

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/>

This material is
based on this book:

<http://dslbook.org>

available Feb 2013



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

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.

Concepts (abstract syntax)

(concrete) Syntax

semantics (generators)

Tools and IDE

	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

General Purpose

C

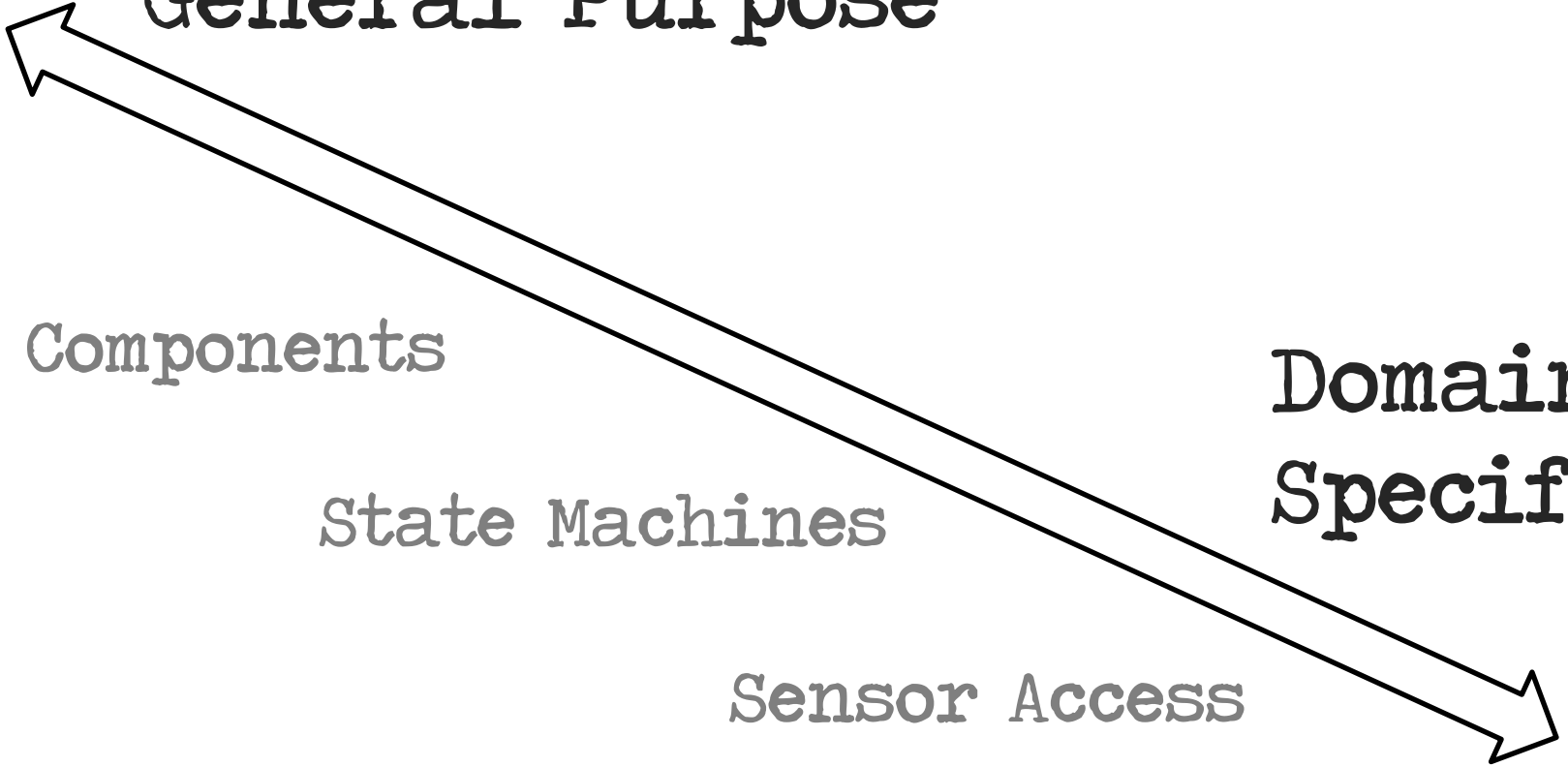
Components

State Machines

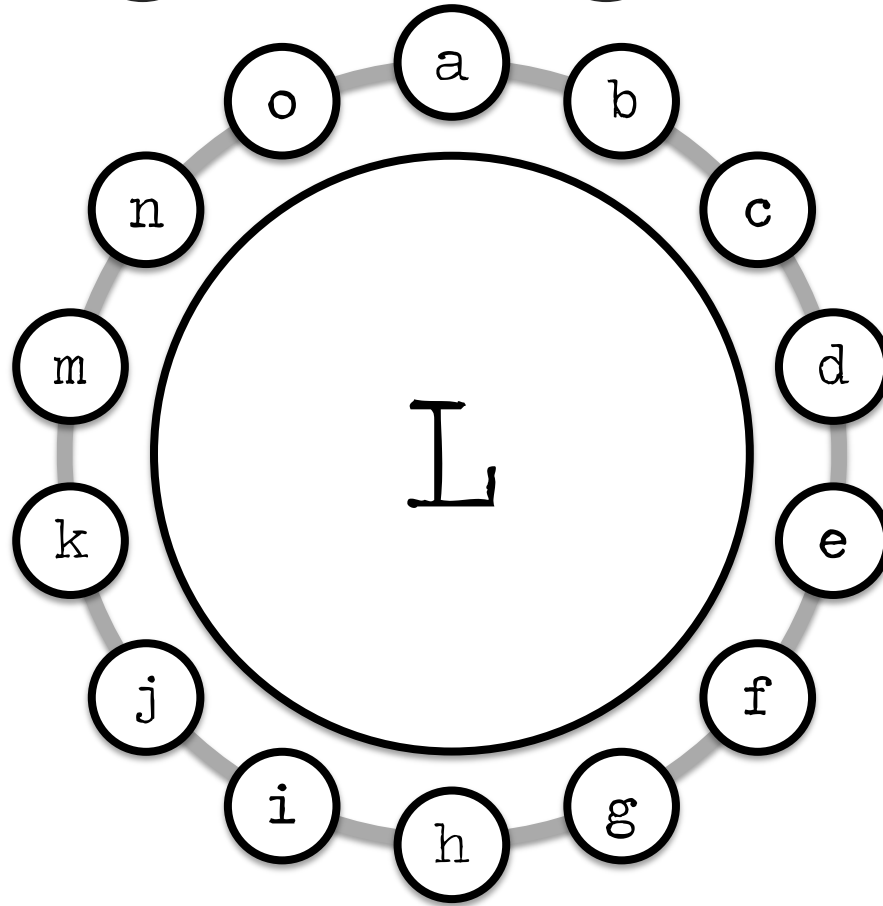
Sensor Access

**Domain
Specific**

LEGO Robot
Control

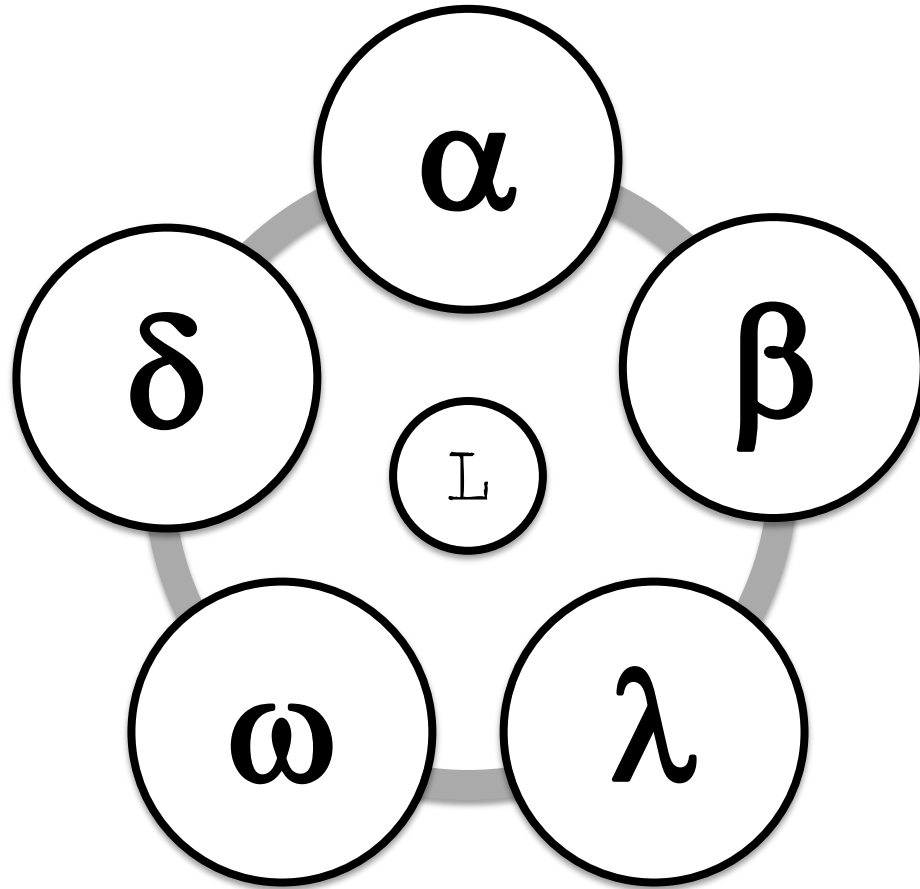


Big Language



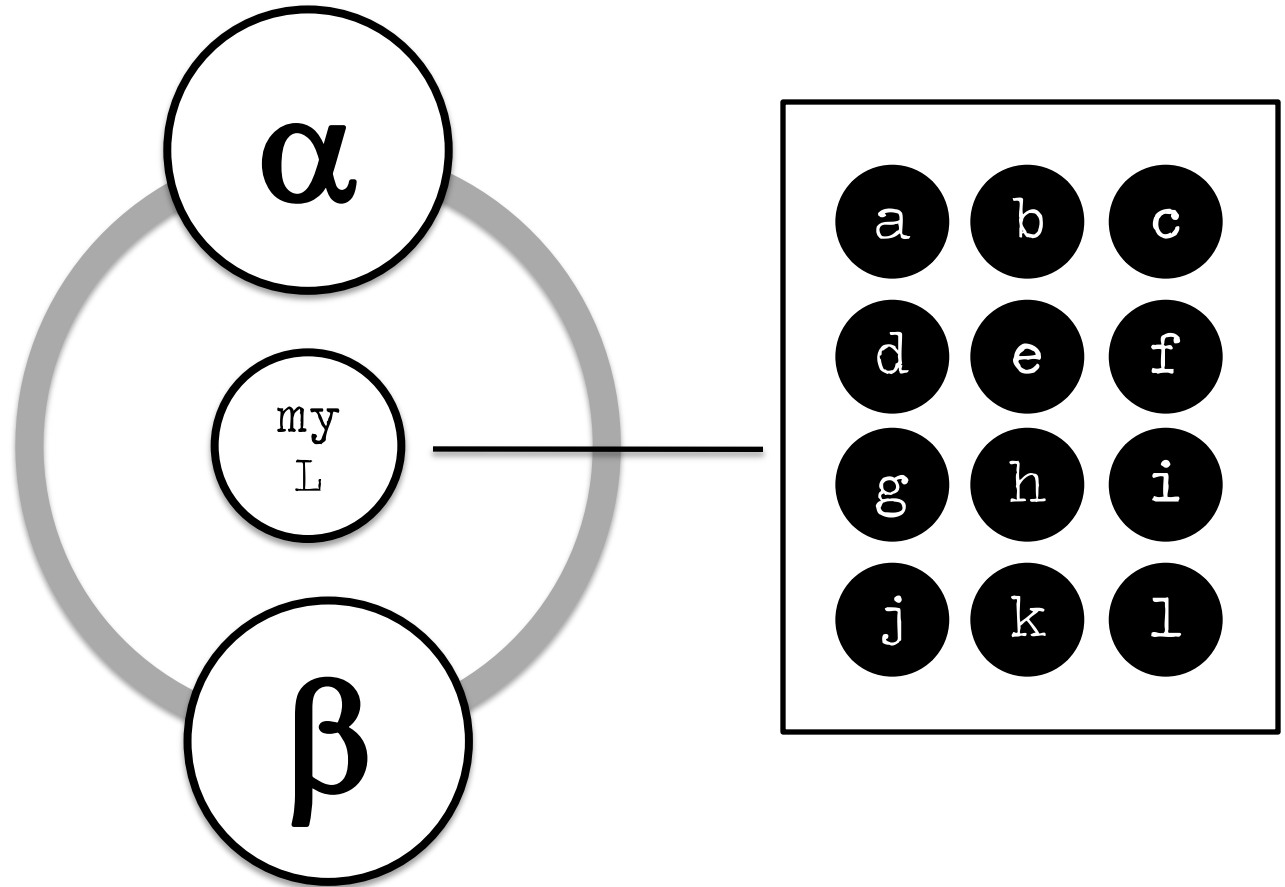
with many first
class concepts!

Small Language



with a few, orthogonal
and powerful concepts

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 Groottebepalingsmethode
 - 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

Model

A schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics

Model

A representation of a set of components of a process, system, or subject area, generally developed for understanding, analysis, improvement, and/or replacement of the process

Model

an abstraction or
simplification of
reality

Model

which ones?

an abstraction or
simplification of
/ reality

what should
we leave out?

Model Purpose

- ... code generation
- ... analysis and checking
- ... platform independence
- ... stakeholder integration
- ... drives design of language!

