reduced Expressiveness

bad?

maybe.

good?

maybe!

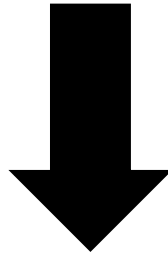
Reduced Expressiveness

bad?

maybe.

good?

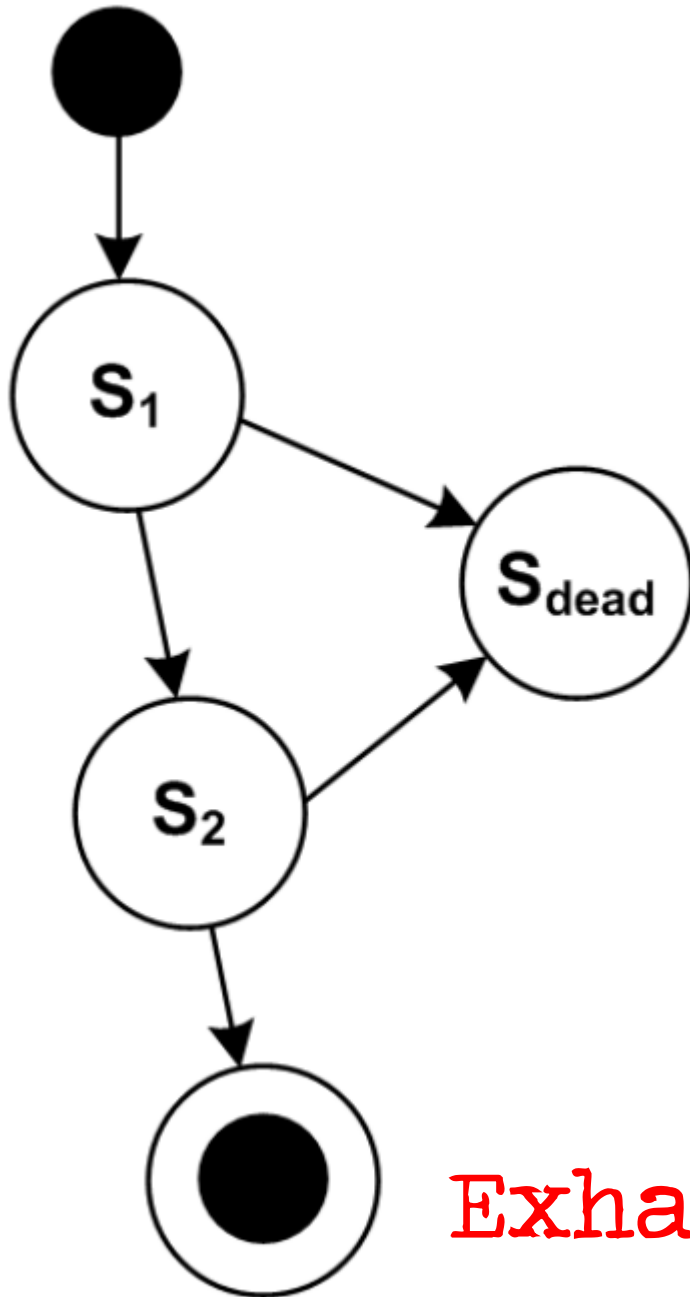
maybe!



Model Checking

SAT Solving

Exhaustive Search, Proof!



Unique State Names

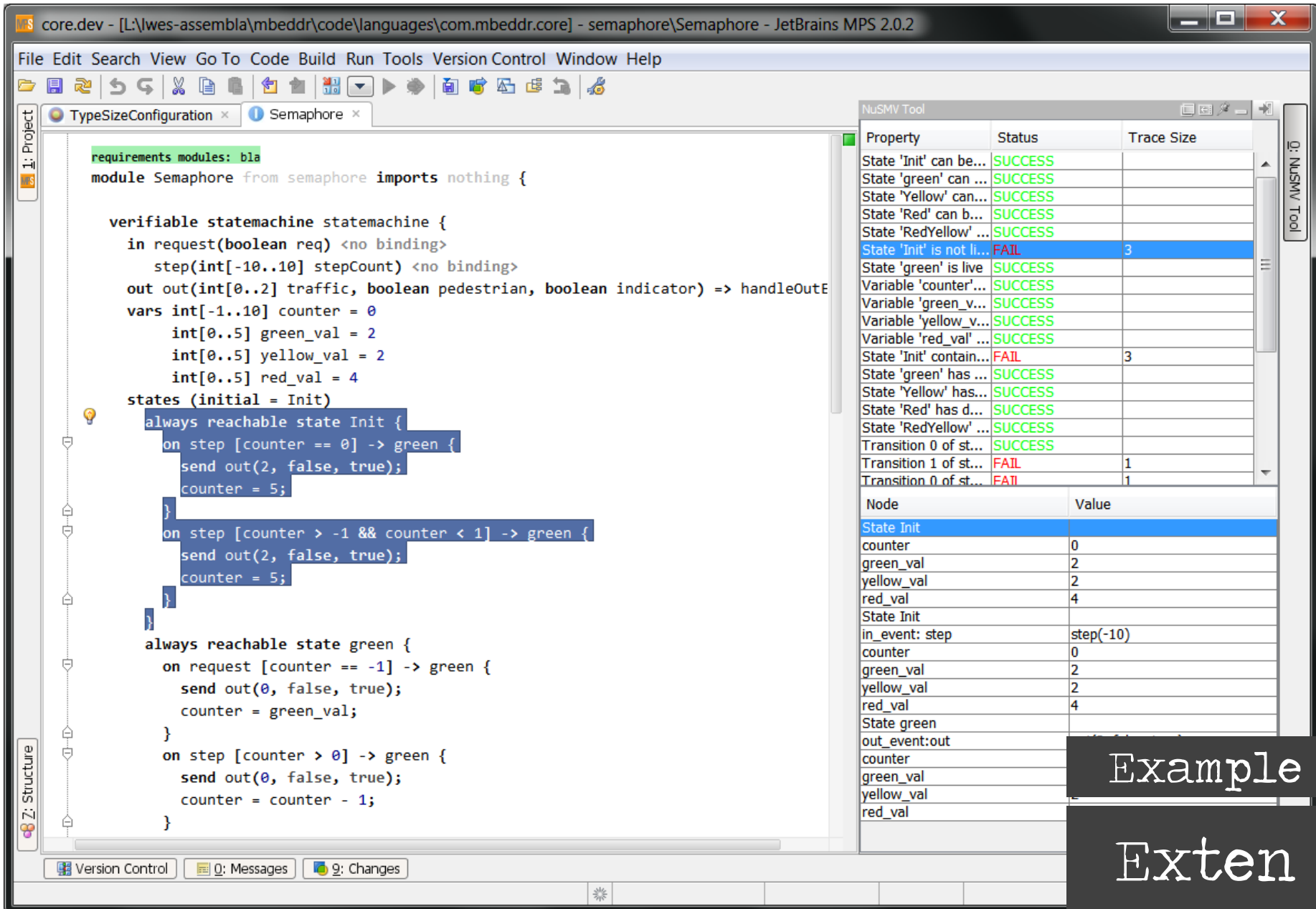
Unreachable States

Dead End States

Guard Decidability

Reachability

Exhaustive Search, Proof!



Example

Exten
ded C

```

c/s interface Decider {
  int decide(int x, int y) pre
}

component AComp extends nothing {
  ports:
    provides Decider decider
  contents:
    int decide(int x, int y) <- op decider.decide {
      return int, 0

```

| | | | |
|--------|--------|-------|--|
| | x == 0 | x > 0 | |
| y == 0 | 0 | 1 | |
| y > 0 | 1 | 2 | |

Example

Extended C

Separation of Concerns

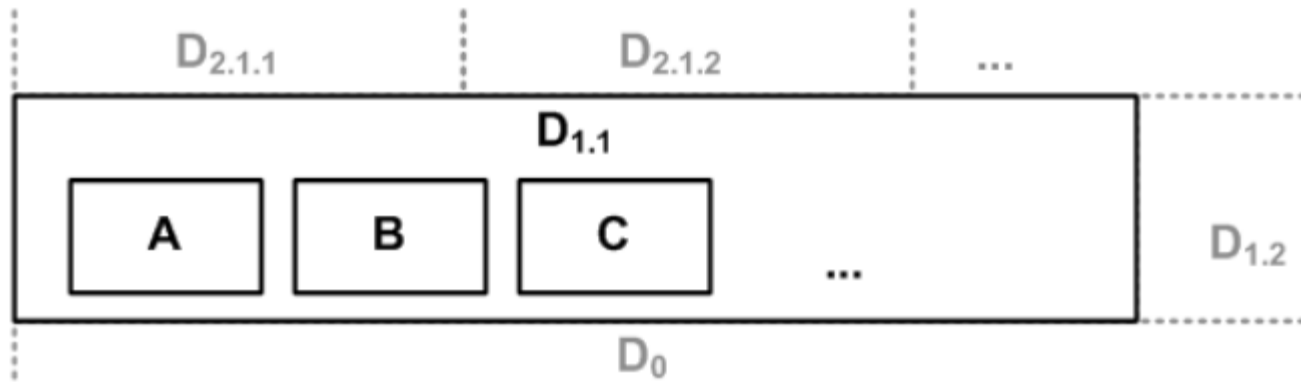
expressivity
coverage
semantics
separation of
concerns

completeness
paradigms
modularity
concrete
syntax

process

Several Concerns

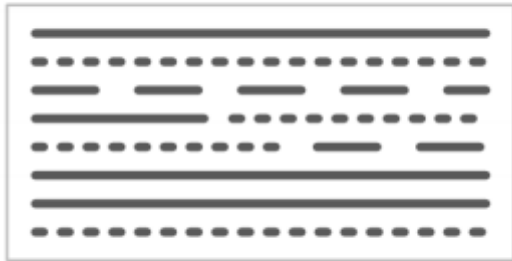
... in one domain



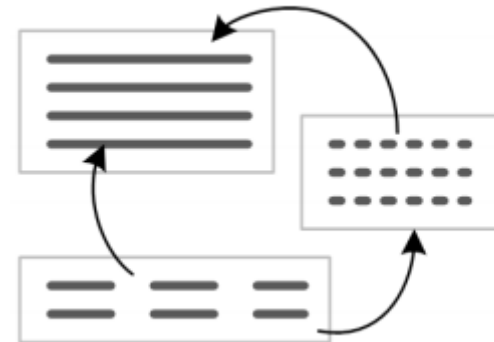
Several Concerns

... in one domain

integrated into
one fragment



separated into
several fragments

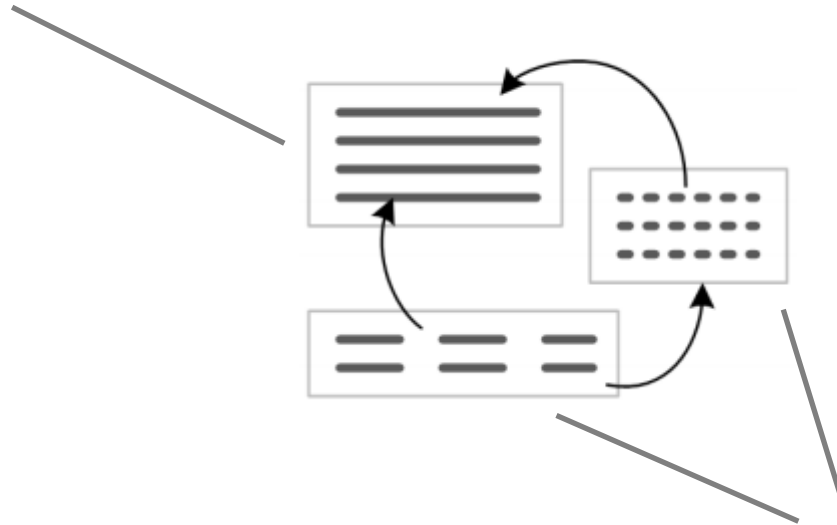


Viewpoints

$$\forall r \in Refs_f \mid fo(r.to) = fo(r.from) = f$$

independent

$$\forall e \in E_f \mid lo(co(e)) = l$$



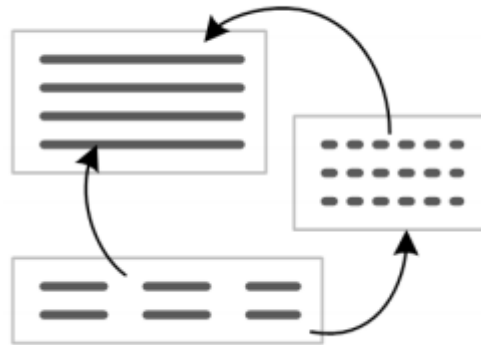
dependent

Viewpoints

Hardware

Behavior

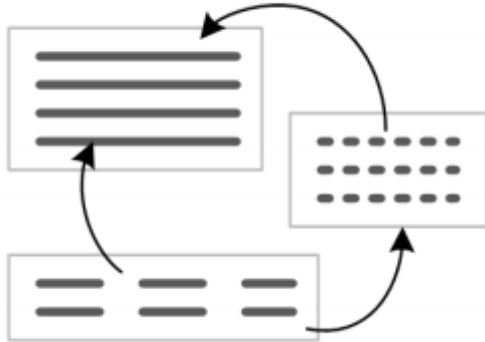
Tests



Example

Refrige
rators

Viewpoints: Why?



Sufficiency

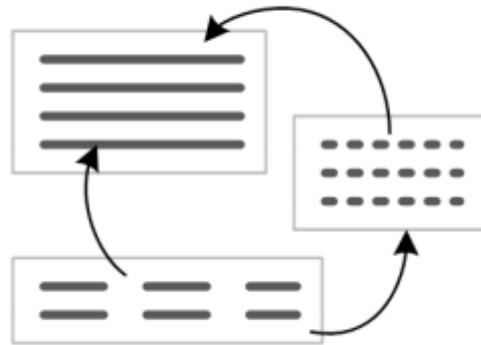
Different Stakeholders

Different Steps in
Process - VCS unit!

Viewpoints

Hardware

Product
Management



Behaviour

Thermo-
dynamics-
Experts

Tests

Example

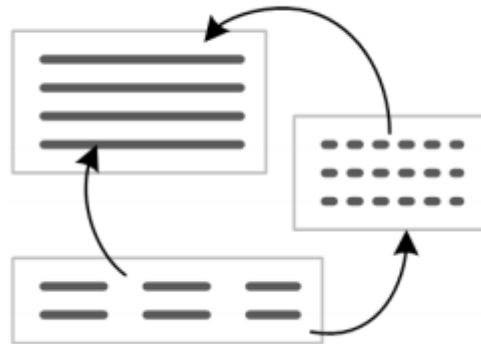
Refrige-
rators

Viewpoints

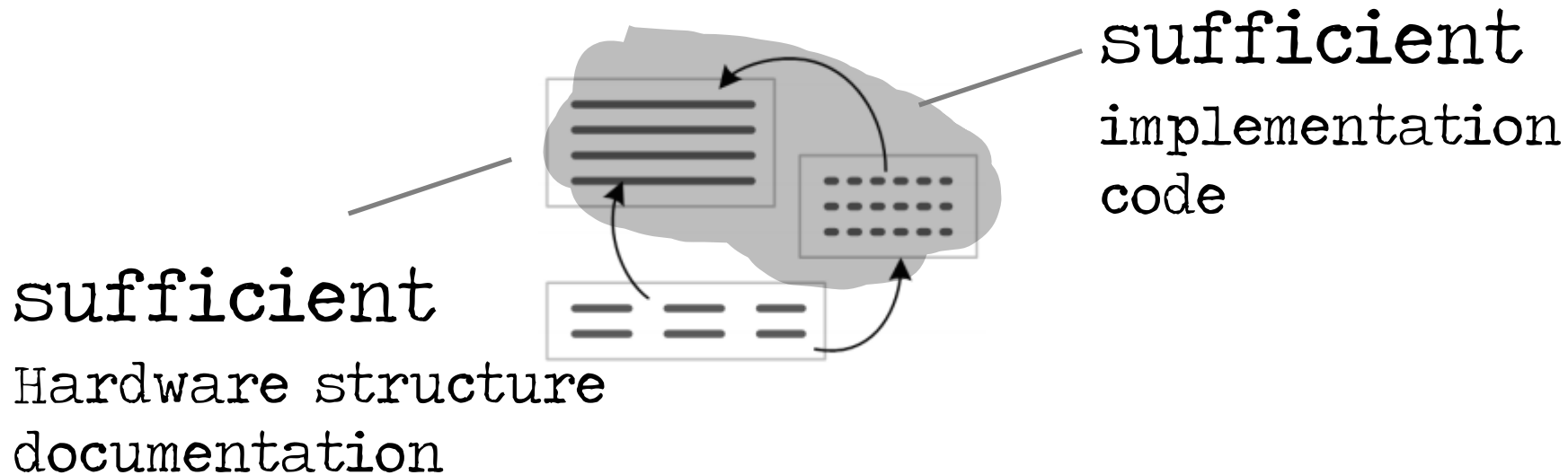
independent

sufficient?