Model Purpose

- ... code generation
- ... analysis and checking
- ... platform independence
- ... stakeholder integration

Example

Components

Model Purpose

- ... code generation
- ... analysis and checking
- ... platform independence
- ... stakeholder integration

Example

Refrige
rators

Model Purpose

- ... code generation
- ... analysis and checking
- ... platform independence
- ... stakeholder integration

Example

Extended C

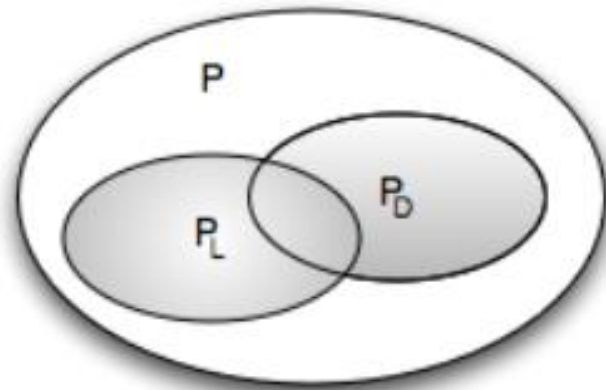
Model Purpose

- ... code generation
- ... analysis and checking
- ... platform independence
- ... stakeholder integration

Example

Pension
Plans

Programs Languages Domains



body of
knowledge
in the real
world

deductive
top down



Domain

existing
software
(family)

inductive
bottom up



deductive
top down



body of
knowledge
in the real
world

Domain

Example

Example

Penion
Plans

Refrige
rators

Domain

existing
software
(family)



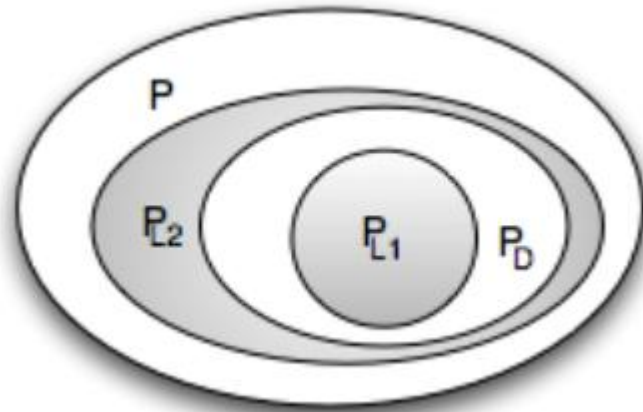
inductive
bottom up

Example

Example

Exten
ded C

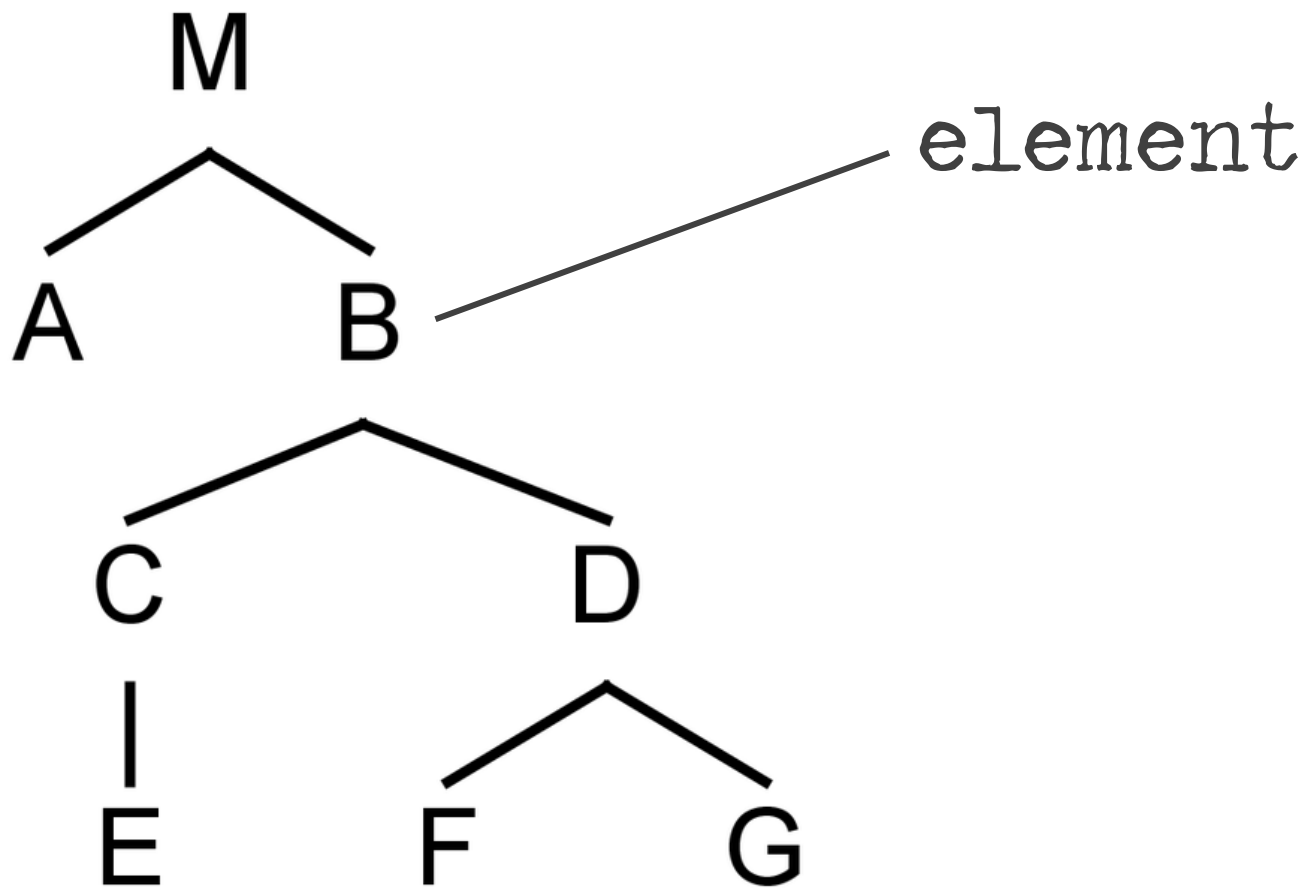
Compo
netns



A DSL L_D for D is a language that is specialized to encoding P_D programs.

more efficient
smaller

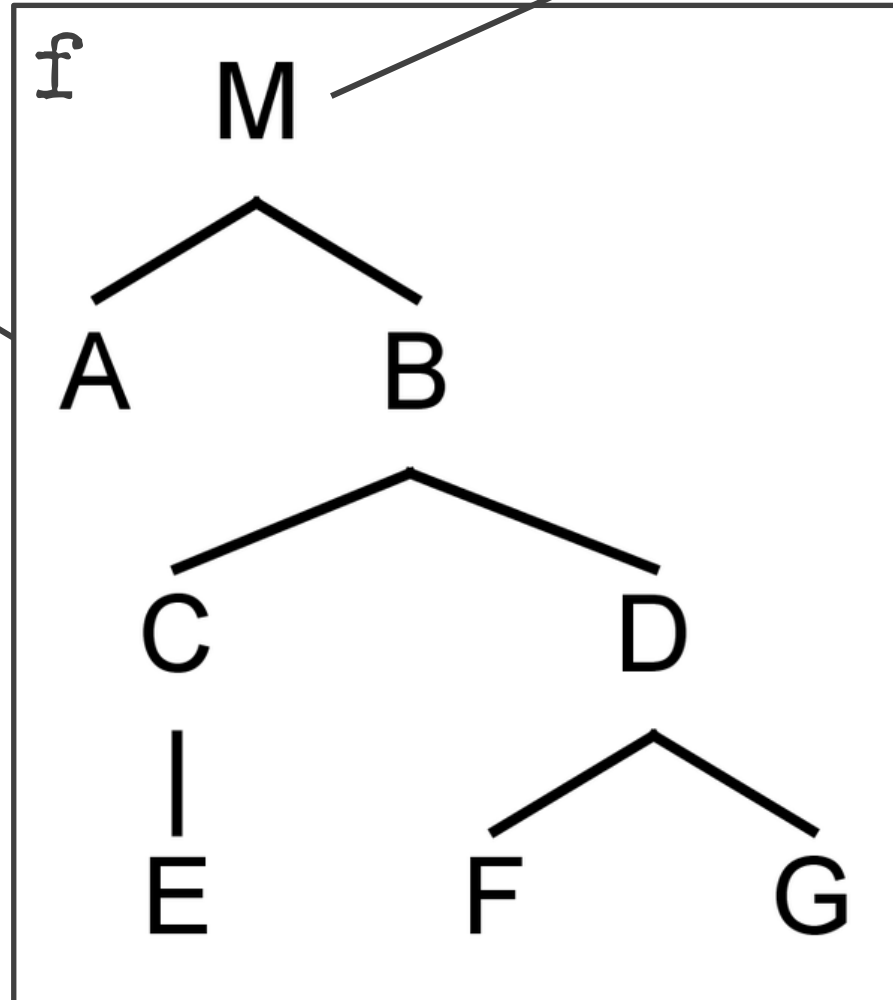
Programs
are trees



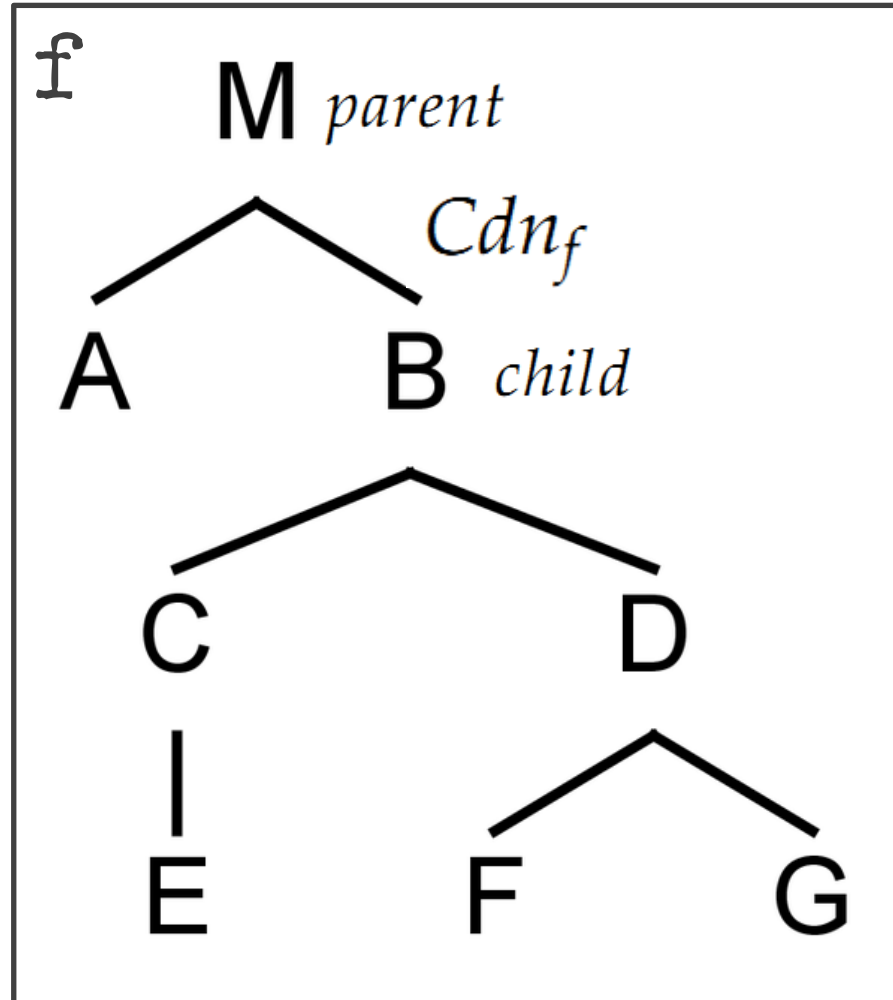
Fragments are
subtrees w/ root

fragment root

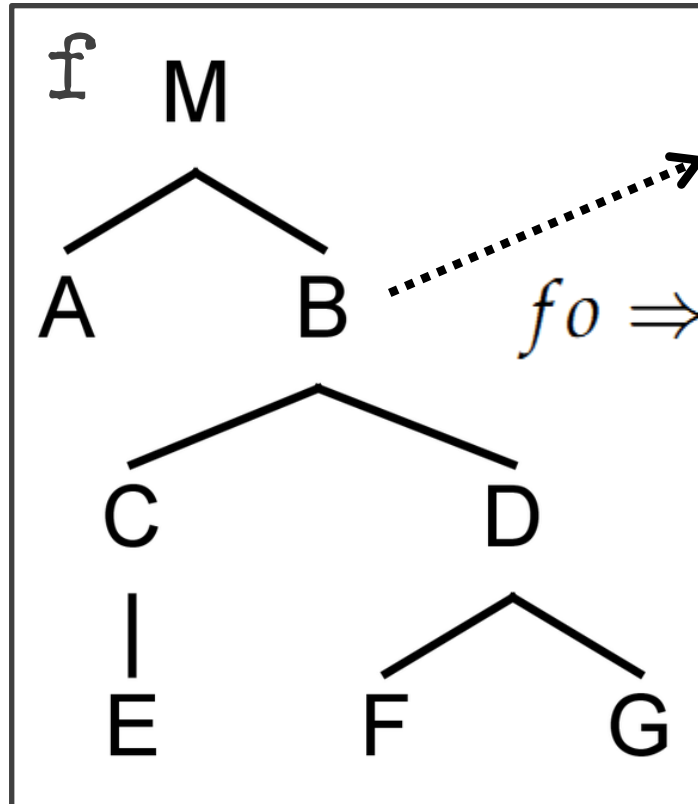
fragment



Parent-Child Relation

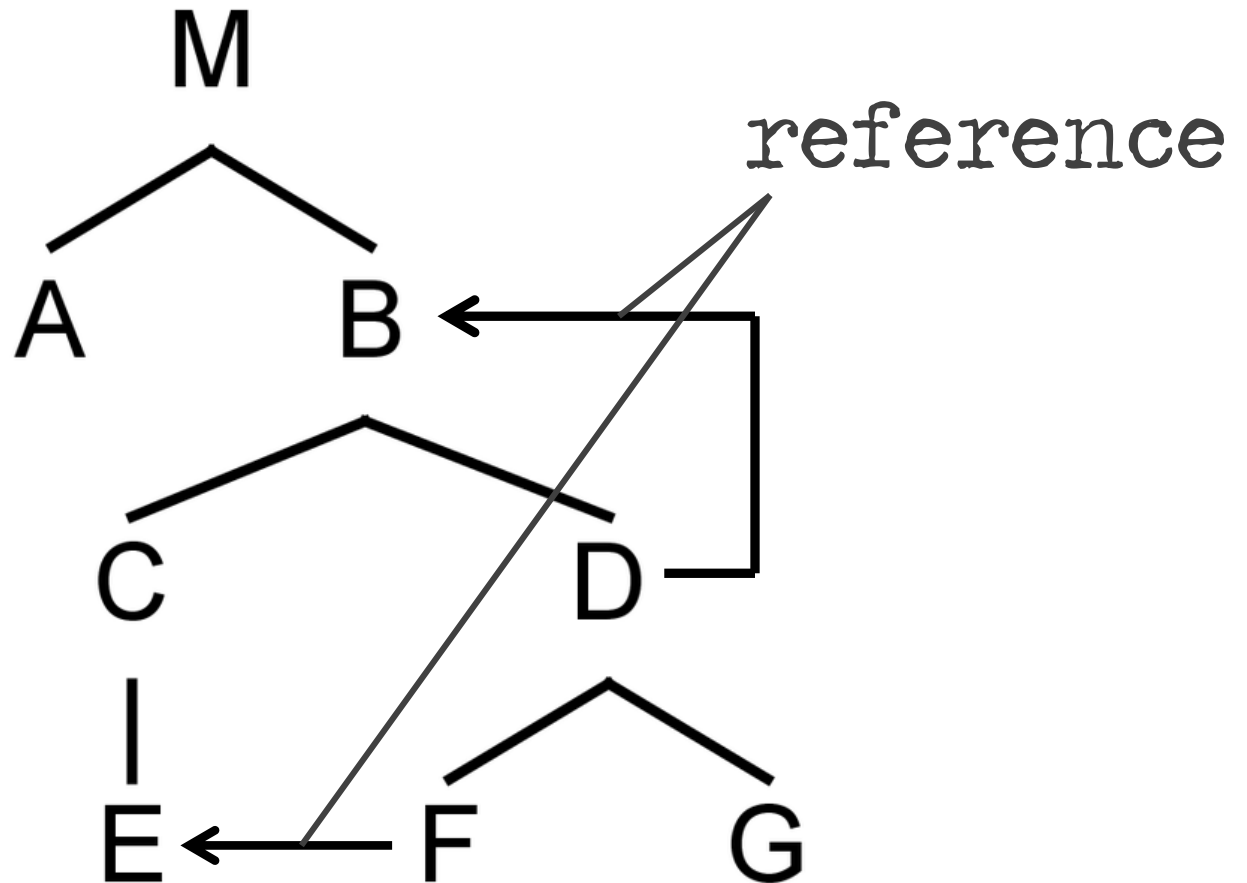


Programs and Fragments

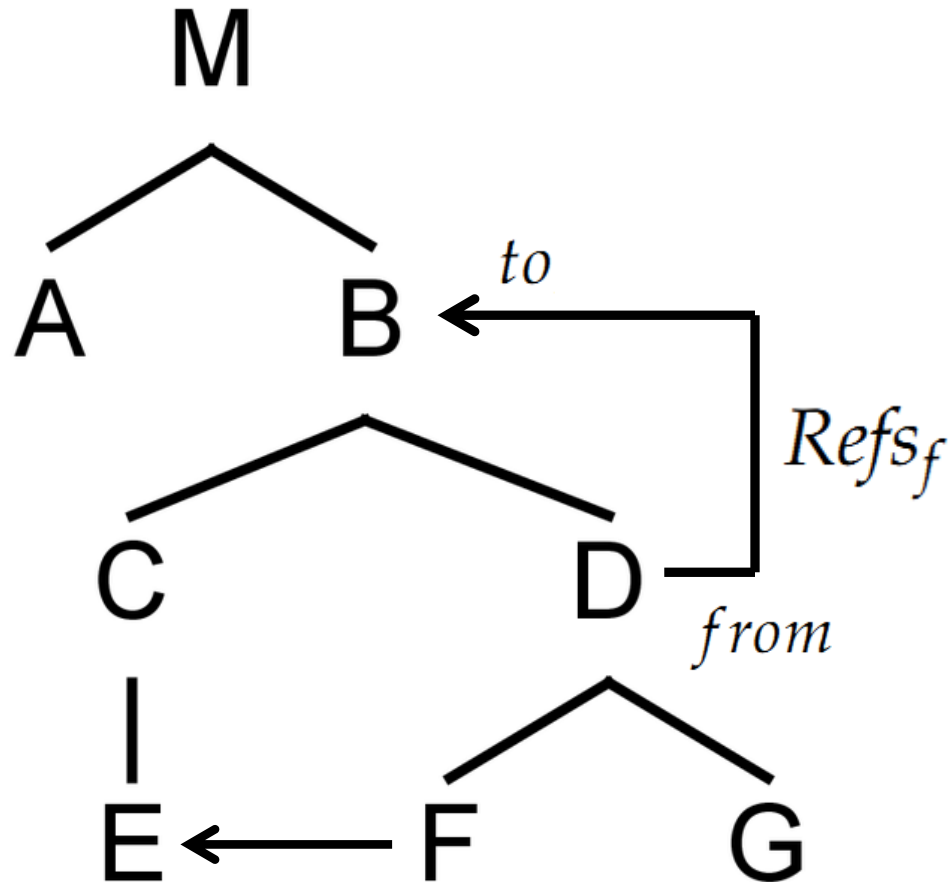


$fo \Rightarrow \text{element} \rightarrow \text{fragment}$

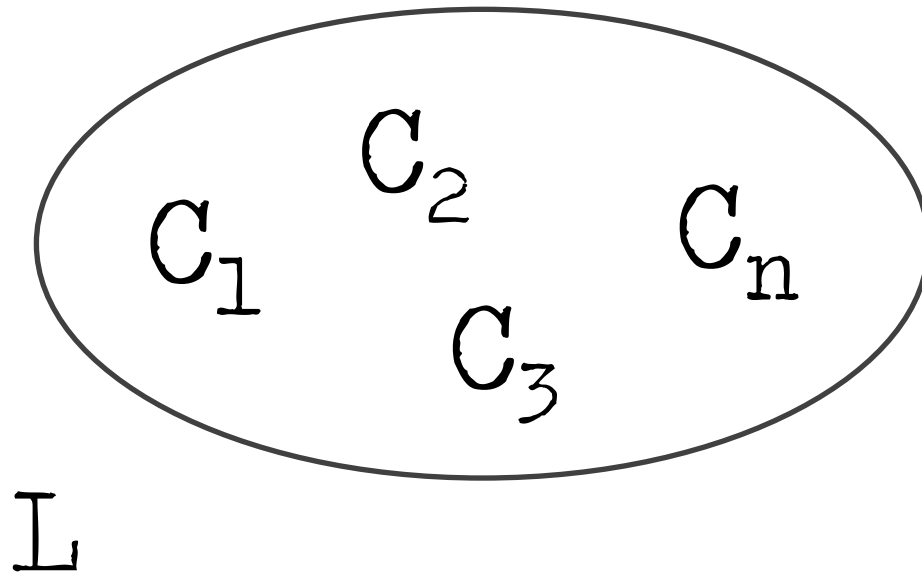
Programs are
graphs, really.



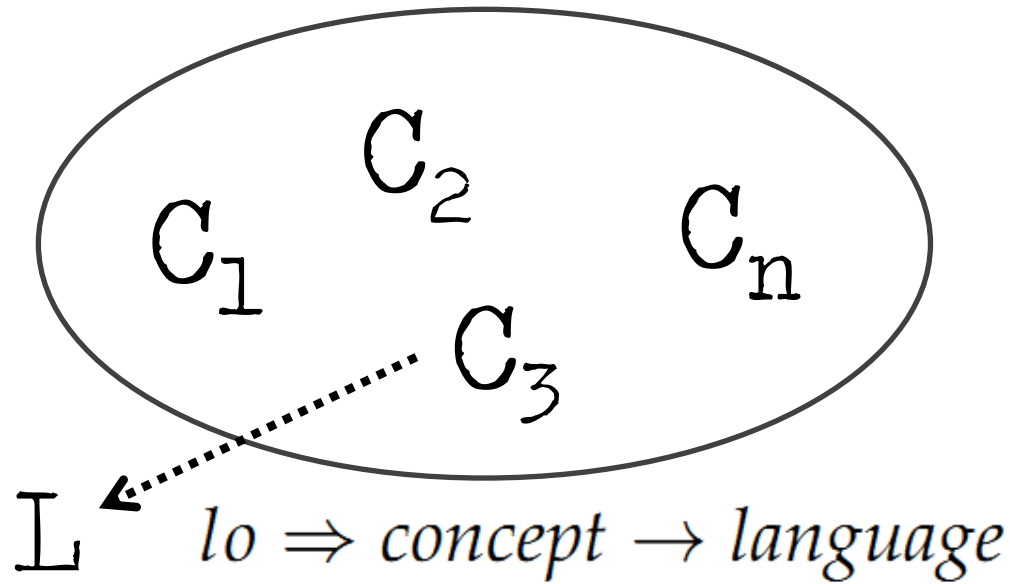
Programs are
graphs, really.



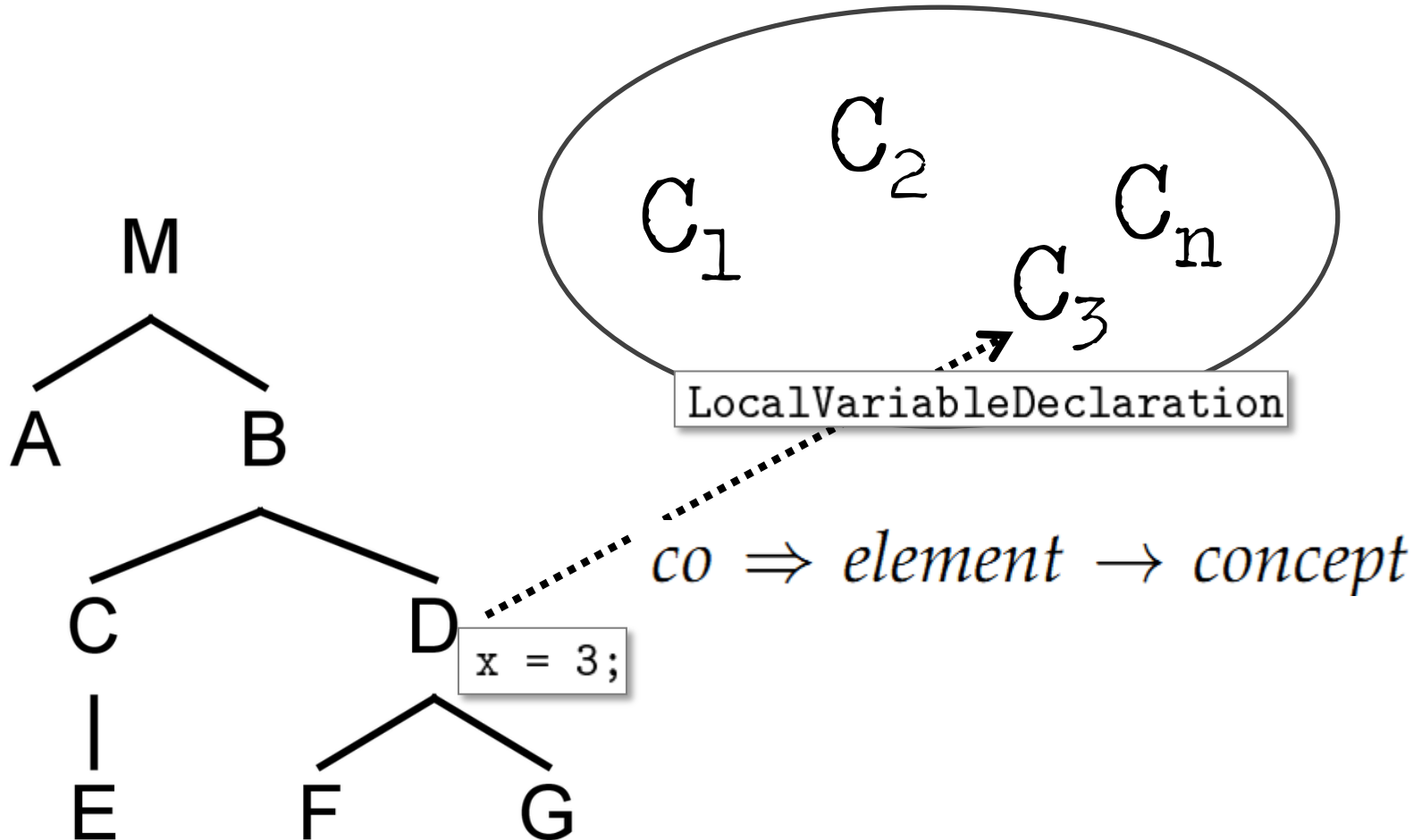
Languages are
sets of concepts



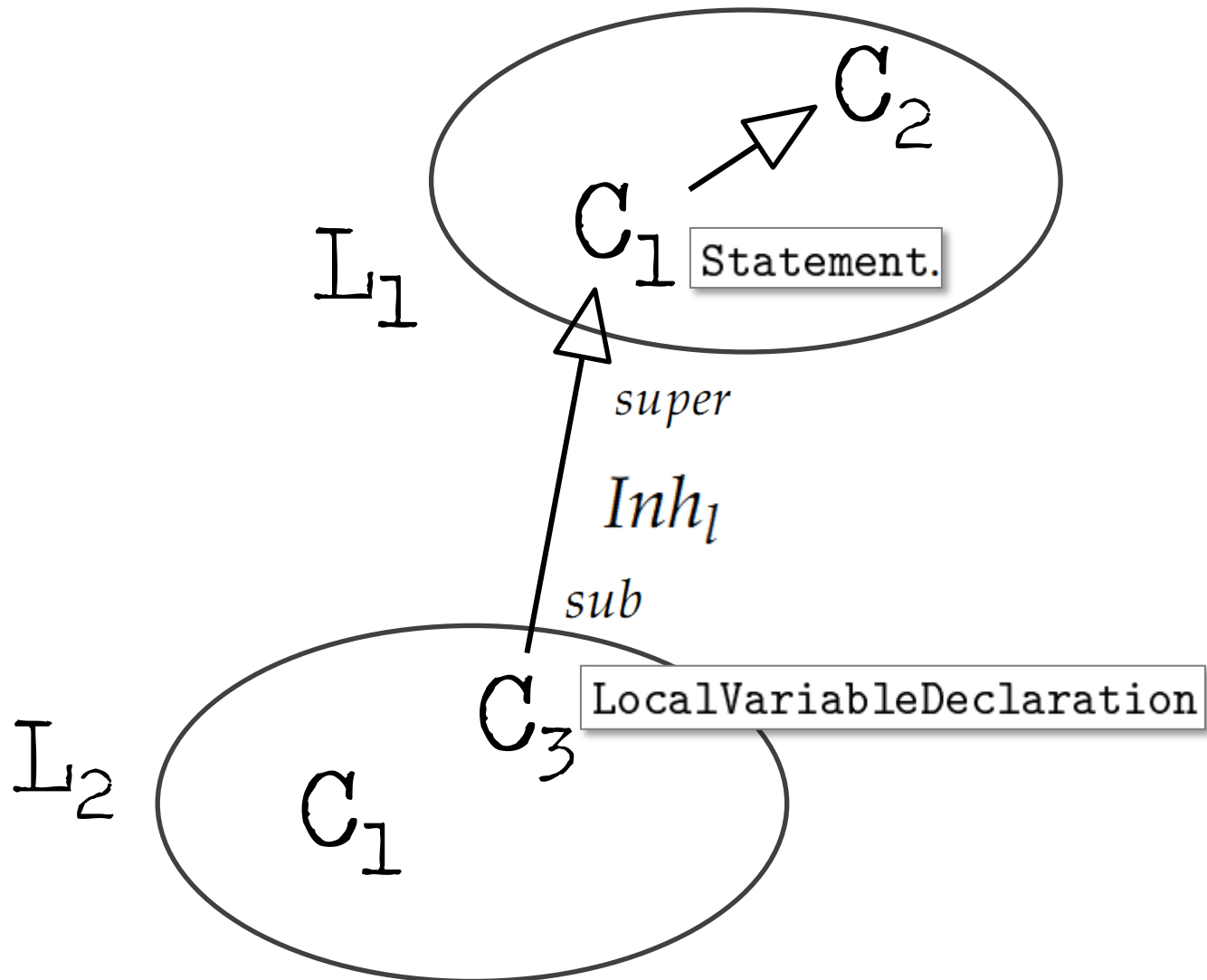
Languages are
sets of concepts



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

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

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

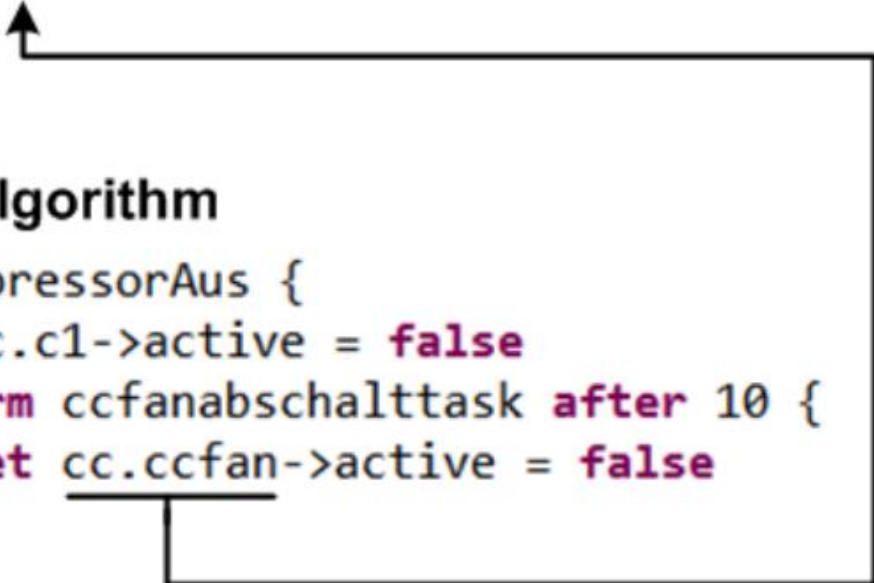
Independence

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

Example

Extended 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 {
```

```
    initism(c1);
```

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

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

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

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