contains all
the data for
running a
meaningful
transformation



Viewpoints

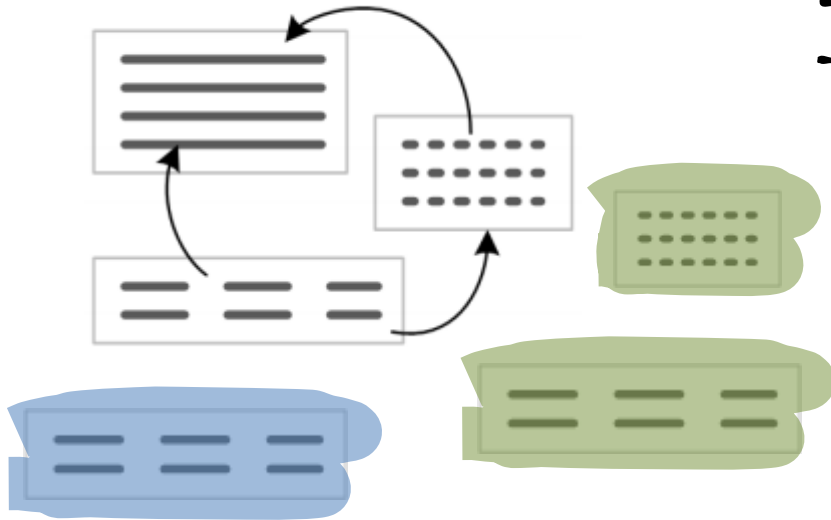


Example

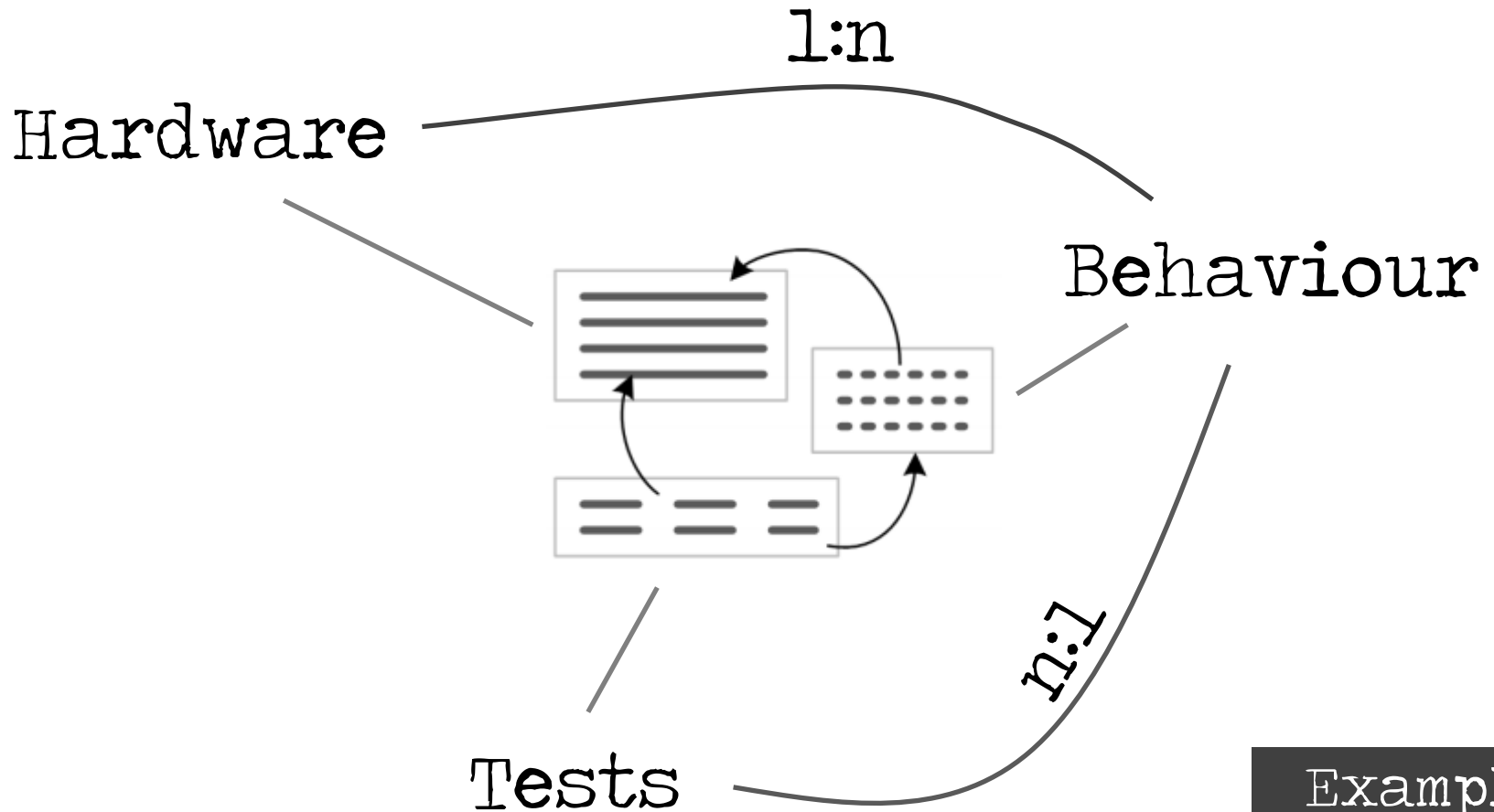
Refrige
rators

Viewpoints: Why?

1:n Relationships



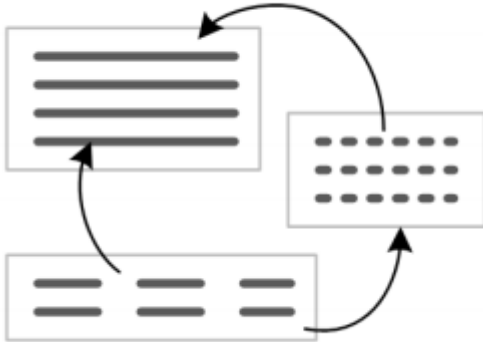
Viewpoints



Example

Refrige
rators

Viewpoints



Well-defined
Dependencies

No Cycles!

Avoid Synchronization!
(unless you use a projectional editor)

Language Modularity

expressivity
coverage
semantics
separation of
concerns

completeness
paradigms
modularity
concrete
syntax

process

Language Modularity, Composition and Reuse (LMR&C)

increase efficiency
of DSL development

Language Modularity, Composition and Reuse (LMR&C)

increase efficiency
of DSL development

4 ways of composition:

Referencing

Reuse

Extension

Reuse

Language Modularity, Composition and Reuse (LMR&C)

increase efficiency
of DSL development

4 ways of composition:

distinguished regarding
dependencies and fragment
structure

Dependencies:

do we have to know about the reuse when designing the languages?

Dependencies:

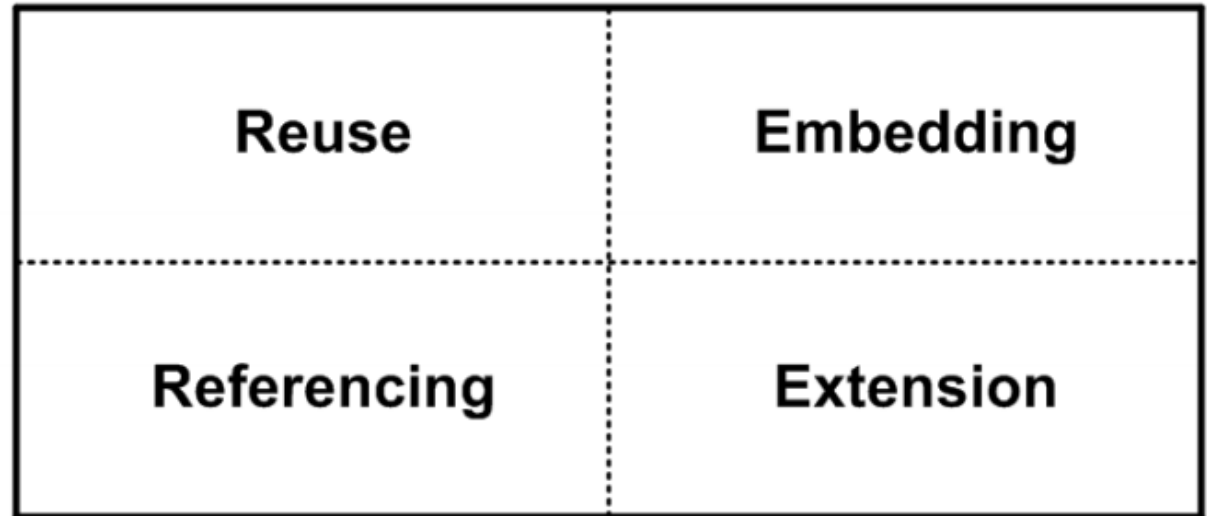
do we have to know about the reuse when designing the languages?

Fragment Structure:

homogeneous vs. heterogeneous
(„mixing languages“)

Dependencies & Fragment Structure:

independent
languages
dependencies
dependent



homogeneous

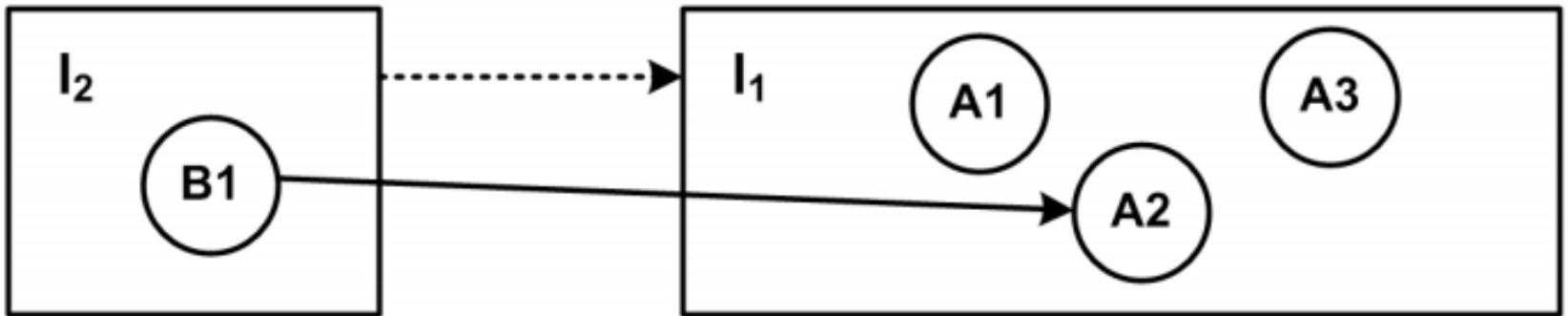
heterogeneous

fragment structure

Referencing

independent
languages
dependencies
dependent

| | |
|--------------------|---------------|
| Reuse | Embedding |
| Referencing | Extension |
| homogeneous | heterogeneous |
| fragment structure | |



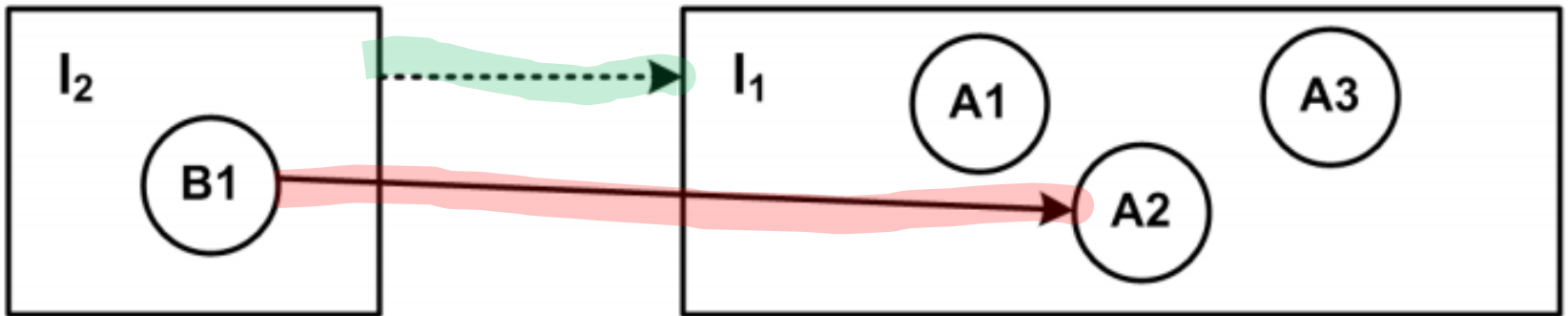
Referencing

independent
languages
dependencies
dependent

| | |
|-------------|-----------|
| Reuse | Embedding |
| Referencing | Extension |