```
}
```

Heterogeneous

C

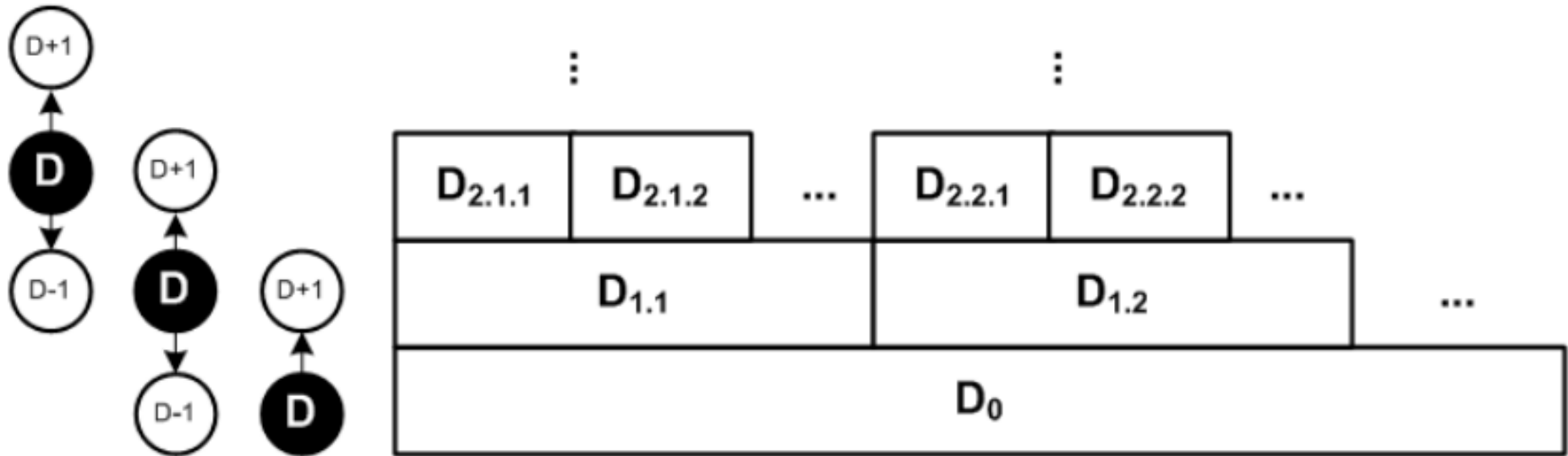
Statemachines

Testing

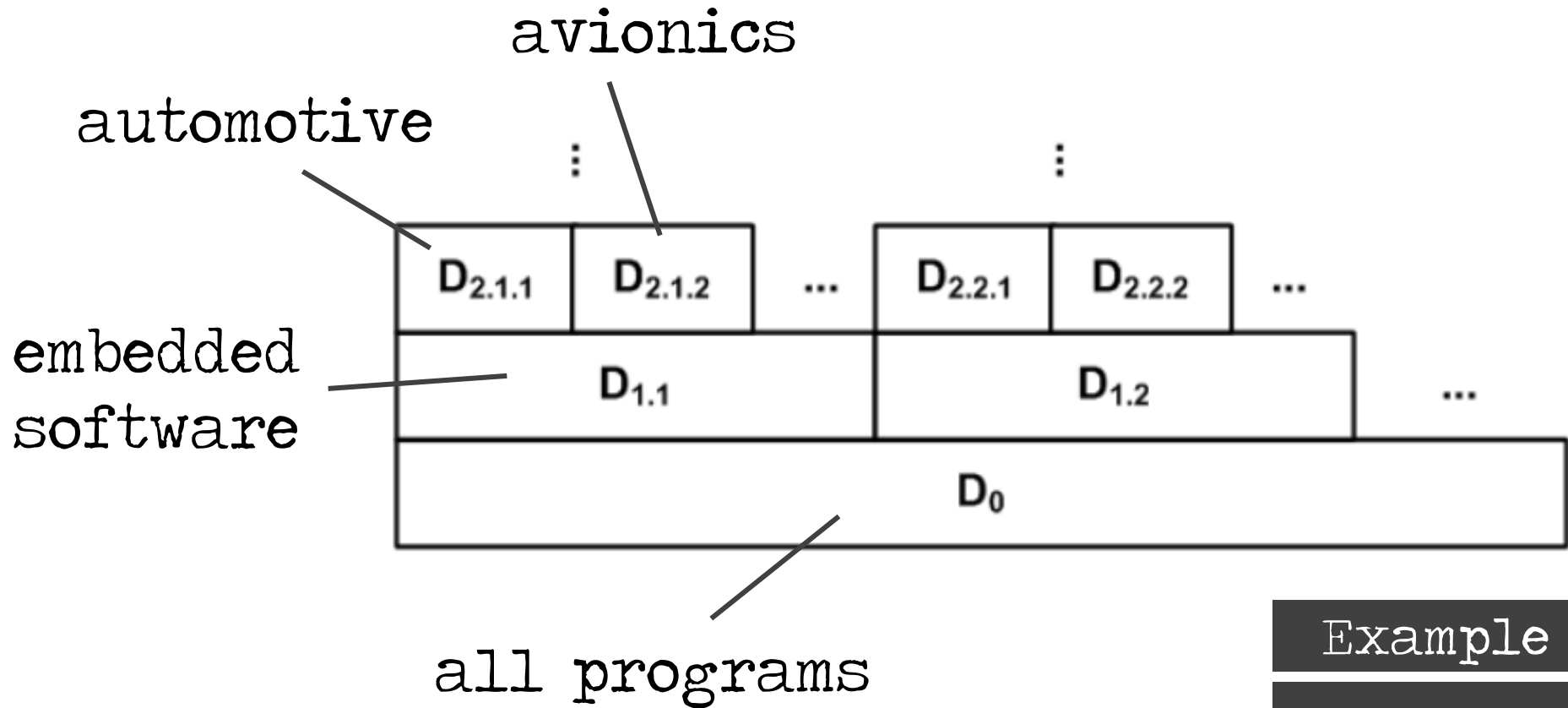
Example

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

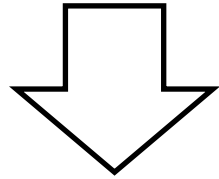
Shorter Programs

More
Accessible
Semantics

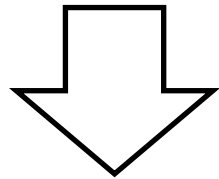
For a limited
Domain!

Domain Knowledge
encapsulated in
language

Smaller Domain



More Specialized
Language



Shorter Programs

The
do-what-I-want
language

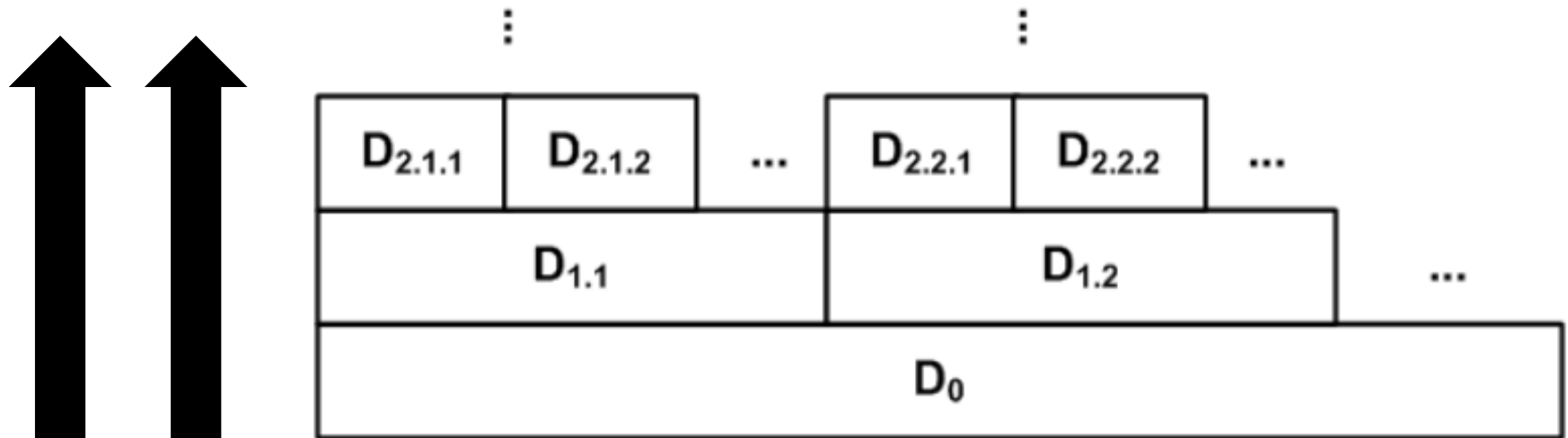
Ψ

Single Program
vs. Class/Domain

No Variability!

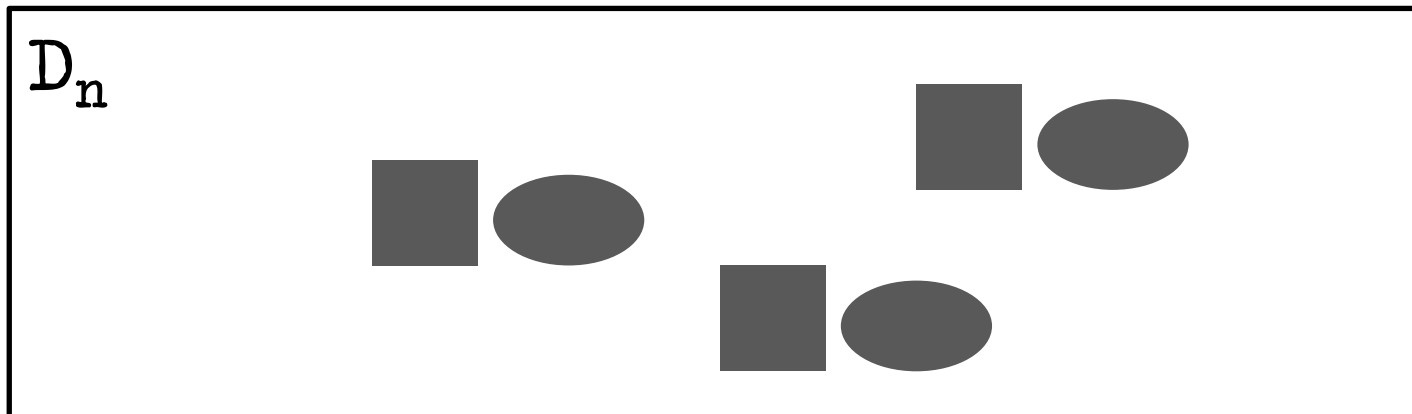
Ψ

Domain Hierarchy

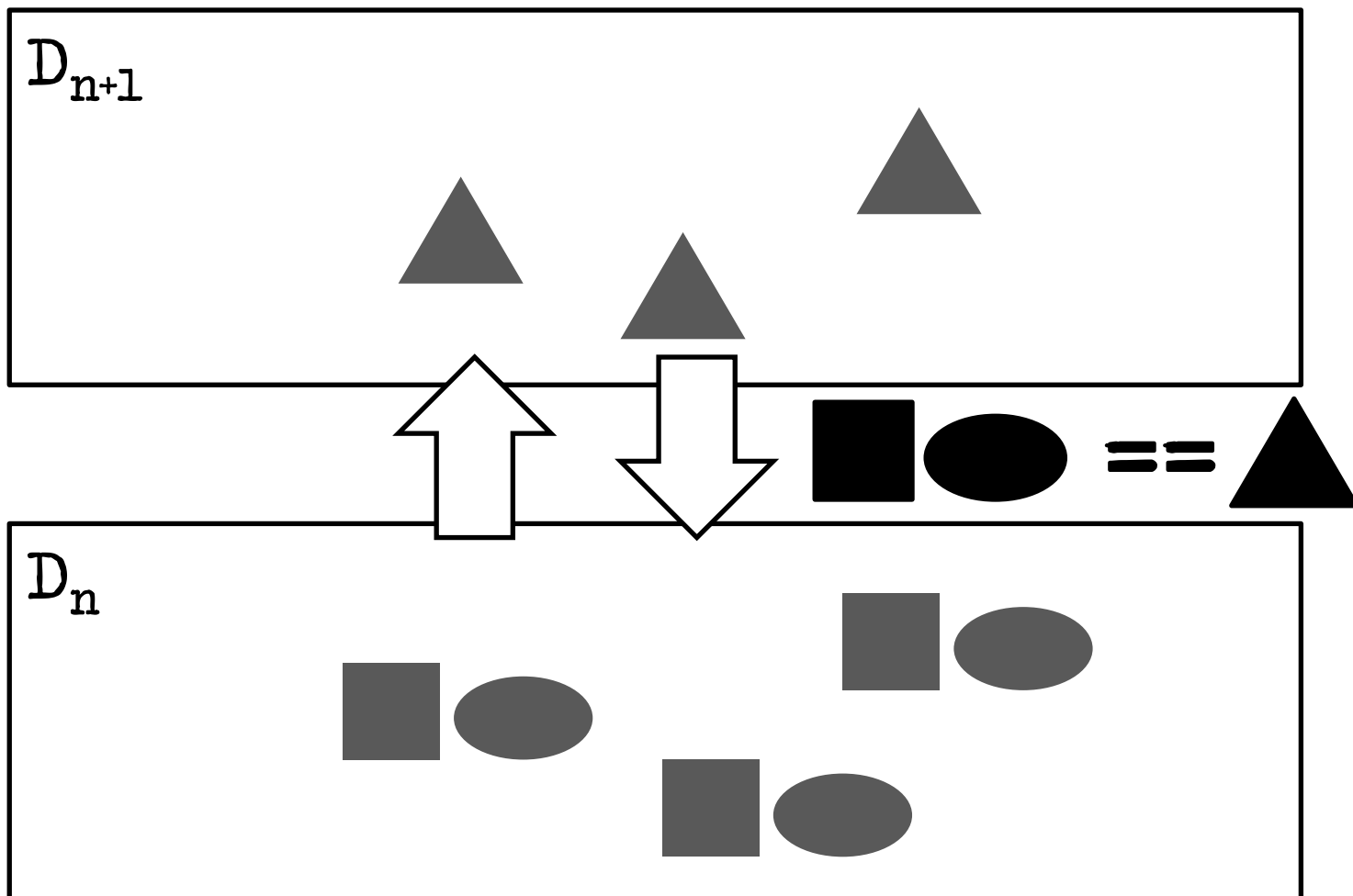


more specialized domains
more specialized languages

Reification

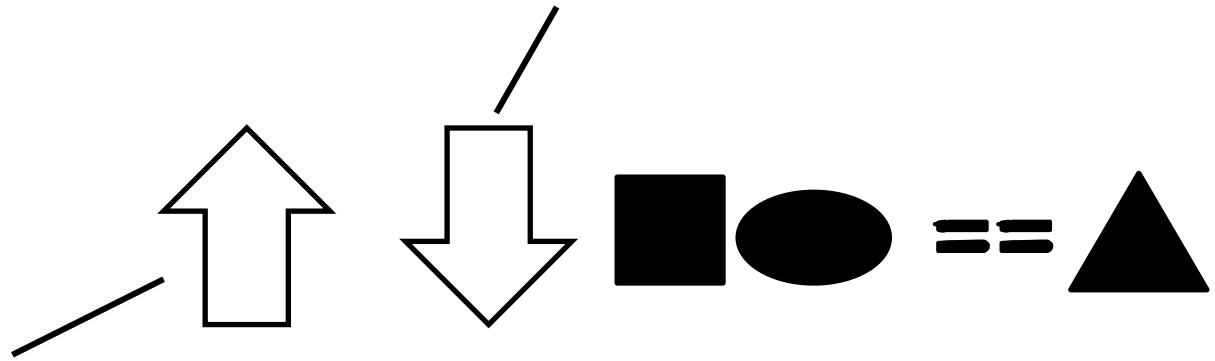


Reification



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] );  
}
```



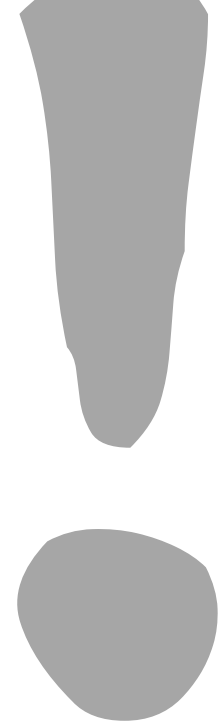
Linguistic Abstraction

```
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

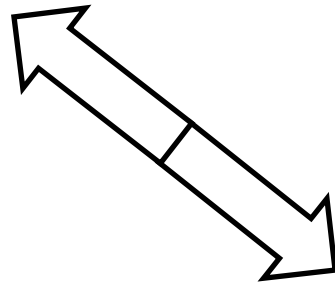
A good DSL does **not** require the use of patterns and idioms to express **semantically**

interesting concepts in D.

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

Declarative!

Linguistic
Abstraction



In-Language
Abstraction

Libraries

Classes

Frameworks

Linguistic Abstraction

Analyzable

Better IDE Support

In-Language Abstraction

User-Definable

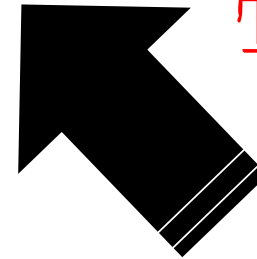
Simpler Language

Linguistic Abstraction

Analyzable

Better IDE Support

Special
Treatment!



In-Language Abstraction

User-Definable

Simpler Language

Linguistic
Abstraction

Std Lib

In-Language
Abstraction

Std Lib

```
lib stdlib {  
  
  command compartment::coolOn  
  command compartment::coolOff  
  property compartment::totalRuntime: int readonly  
  property compartment::needsCooling: bool readonly  
  property compartment::couldUseCooling: bool readonly  
  property compartment::targetTemp: int readonly  
  property compartment::currentTemp: double readonly  
  property compartment::isCooling: bool readonly  
  
}
```

Example

Refrige
rators

Language
Evolution
Support

Precision
vs.
Algorithmics

Coverage

expressivity

coverage

semantics

separation of

concerns

completeness

paradigms

modularity

concrete

syntax

process

NOTICE



THIS PART OF THE
SLIDE DECK
HAS BEEN SKIPPED.

PLEASE REFER TO THE
DSL ENGINEERING BOOK.

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

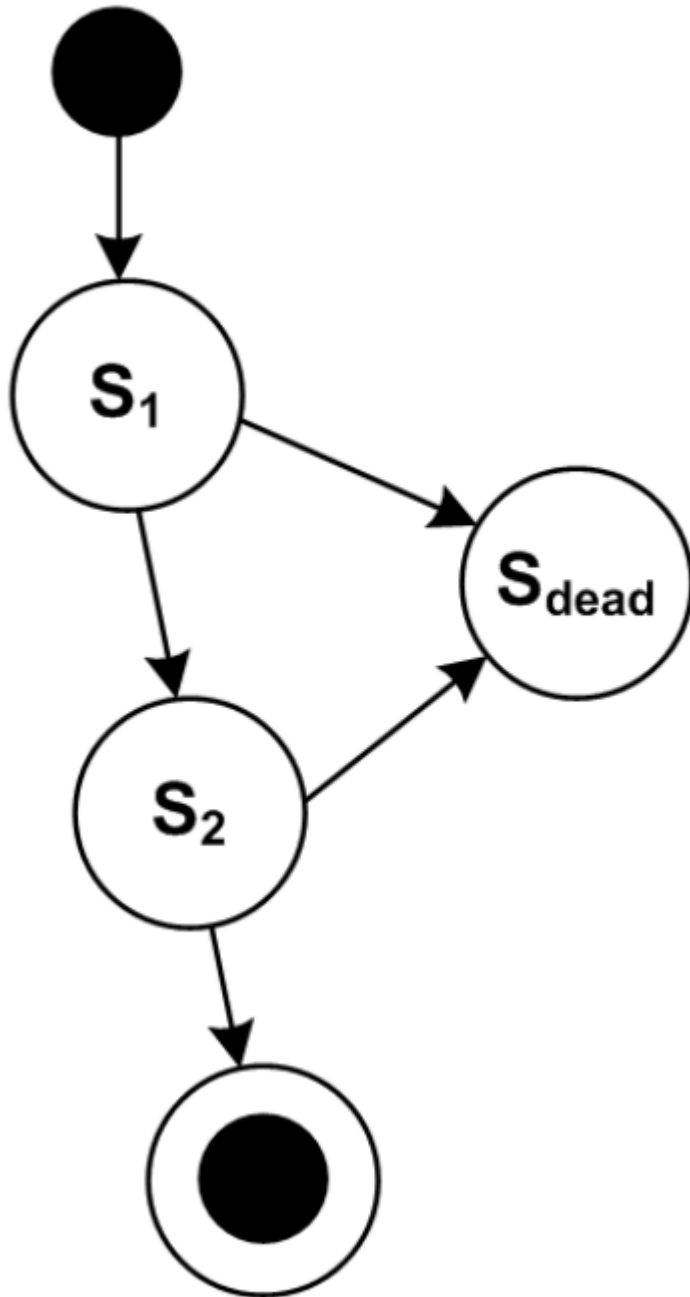
Type Systems

Unique State Names

Unreachable States

Dead End States

...



Example

Extended C

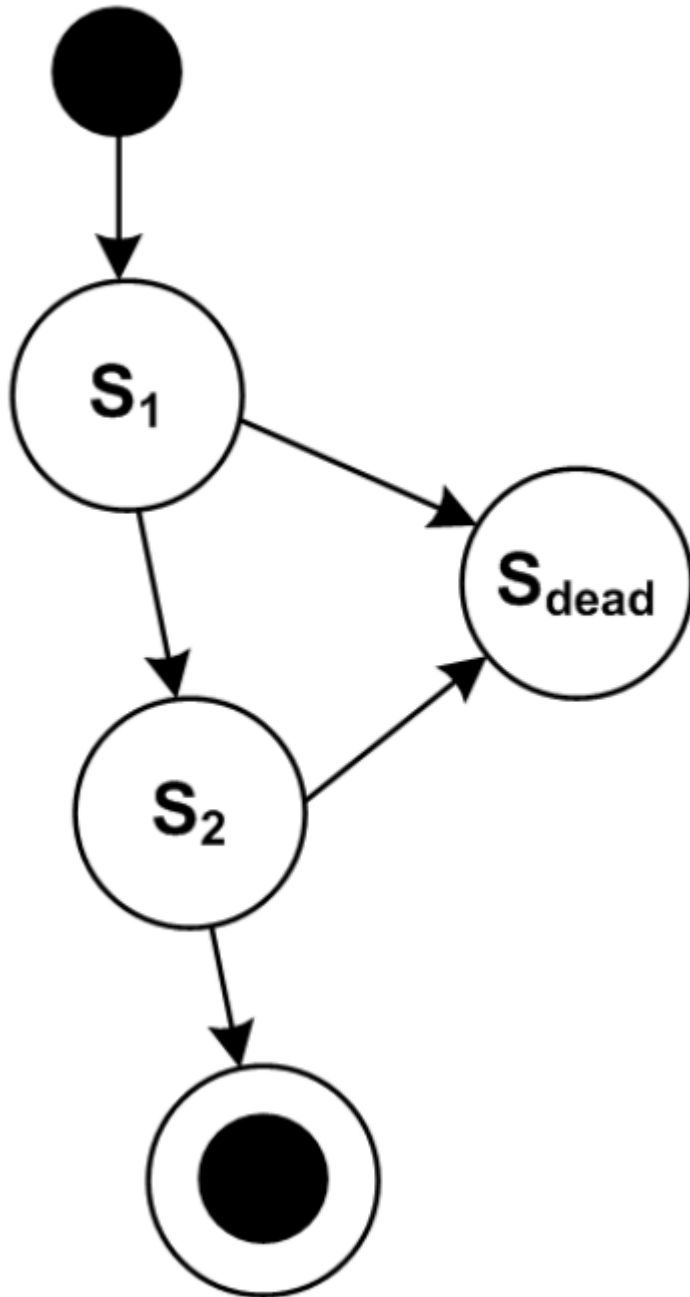
Unique State Names

Unreachable States

Dead End States

...

Easier to do on a
declarative Level!



Example

Extended C

Unique State Names

Unreachable States

Dead End States

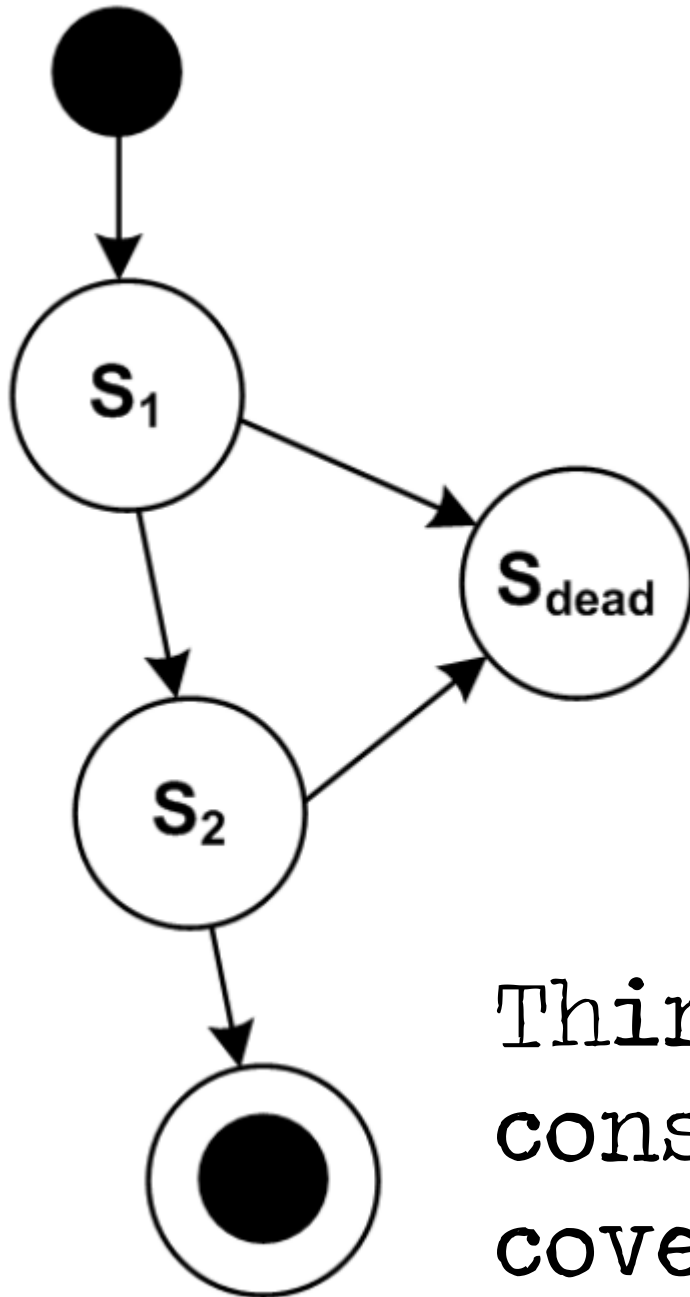
...

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

What does a type system do?

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

Assign fixed types



Derive Types

What does a type system do?

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

Assign fixed types

A diagram with several blue arrows pointing from the code above to the text 'Assign fixed types'. The arrows originate from the 'int' type, the '2' constant, the '*' operator, the 'someFunction' identifier, and the 'sqrt(2)' expression.

Derive Types

A diagram with a single orange arrow pointing from the 'int' type in the code above to the text 'Derive Types'.

Calculate Common Types

A diagram with a single green arrow pointing from the '*' operator in the code above to the text 'Calculate Common Types'.

What does a type system do?