homogeneous heterogeneous
fragment structure

Dependent



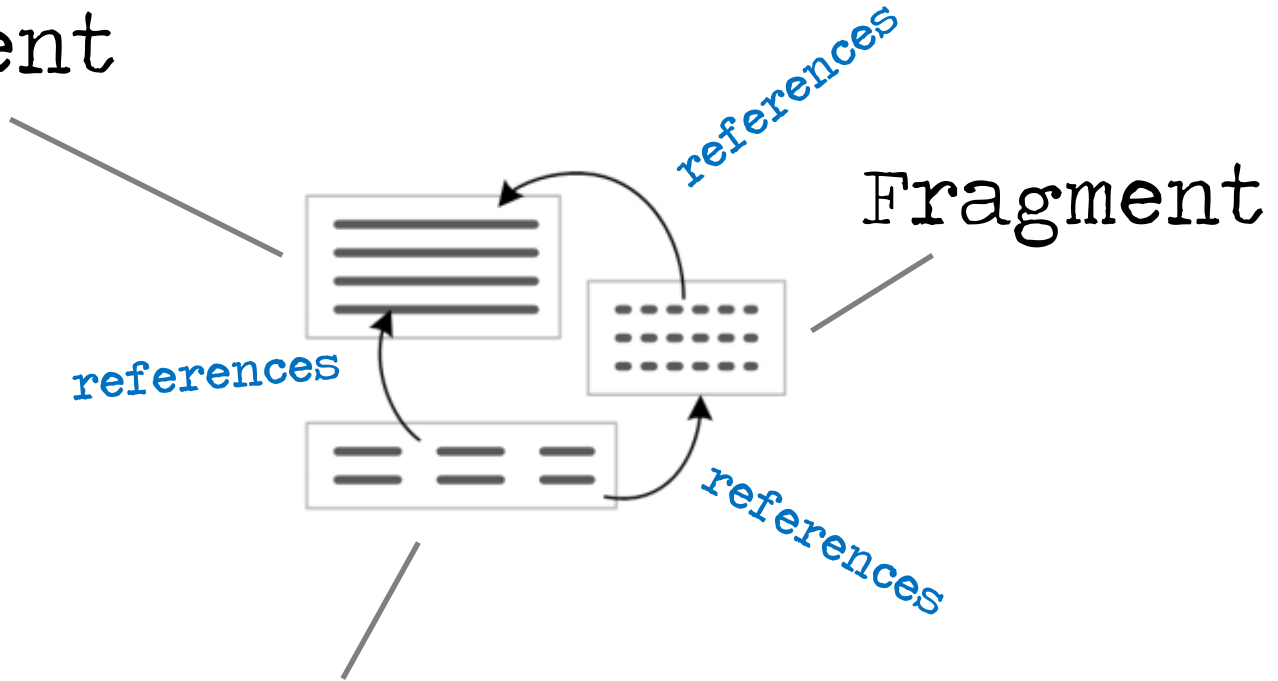
No containment

Referencing

Used in
viewpoints

Referencing

Fragment



Fragment

Fragment

Referencing

```
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

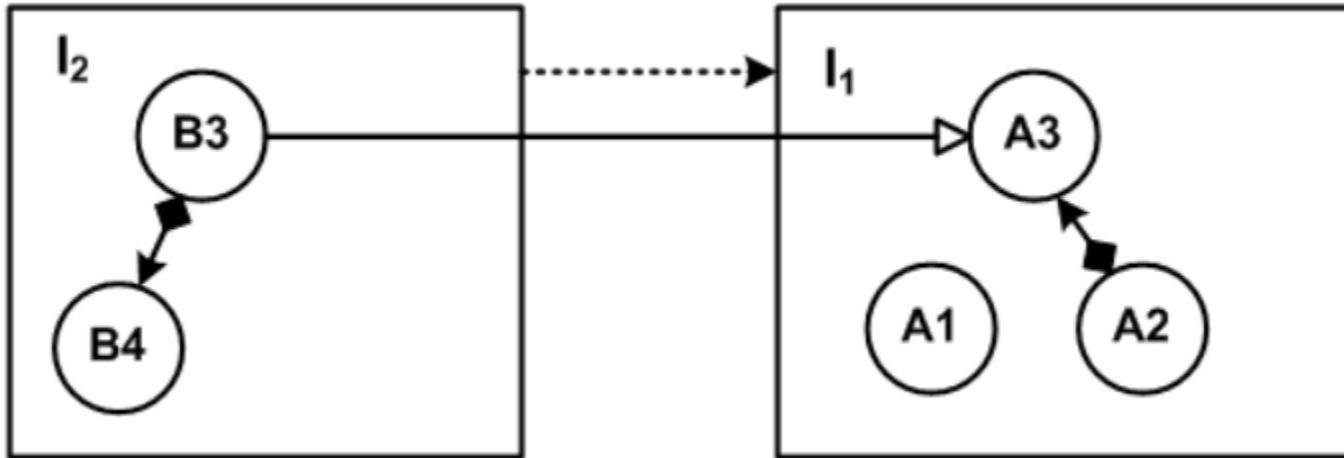
Refrige
rators

Extension

independent
languages
dependencies
dependent

| | |
|-------------|-----------|
| Reuse | Embedding |
| Referencing | Extension |

homogeneous heterogeneous
fragment structure



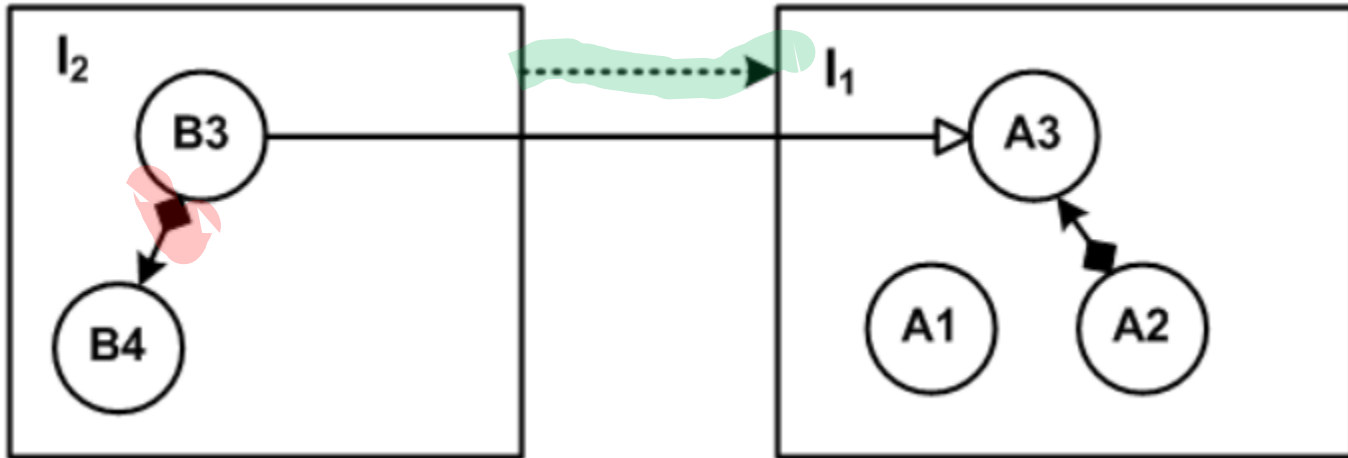
Extension

independent
languages
dependencies
dependent

| | |
|-------------|-----------|
| Reuse | Embedding |
| Referencing | Extension |

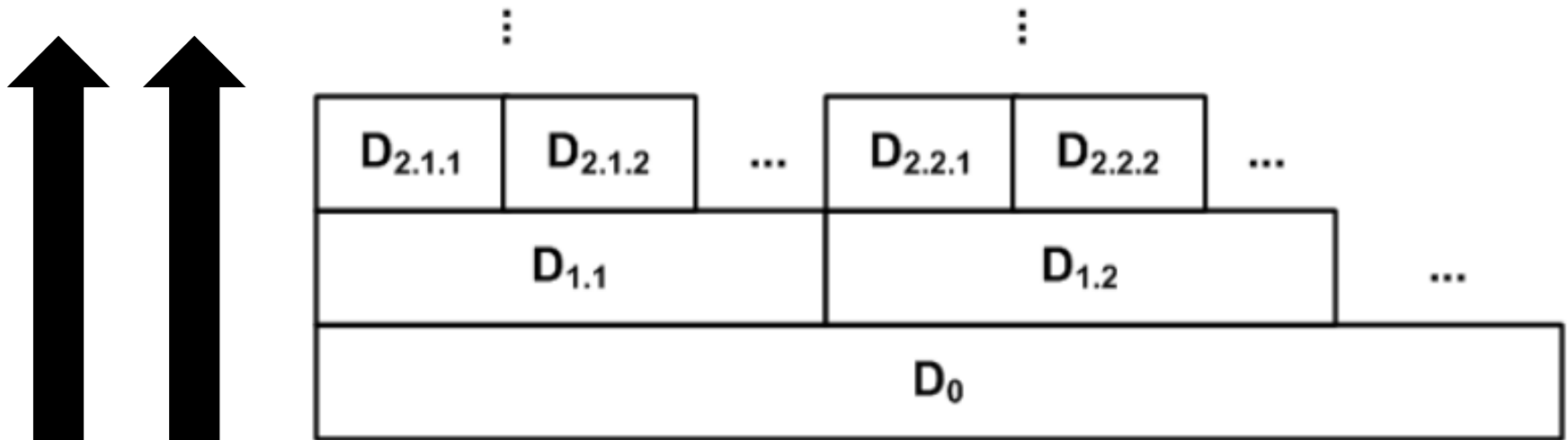
homogeneous heterogeneous
fragment structure

Dependent



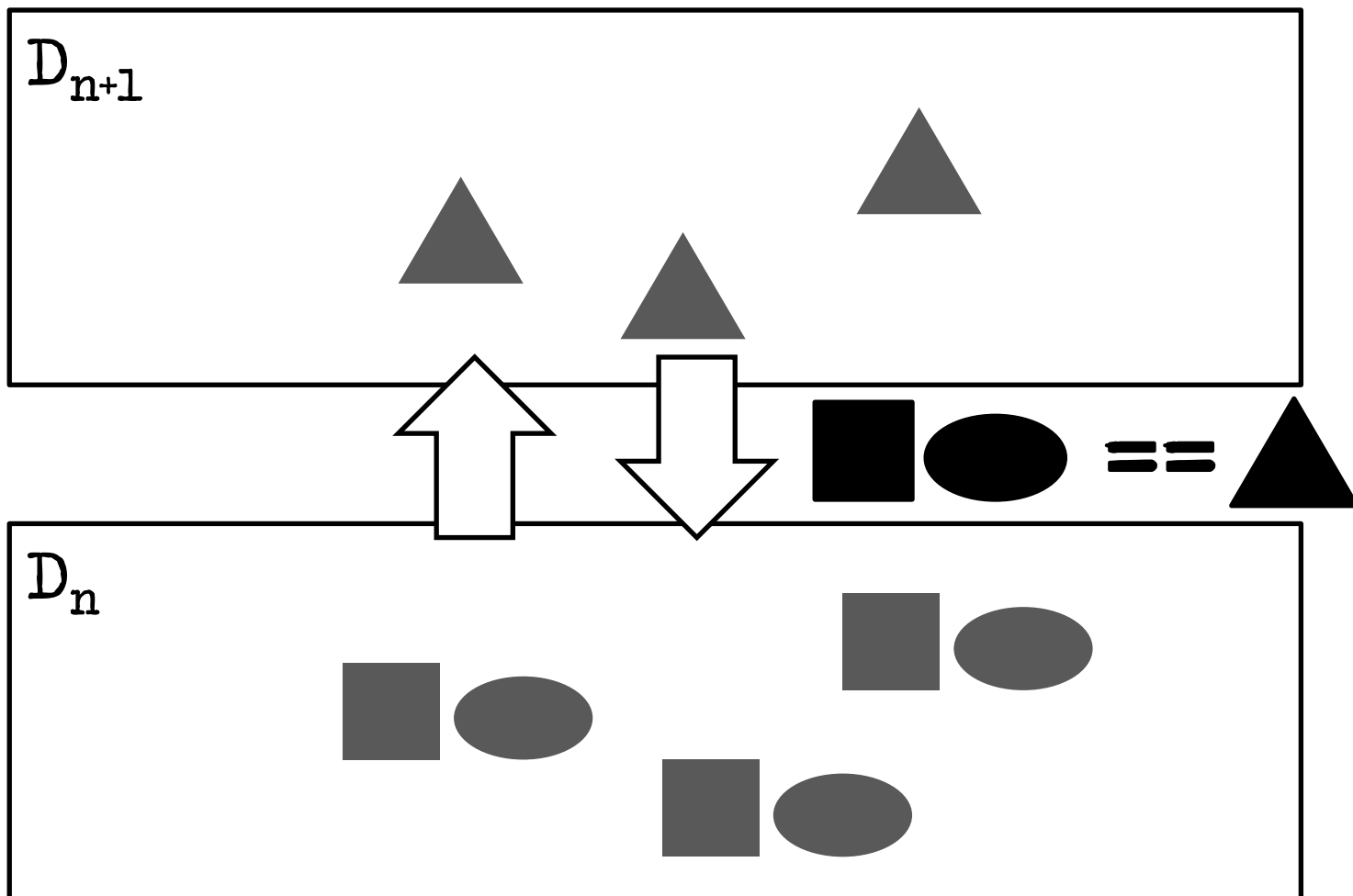
Containment

Extension

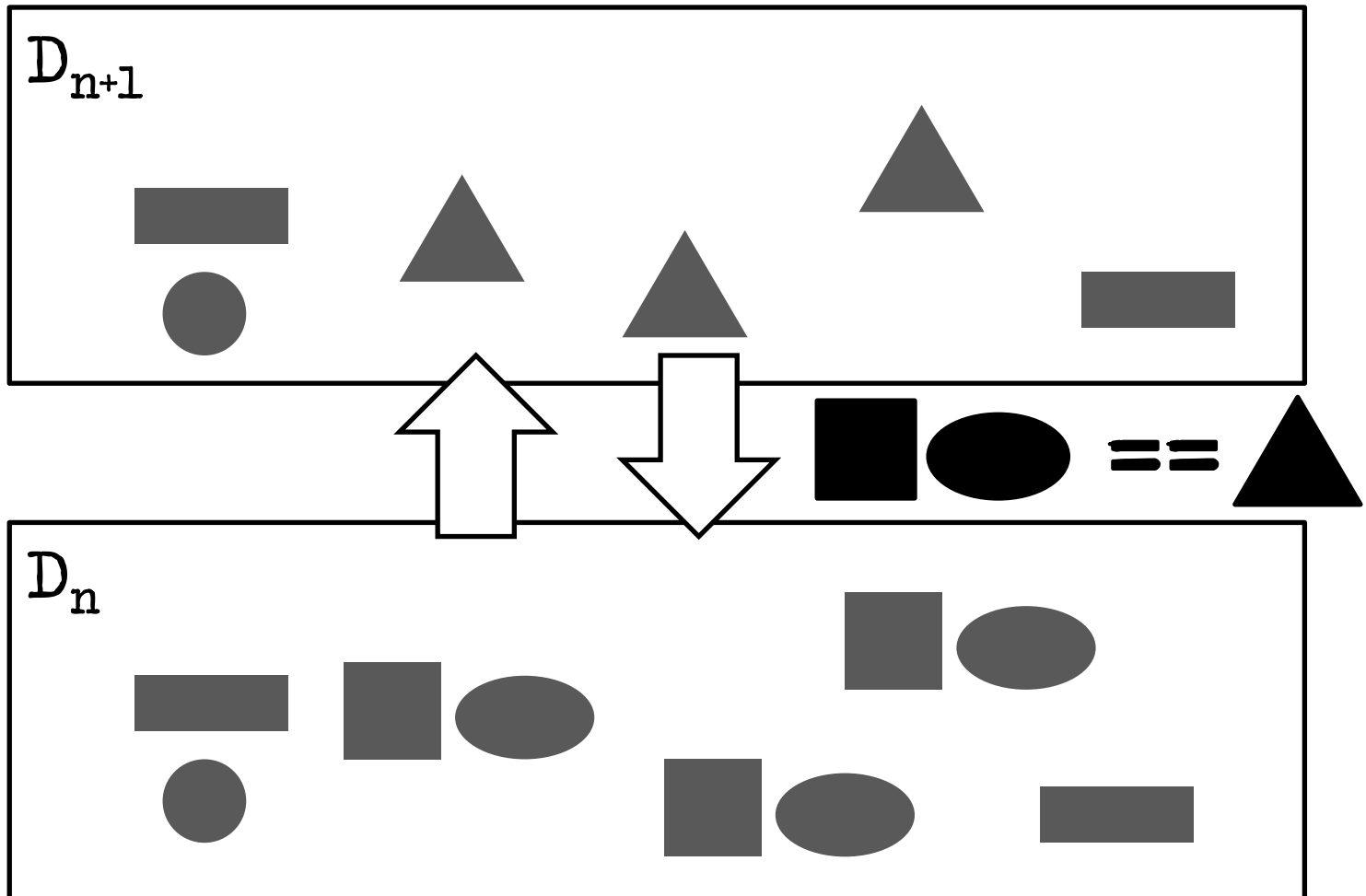


more specialized domains
more specialized languages

Extension

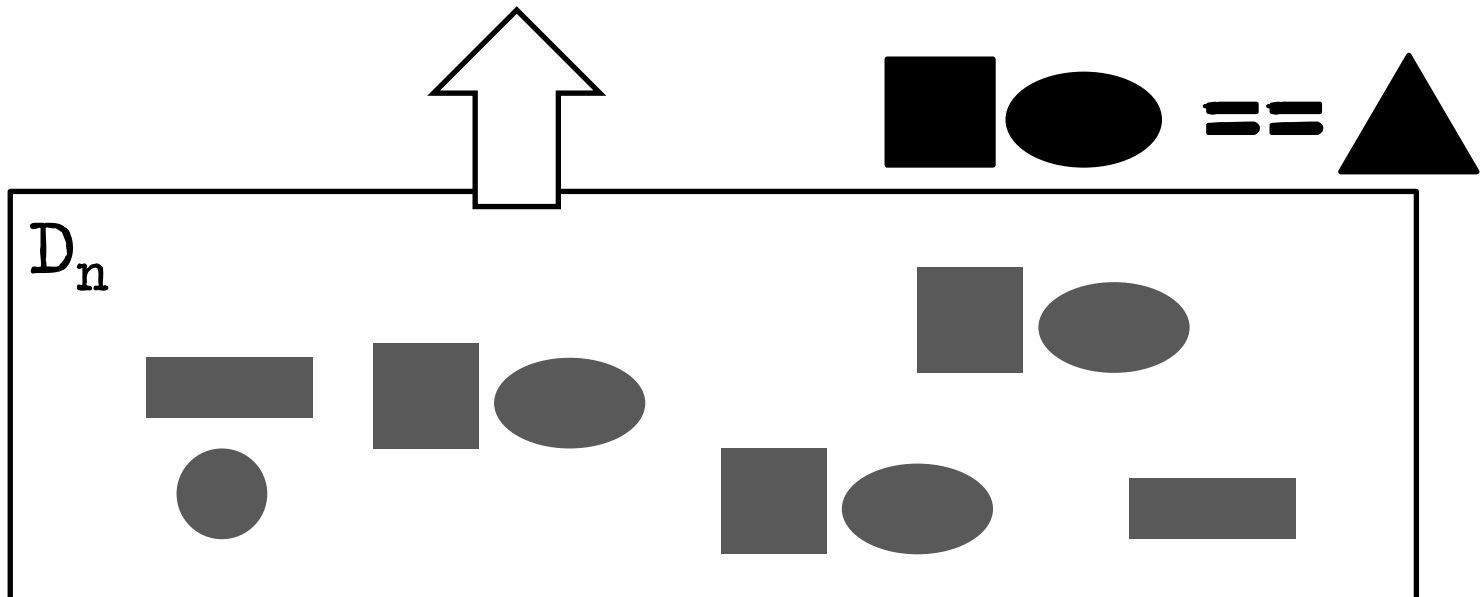


Extension



Extension

Good for **bottom-up** (inductive) domains, and for use by **technical** DSLs (people)



Extension

Drawbacks

tightly bound to base
potentially hard to analyze
the combined program

Extension