```
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

```
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

```
macro kompressorAus {  
  set cc.c1->active = "aString"  
  perform cc.fanausscha  
    set cc.ccfan->ac  
  }  
}
```

⊗ incompatible type BoolType|COMPARABLE and StringType (on a AssignmentStatement)

Example

Refrige
rators

What does it
all mean?

Execution Semantics

Def: Semantics

... via mapping to lower level

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

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

OB: Observable Behaviour (Test Cases)

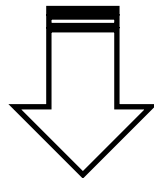
Def: Semantics

... via mapping to lower level

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

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

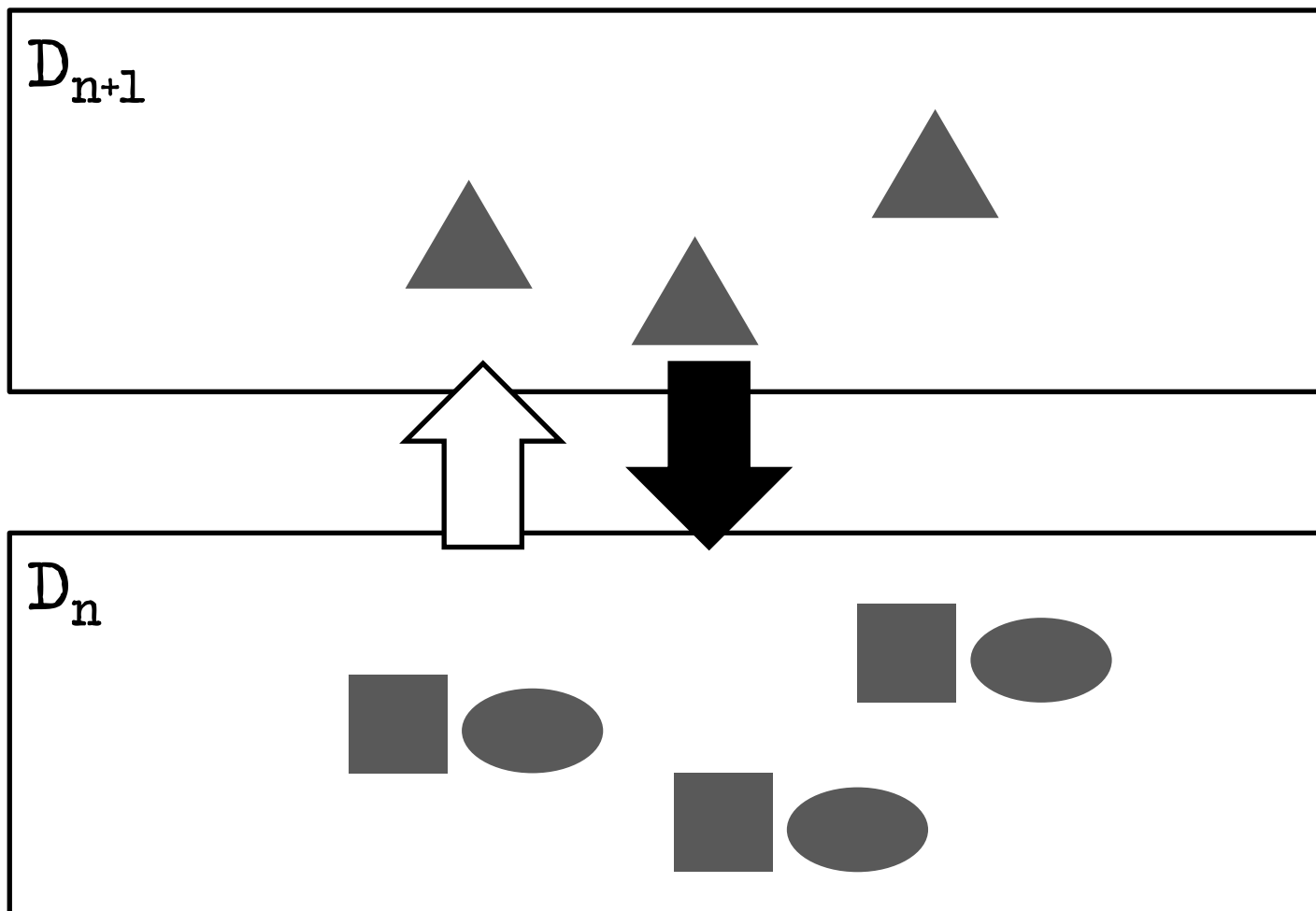
L_D



Transformation
Interpretation

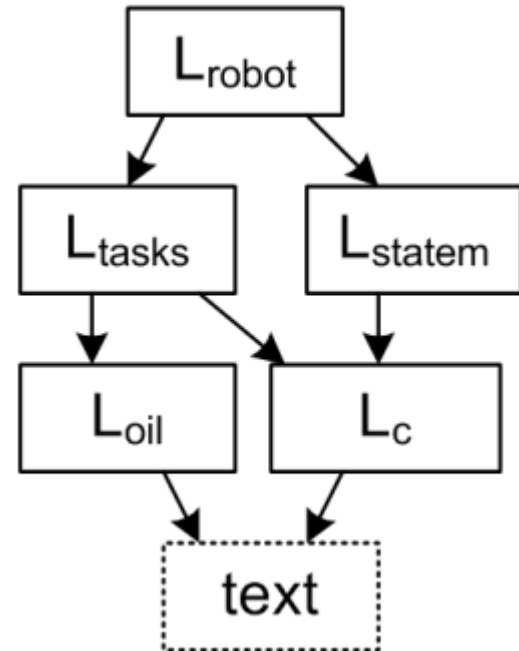
L_{D-1}

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  
  
}
```

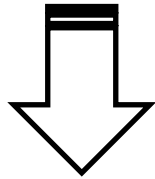


Example

Extended C

Transformation

L_D



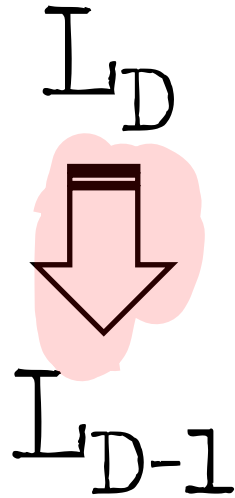
Transformation

L_{D-1}

Known Semantics!

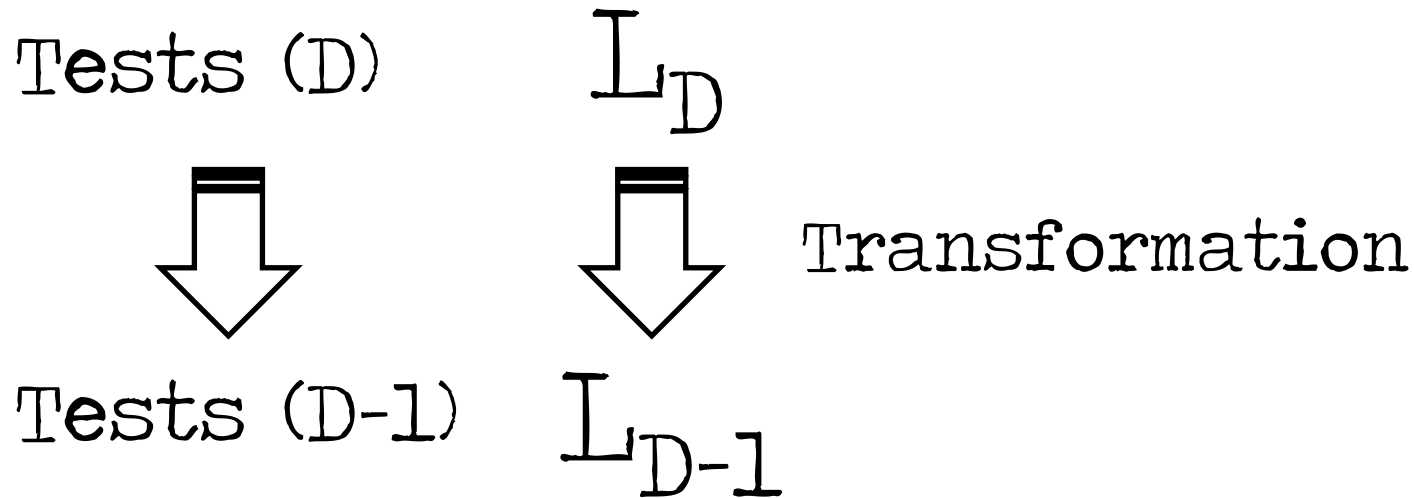
Transformation

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

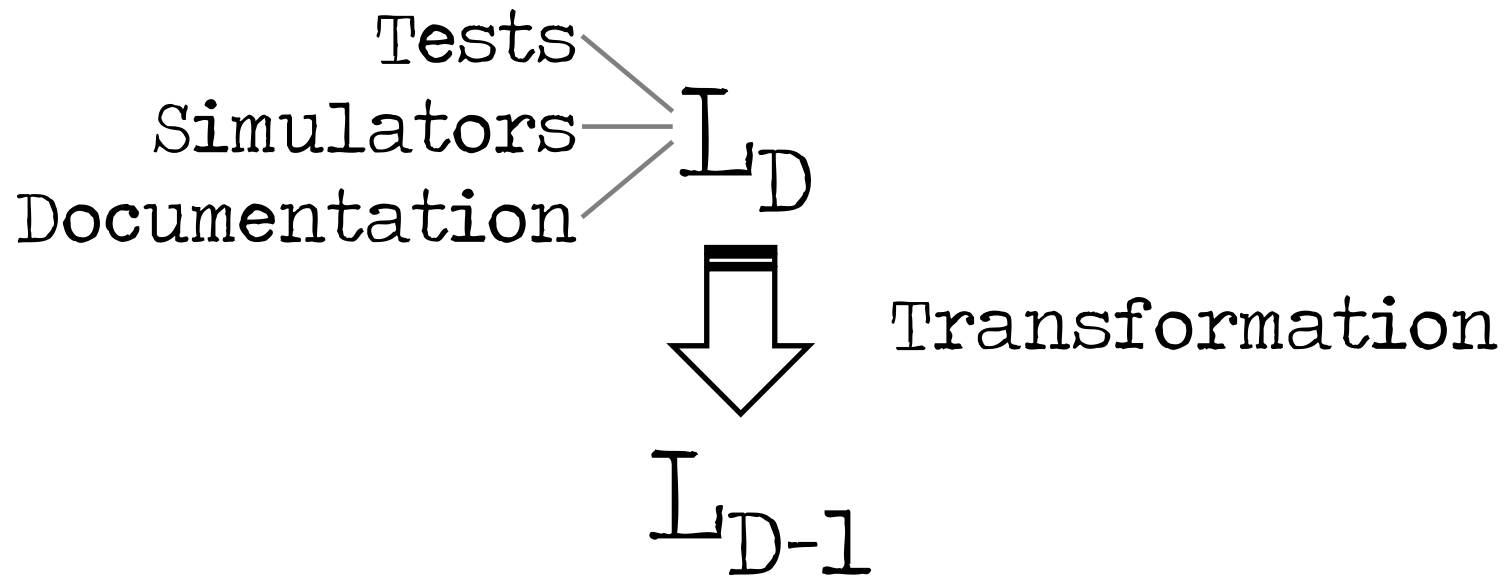
Transformation

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

Transformation



Simulation View

Status **Control**

Current Test: KIRAbtauen Autorun 10 Enable Breakpoints

Current State: - Single Step

Current Step: -

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

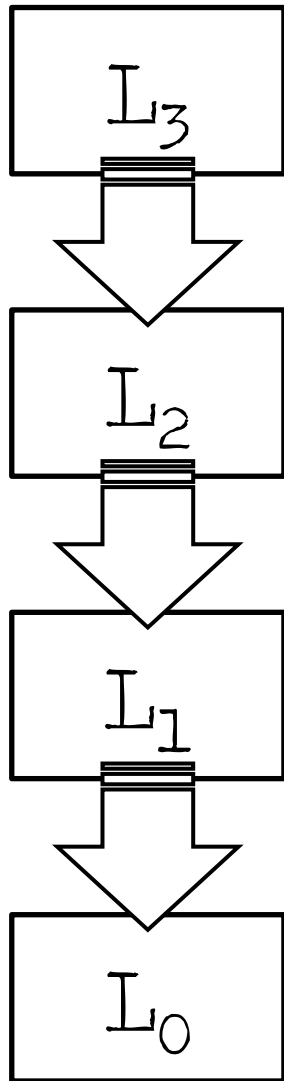
Running Tasks

Task	Sinc...