```
module main imports OsekKernel, EcAPI, BitLevelUtilities {

    constant int WHITE = 500;
    constant int BLACK = 700;
    constant int SLOW = 20;
    constant int FAST = 40;

    statemachine linefollower {
        event initialized;
        initial state initializing {
            initialized [true] -> running
        }
        state running { }
    }

    initialize {
        ecrobot_set_light_sensor_active
            (SENSOR_PORT_T::NXT_PORT_S1);
        event linefollower:initialized
    }

    terminate {
        ecrobot_set_light_sensor_inactive
            (SENSOR_PORT_T::NXT_PORT_S1);

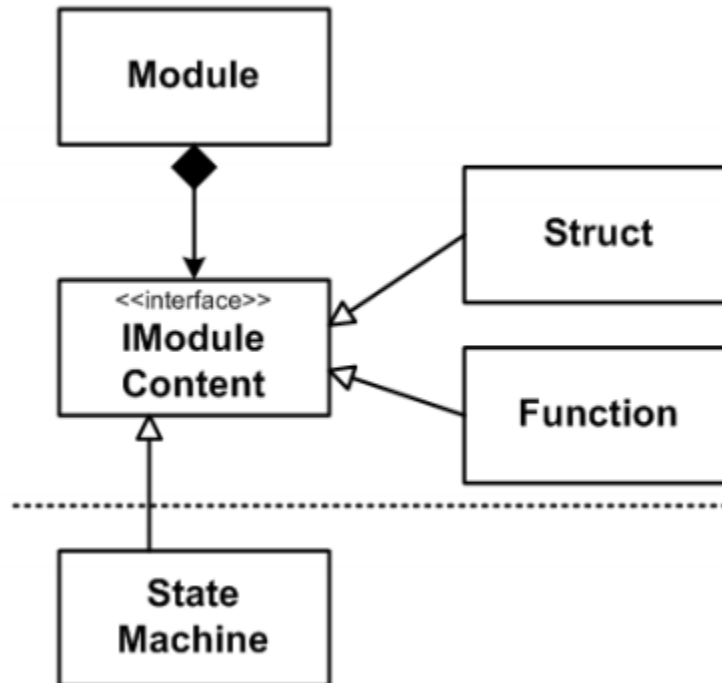
        task run cyclic prio = 1 every = 2 {
            stateswitch linefollower
                state running
                    int32 light = 0;
                    light = ecrobot_get_light_sensor
                        (SENSOR_PORT_T::NXT_PORT_S1);
                    if ( light < ( WHITE + BLACK ) / 2 ) {
                        updateMotorSettings(SLOW, FAST);
                    } else {
                        updateMotorSettings(FAST, SLOW);
                    }
                default
                    <noop>;
            }

            void updateMotorSettings( int left, int right ) {
                nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_M1, left);
                nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_M2, right);
            }
        }
    }
}
```

Example

Extended C

Extension



Example

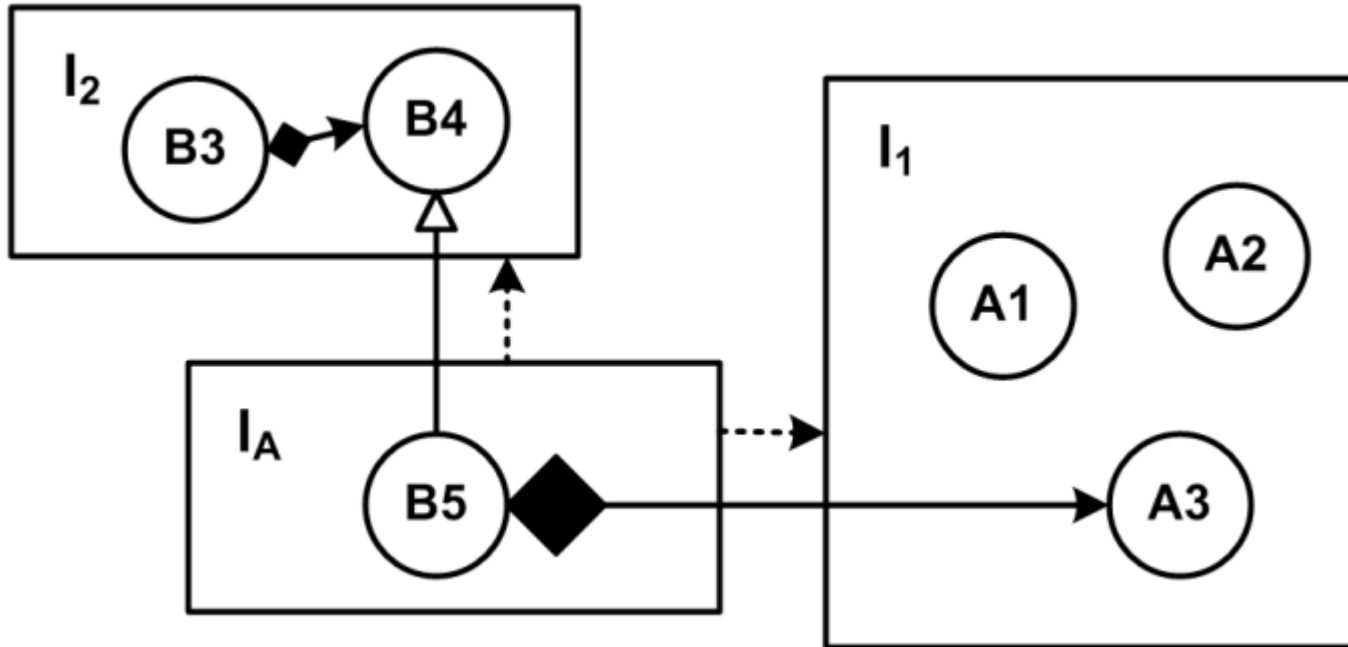
Extended C

Embedding

independent
languages
dependencies
dependent

| | |
|-------------|-----------|
| Reuse | Embedding |
| Referencing | Extension |

homogeneous heterogeneous
fragment structure

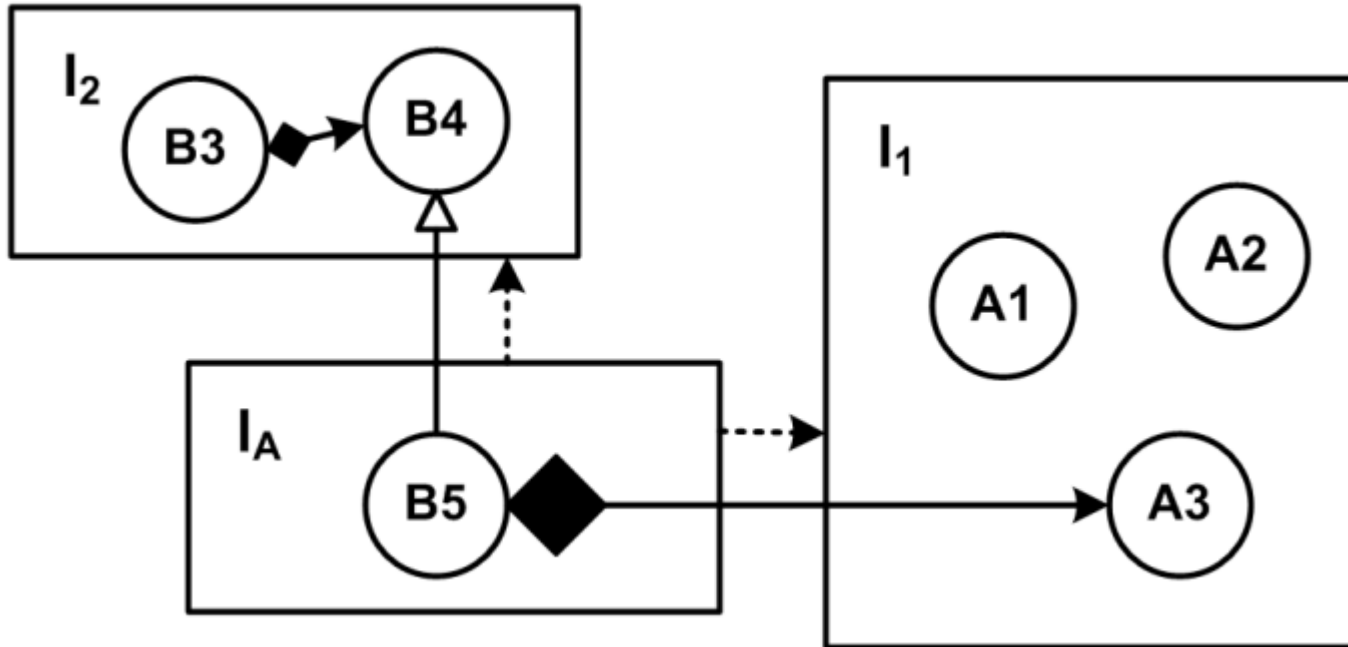


Embedding

independent
languages
dependencies
dependent

| | |
|-------------|-----------|
| Reuse | Embedding |
| Referencing | Extension |

homogeneous heterogeneous
fragment structure



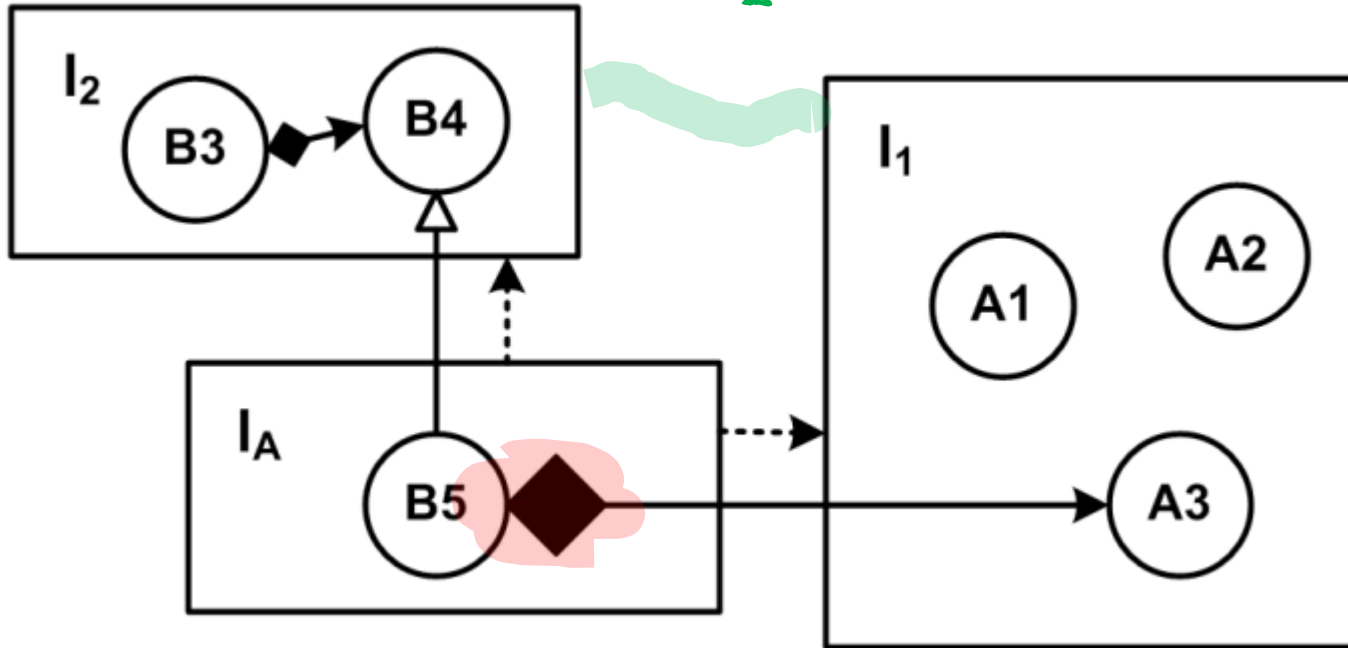
Embedding

independent
languages
dependencies
dependent

| | |
|-------------|-----------|
| Reuse | Embedding |
| Referencing | Extension |

homogeneous heterogeneous
fragment structure

Independent



Containment

Embedding

The screenshot shows the Capgemini Pension Workbench interface. The main window displays the 'Rule Bereken Mutatieperiode' with its documentation and test cases. The test cases table is as follows:

| Name | Valid time | Transaction time | Fixture | Product | Element | Expected value | Actual value |
|----------------|------------|------------------|--|---------|---------|----------------|--------------|
| Gelijke datums | 03/01/2008 | | Mutatieperiode - Mutatiedatum = Mutatiedatum Vorig | | | 3 | 0 |
| Periode < 30 | 03/01/2008 | | Mutatieperiode - Mutatiedatum > Mutatiedatum Vorig (binnen 1 maand) | | | 15 | 15 |
| Periode > 30 | 03/01/2008 | | Mutatieperiode - Mutatiedatum > Mutatiedatum Vorig (meerdere maanden) | | | 60 | |

The interface also shows a 'Table of Contents' on the left with a tree view of 'Library', 'Documentation', 'Foundation', 'Value sets...', 'Tag definitions', and 'Shared' elements.

Example

Pension
Plans

Embedding

Embedding often uses
Extension to extend the
embedded language to adapt it
to its new context.

Concrete Syntax

expressivity
coverage
semantics
separation of
concerns

completeness
paradigms
modularity
concrete
syntax

process

UI for the language!

Important for acceptance by users!

Textual

Symbolic

Tabular

Graphical



Reuse existing
syntax of
domain, if any!

Tools let you
freely combine
all kinds.

Default: Text

Editors simple to build

Productive

Easy to integrate w/ tools

Easy to evolve programs

Default: Text

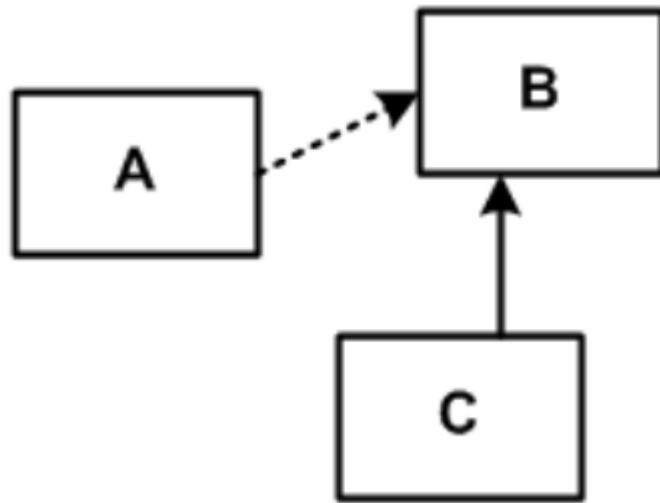
Editors simple to build
Productive

Easy to integrate w/ tools