Example

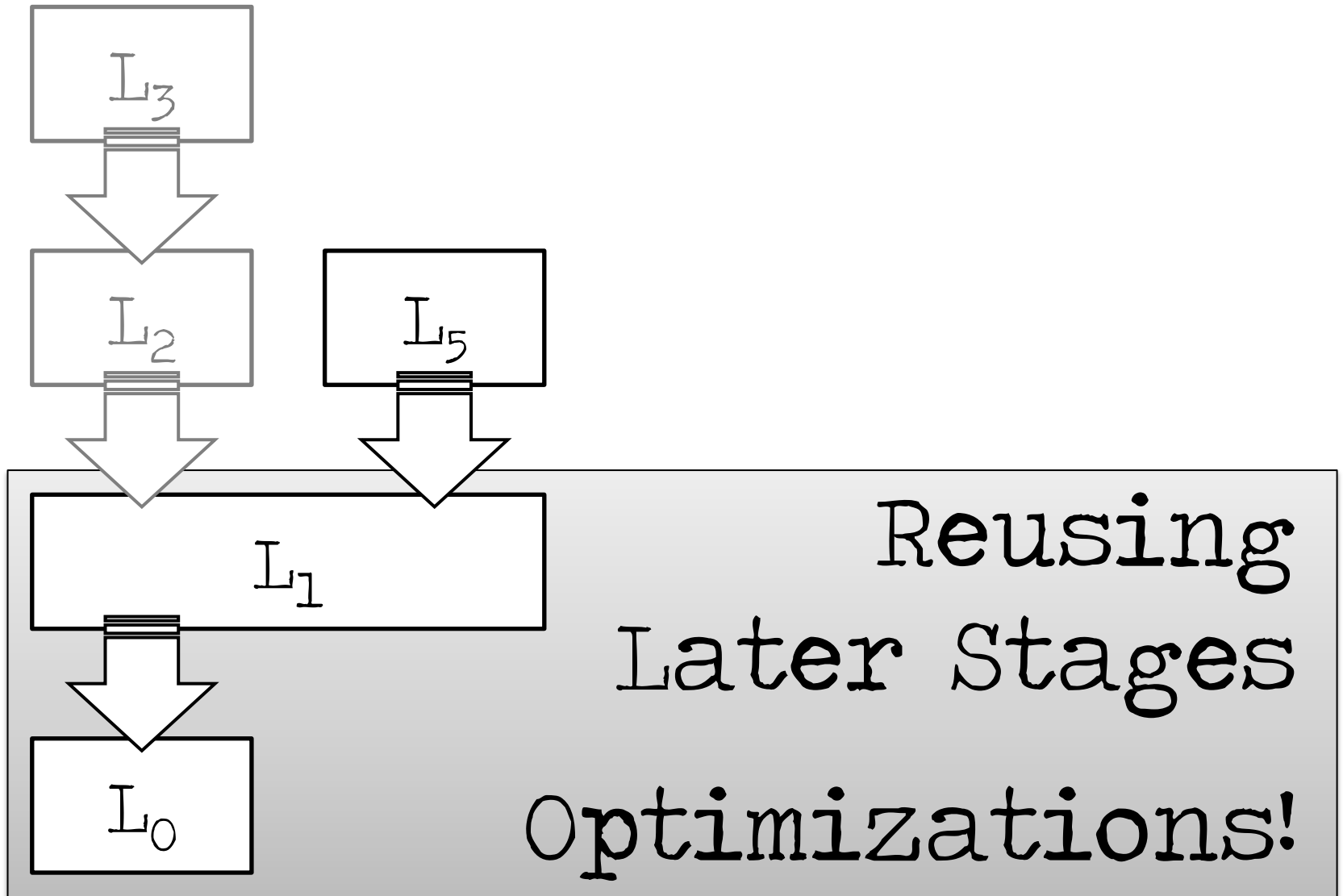
Refrige
rators

Multi-Stage

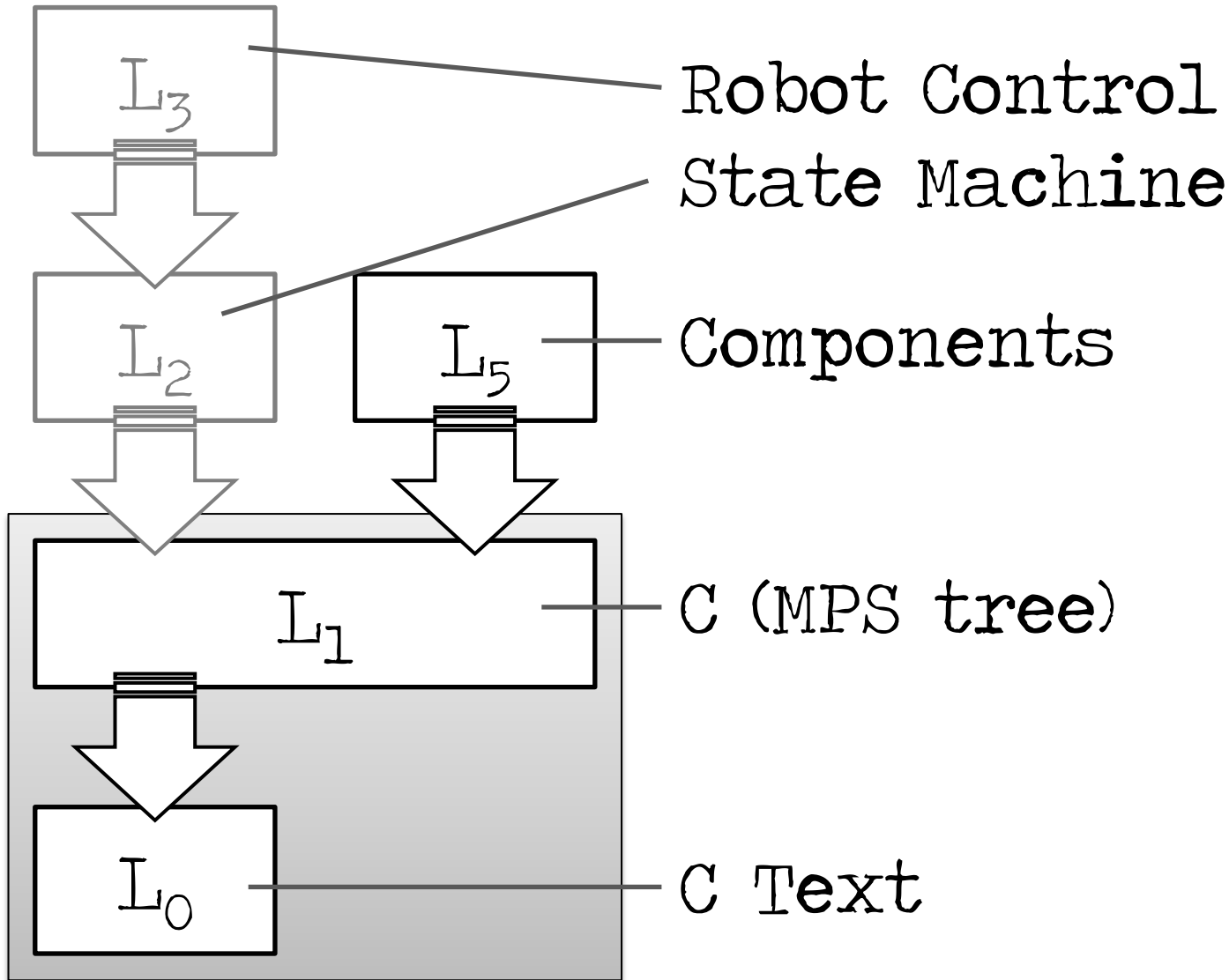


Modularization

Multi-Stage: Reuse



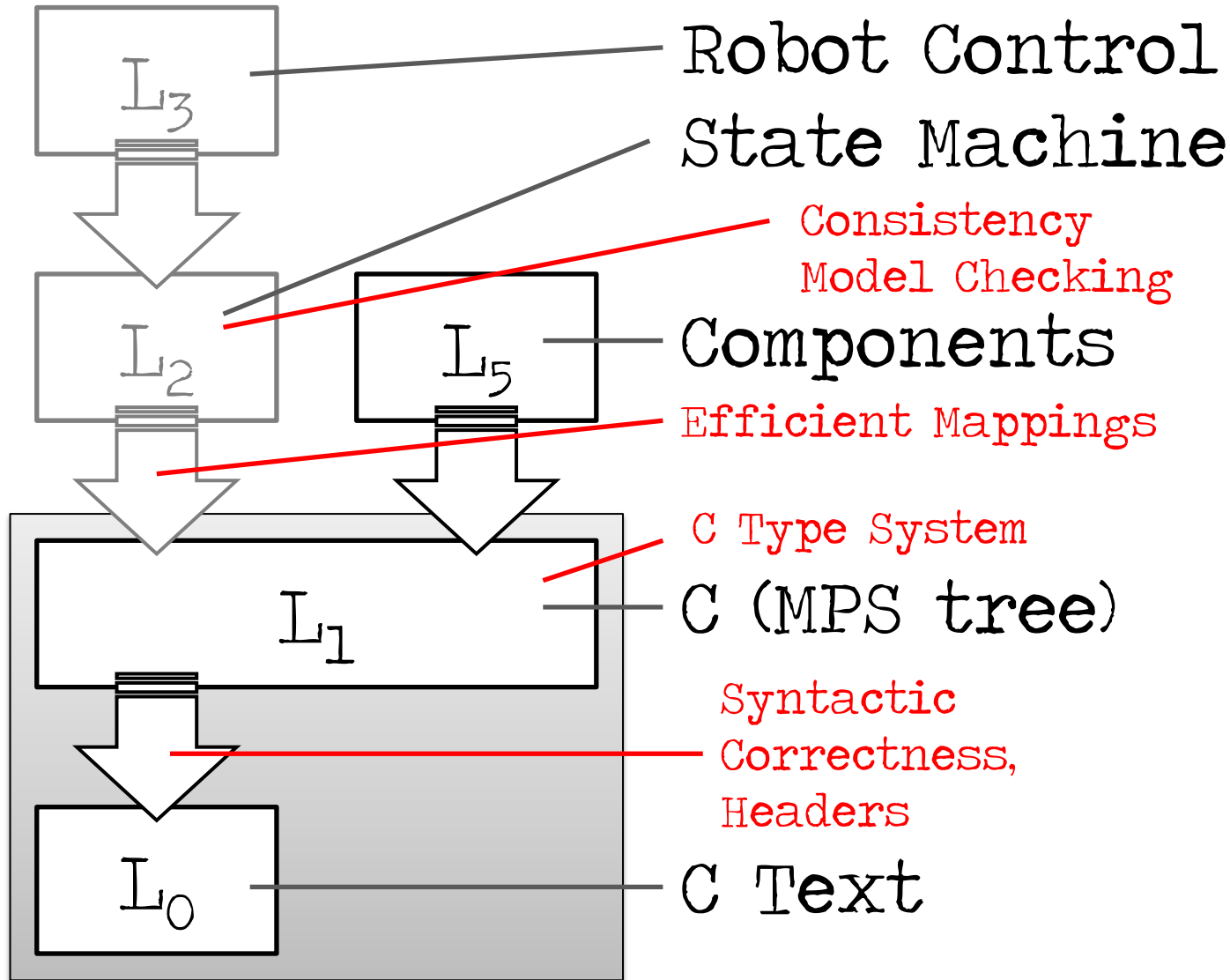
Multi-Stage: Reuse



Example

Extended C

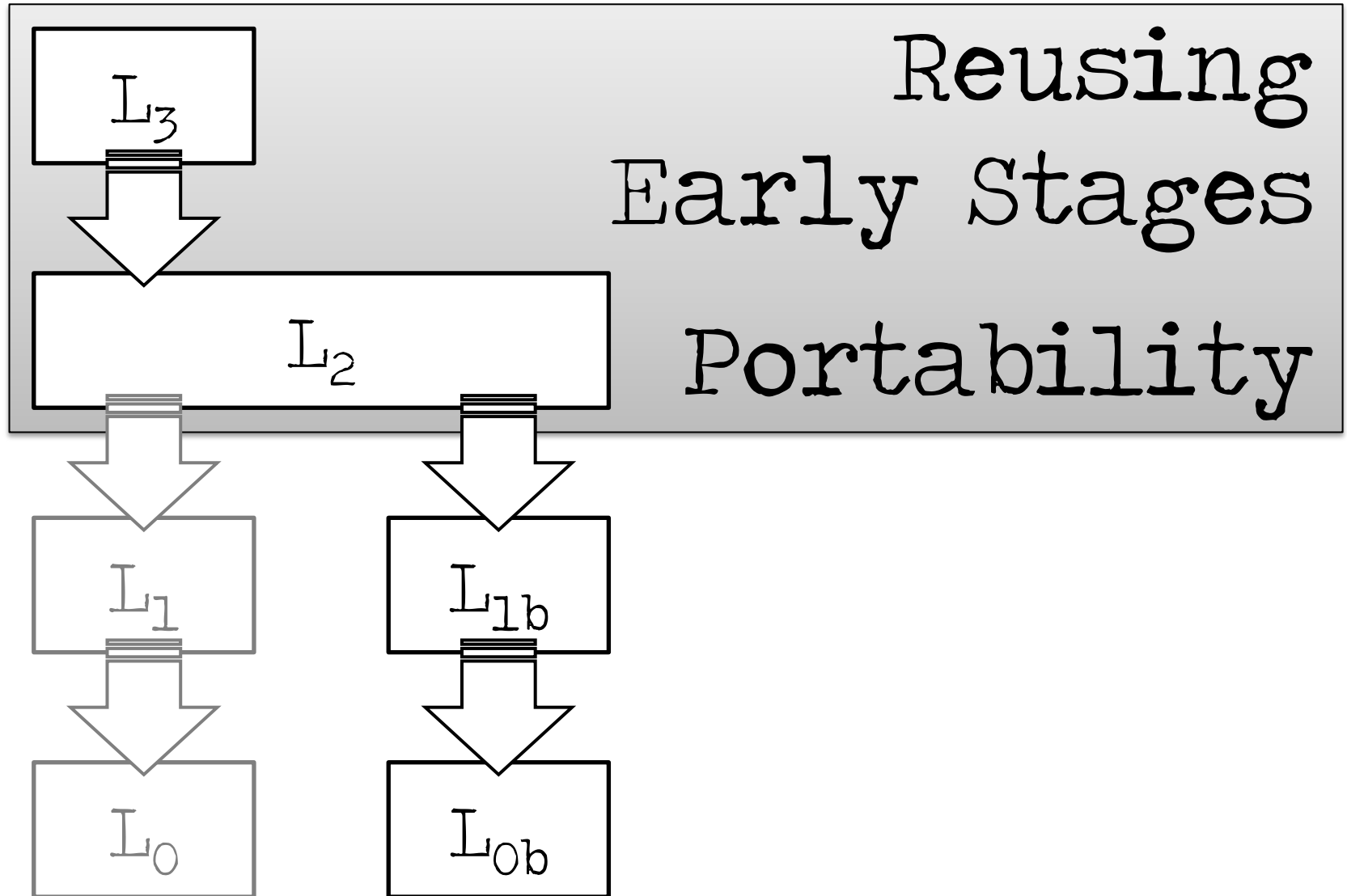
Multi-Stage: Reuse



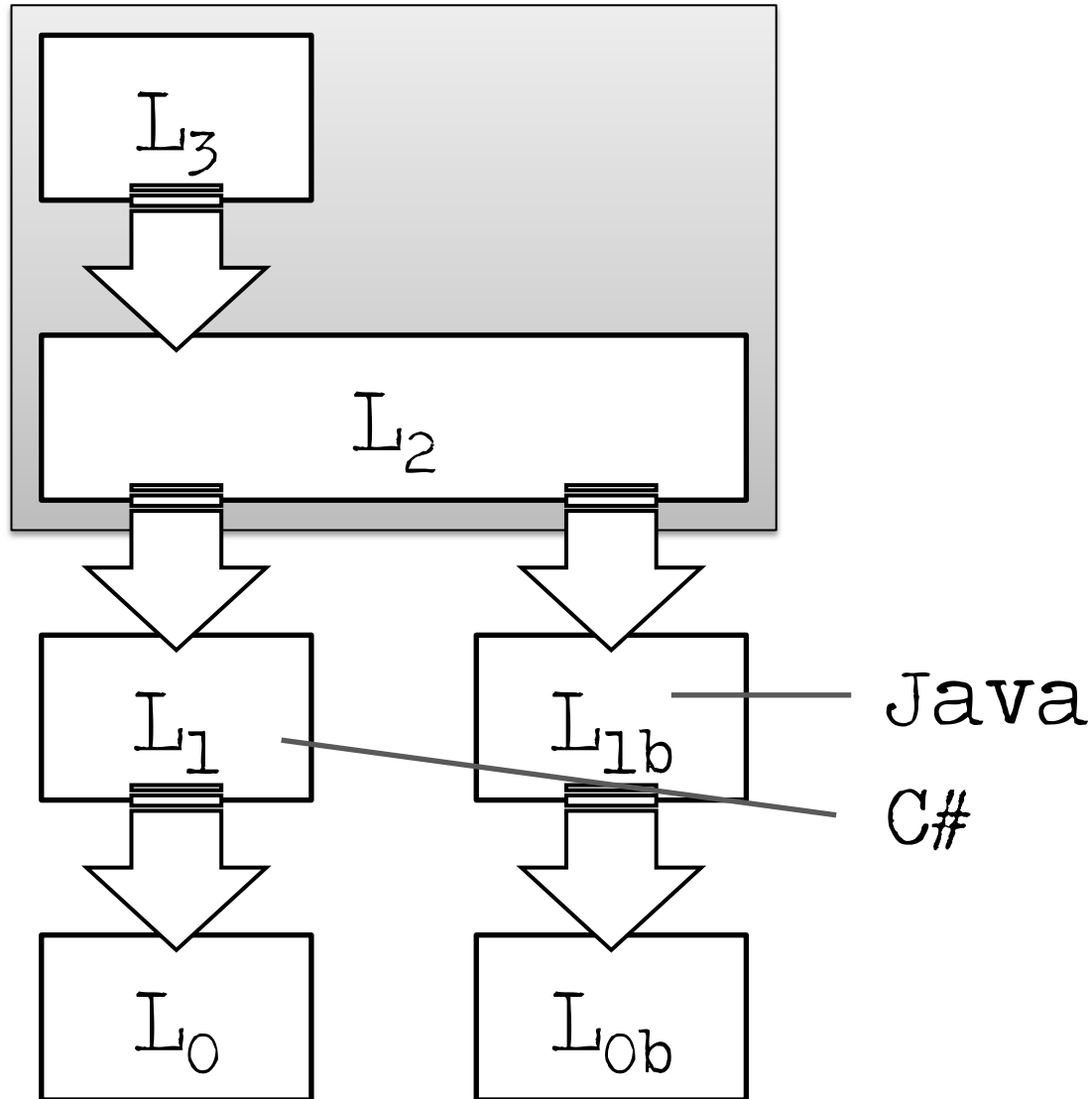
Example

Extended C

Multi-Stage: Reuse



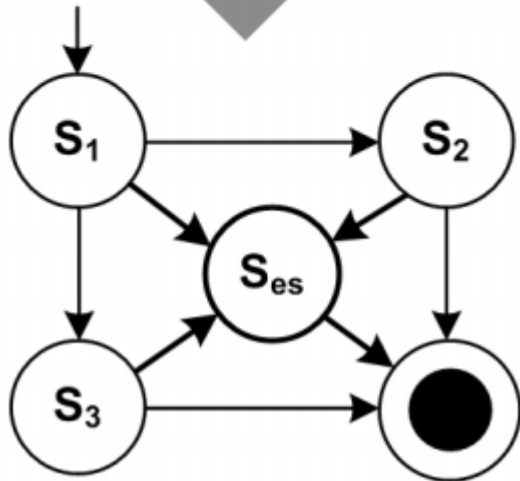
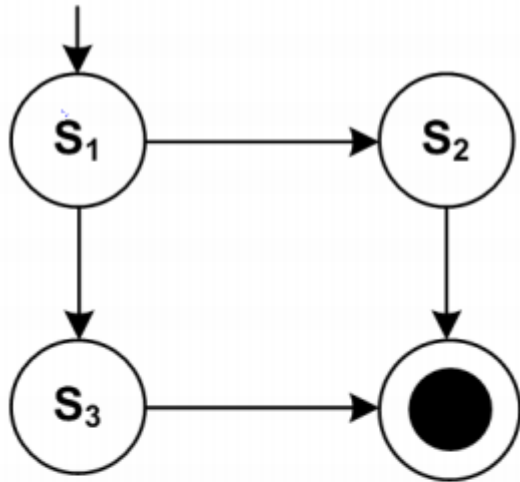
Multi-Stage: Reuse