Easy to evolve programs

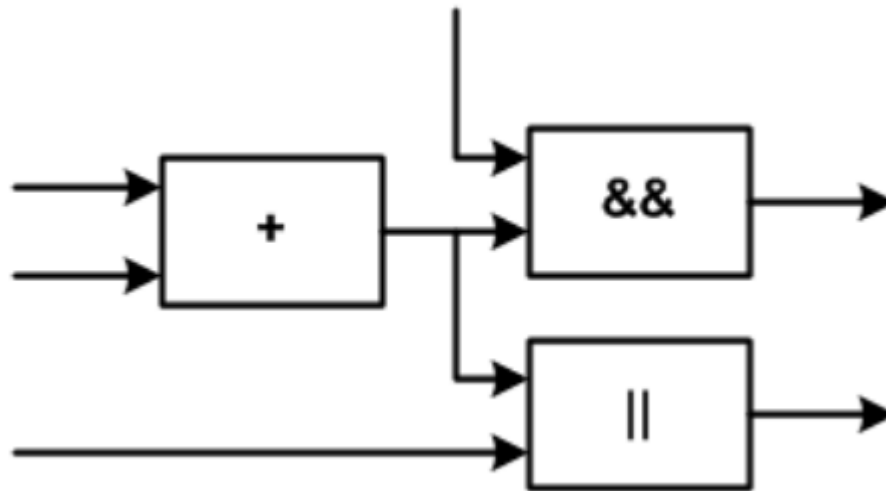
... then add other forms,
if really necessary

Graphical in case..



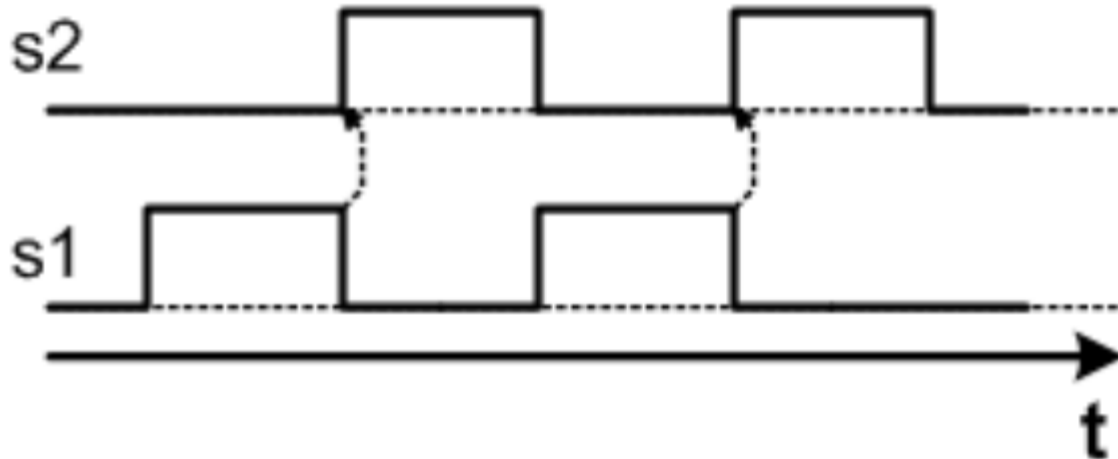
Relationships

Graphical in case..



Flow and
Dependency

Graphical in case...



Causality
and Timing

Symbolic

Either
Mathematical,
or often
highly
inspired by
domain

▣ *3.3 Commutatietallen op 1 leven*

$$D_x = v^x * \frac{l}{100} \quad \approx 6 \text{ Dec (3)}$$

Implemented in [V9401](#)

¶

$$N_x = \sum_{t=0}^{\omega-x} D_{x+t} \quad \approx 7 \text{ Dec (3)}$$

¶

▣ *3.6 Contante waarde 1 leven/ 2 levens*

$$E_x = \frac{D_{x+n}}{D_x} \quad \approx 19 \text{ Dec (4)}$$

¶

$$a_x = \ddot{a}_x - 1 \quad \approx 21 \text{ Dec (3)}$$

¶

$$\bar{a}_x = \ddot{a}_x - 0,5 \quad \approx 22 \text{ Dec (3)}$$











¶

$$\ddot{a}_{x:n} = \frac{N_x - N_{x+n}}{D_x} \quad \approx 23 \text{ Dec (3)}$$

$$\bar{a}_{x:n} = \ddot{a}_{x:n} - 0,5 + 0,5 * E_{n|x} \quad \approx 25 \text{ Dec (3)}$$

¶

Tables

| Name | Documentation | Tags | Valid time | Transaction time | Fixture | Product | Element | Expected value | Actual value |
|------------------------------|---------------|---|------------|------------------|---------------|-----------------|------------------------------------|----------------|--------------|
| Accrued right at retireme | |  | 2006-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued right | 761.0402 | 761.0402 |
| Accrued Right last final pay | |  | 2004-1-1 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued right | 705.0589 | 705.0589 |
| premium last year | |  | 2006-1-1 | 2007-9-24 | Jan De Jong | Old Age Pension | Premium old age pension | 329.0625 | 329.0625 |
| Accrued right at retireme 2) | |  | 2006-12-31 | 2007-9-24 | Piet Van Dijk | Old Age Pension | Accrued right | 740.94 | 724.7658 |
| | |  | 1985-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued Right in service period | 73.661 | 73.661 |
| | |  | 1985-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Years of service in service period | 3.7534 | 3.7534 |
| | |  | 1987-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Pension base average FP | 7750 | 7750 |
| | |  | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Accrued Right in service period | 387.7449 | 387.7449 |
| | |  | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Years of service in service period | 10.8082 | 10.8082 |
| | |  | 1998-12-31 | 2007-9-24 | Jan De Jong | Old Age Pension | Pension base average FP | 8250 | 8250 |

Combinations

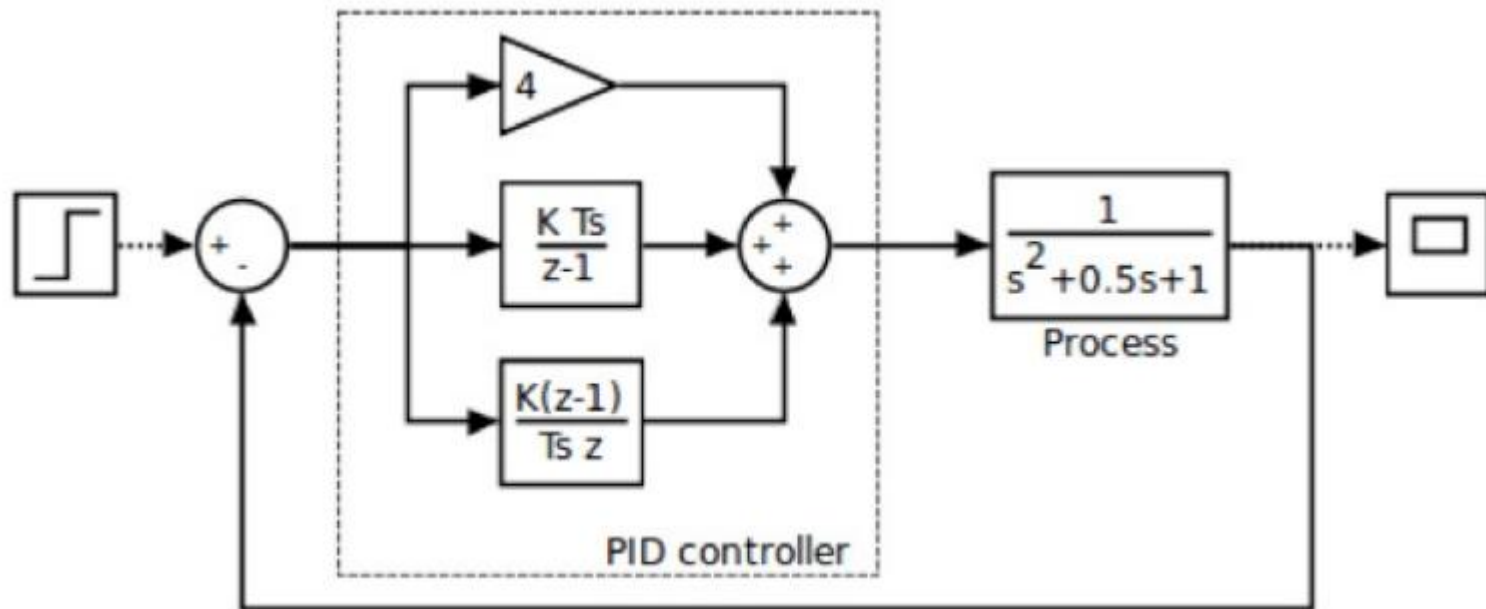
```
c/s interface Decider {
  int decide(int x, int y) pre
}

component AComp extends nothing {
  ports:
    provides Decider decider
  contents:
    int decide(int x, int y) <- op decider.decide {
      return int, 0

```

| | | | |
|--------|--------|-------|---|
| | x == 0 | x > 0 | ; |
| y == 0 | 0 | 1 | |
| y > 0 | 1 | 2 | |

Combinations



Combinations

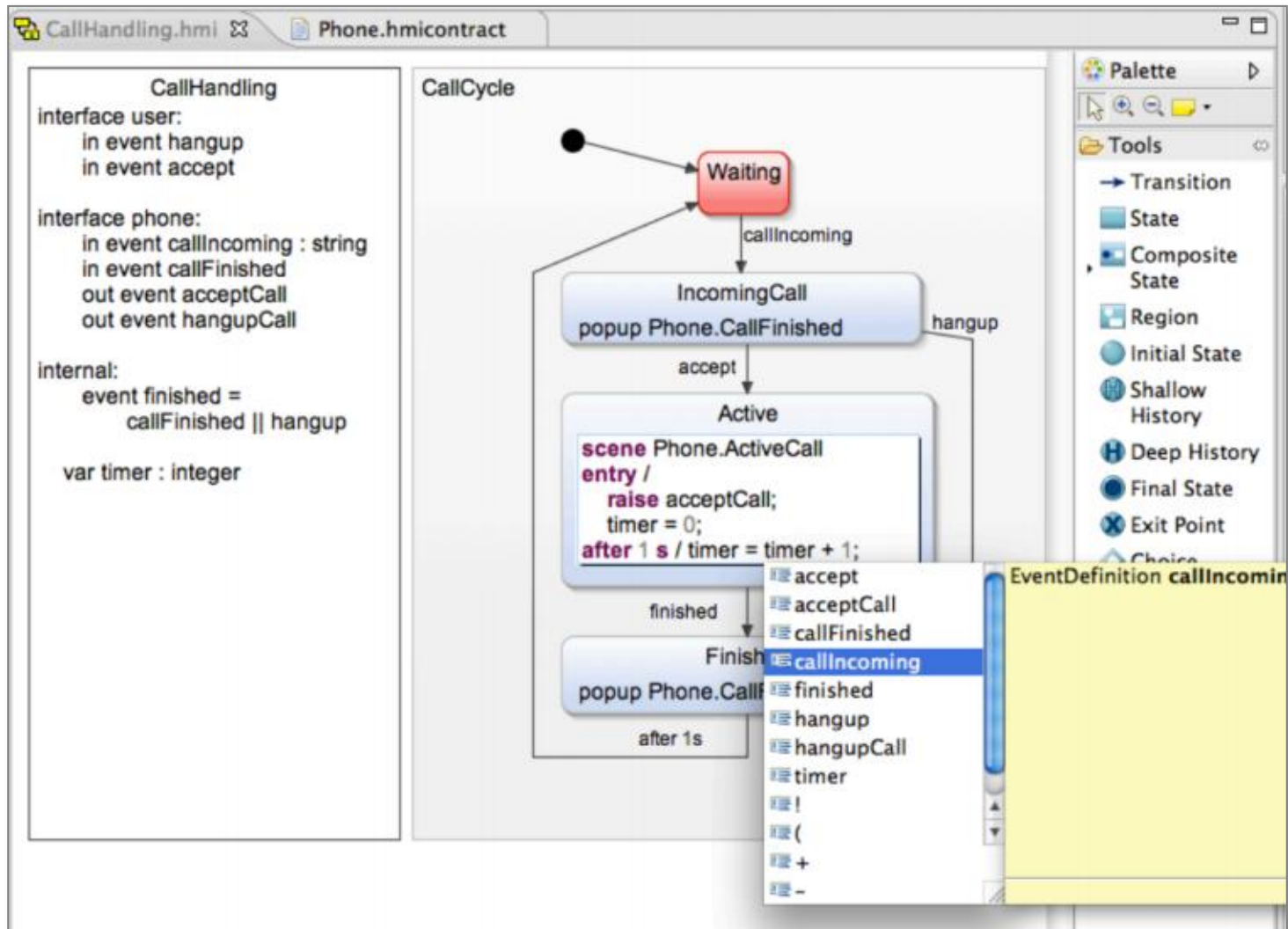
The screenshot shows a software interface with two tabs: "*RIF Model.rif-xmi" and "*Specification Document". The active tab is "Specification Document", which contains a table with two columns: "Description" and "Link".

| | Description | Link |
|---|---|------|
| 1 | system SHALL display speed system SHALL display rpm delay is less than "5" rpm is greater than | |

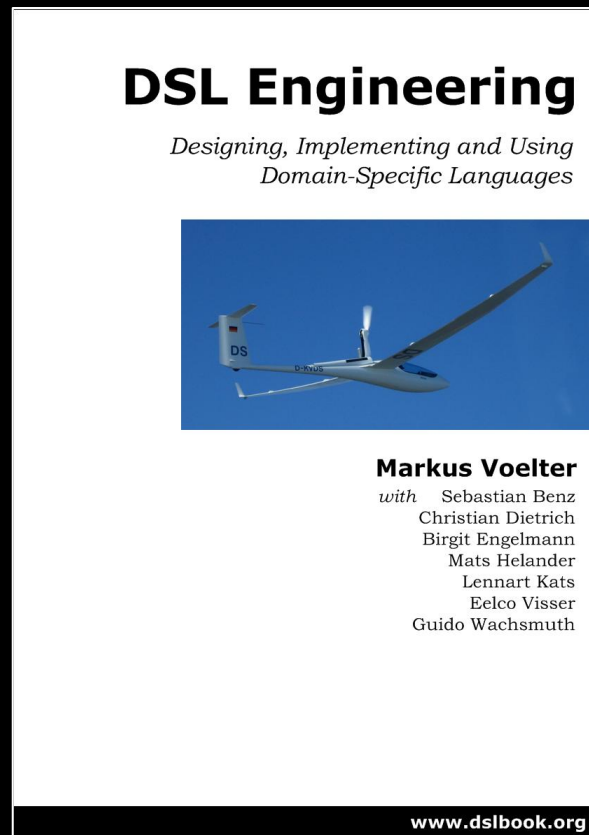
A dropdown menu is open over the first row of the table, showing a list of modal verbs and operators:

- SHALL
- and
- is disabled
- is enabled
- is equal to
- is greater than
- is less than
- is not equal to
- or
- xor

Combinations



The End.



@markusvoelter
+Markus Voelter