Example

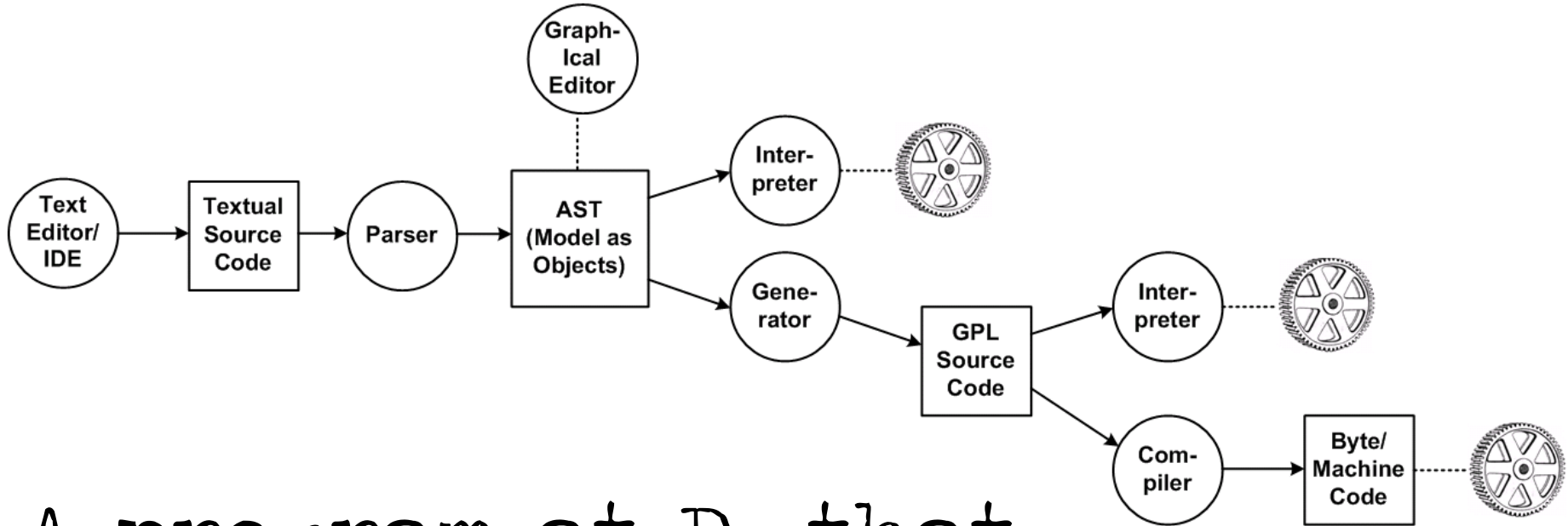
Extended C

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	
			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


```

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

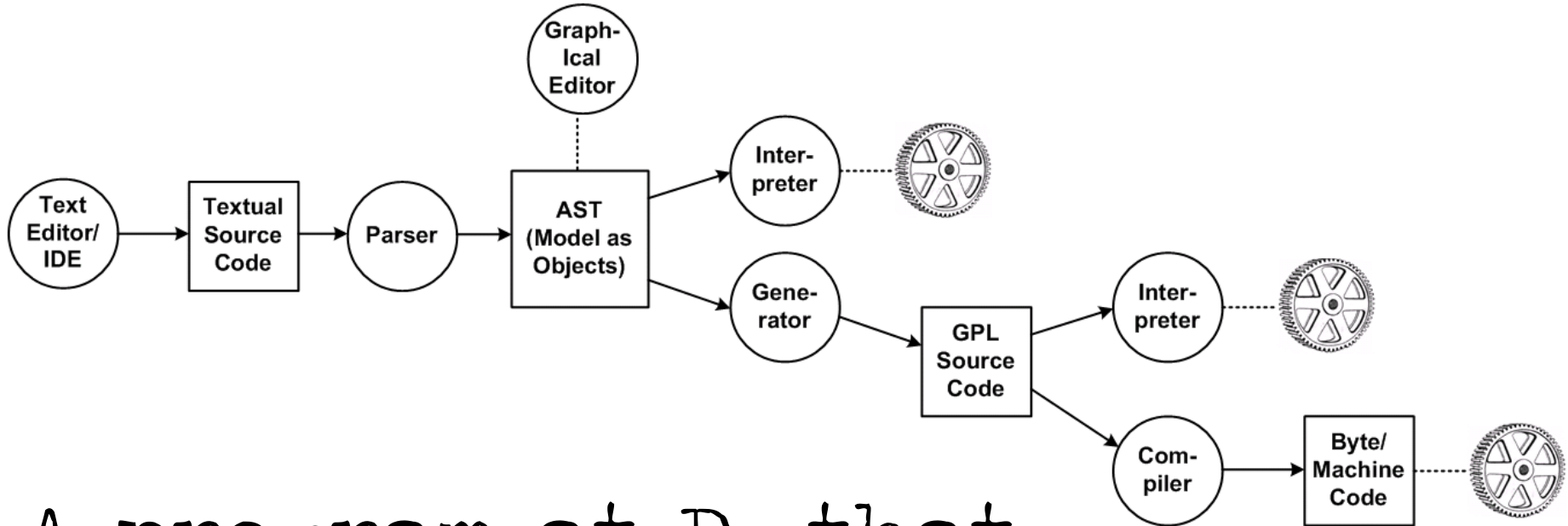
Running Tasks

Task	Sinc...

Example

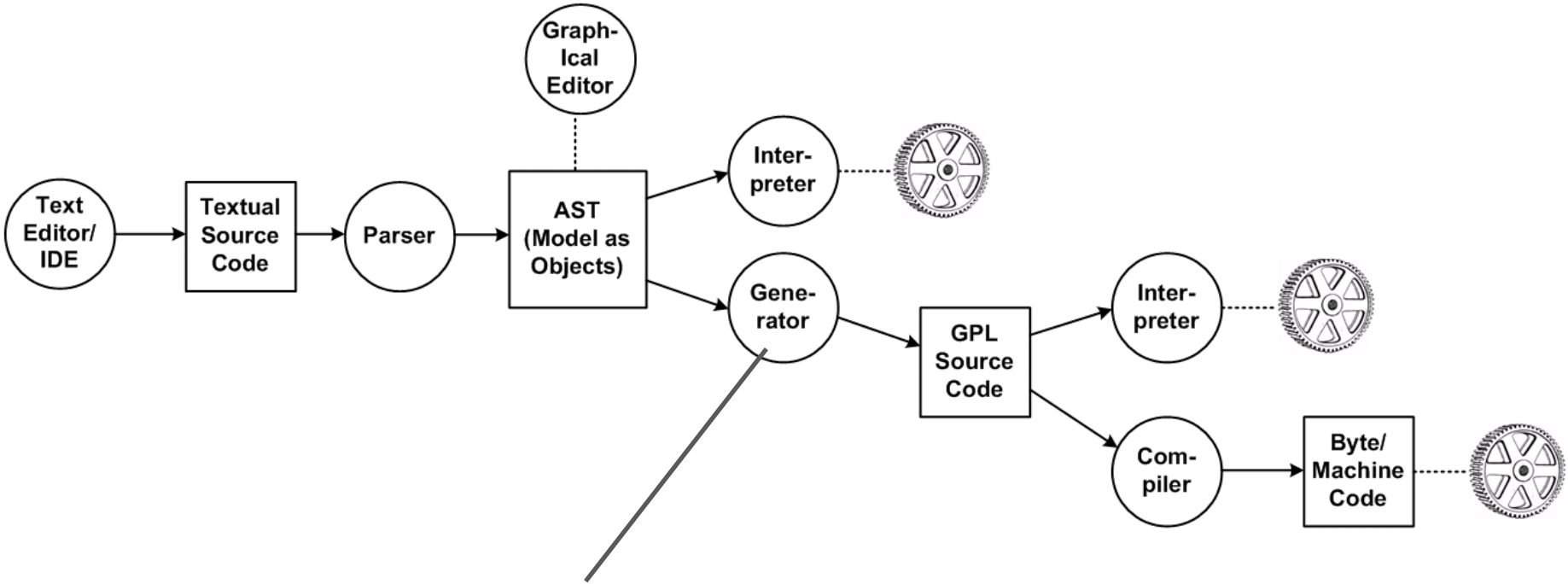
Refrige
rators

Interpretation



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

Interpretation



An interpreter :-)

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

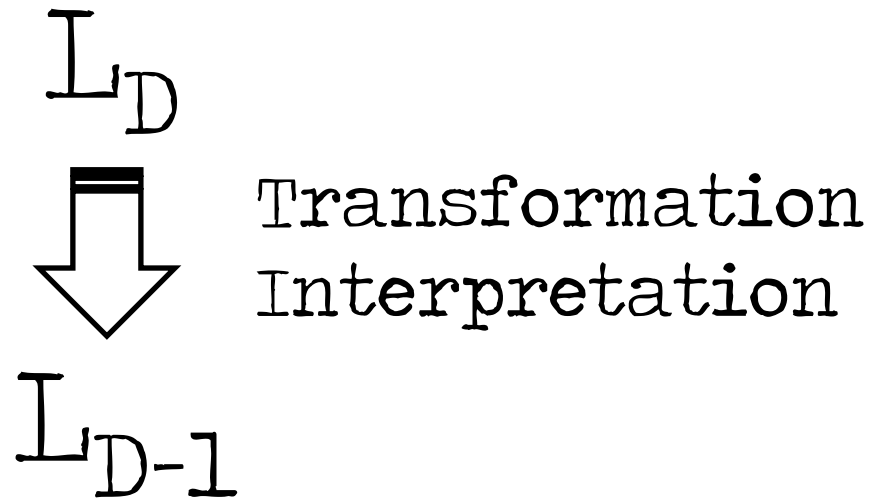
- + Turnaround Time
- + Runtime Change
Change Business
Rules without
Redeployment

Example

Pension
Plans

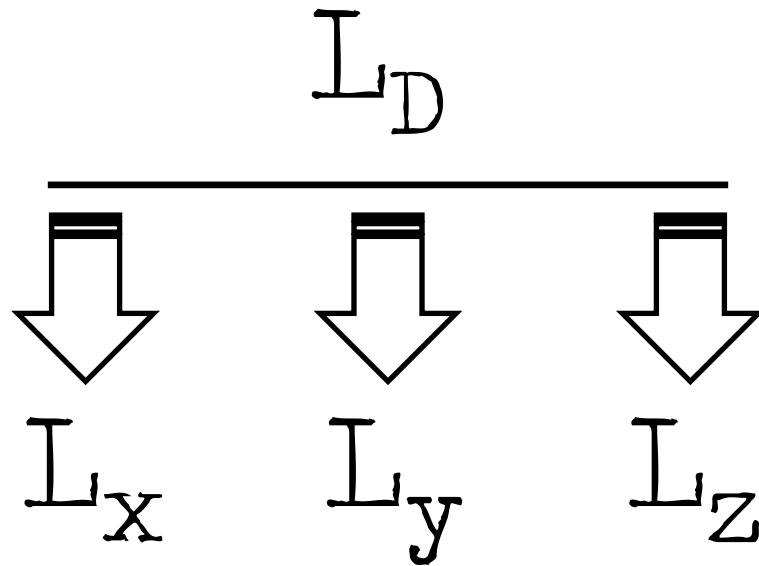
Def: Semantics

... via mapping to lower level



Multiple Mappings

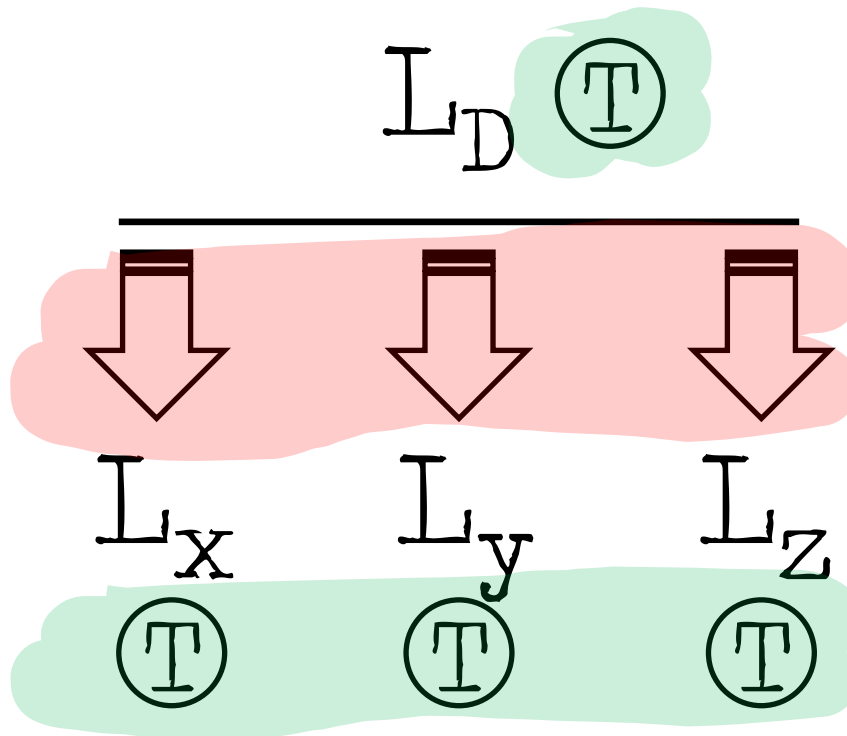
... at the same time



Similar
Semantics?

Multiple Mappings

... at the same time

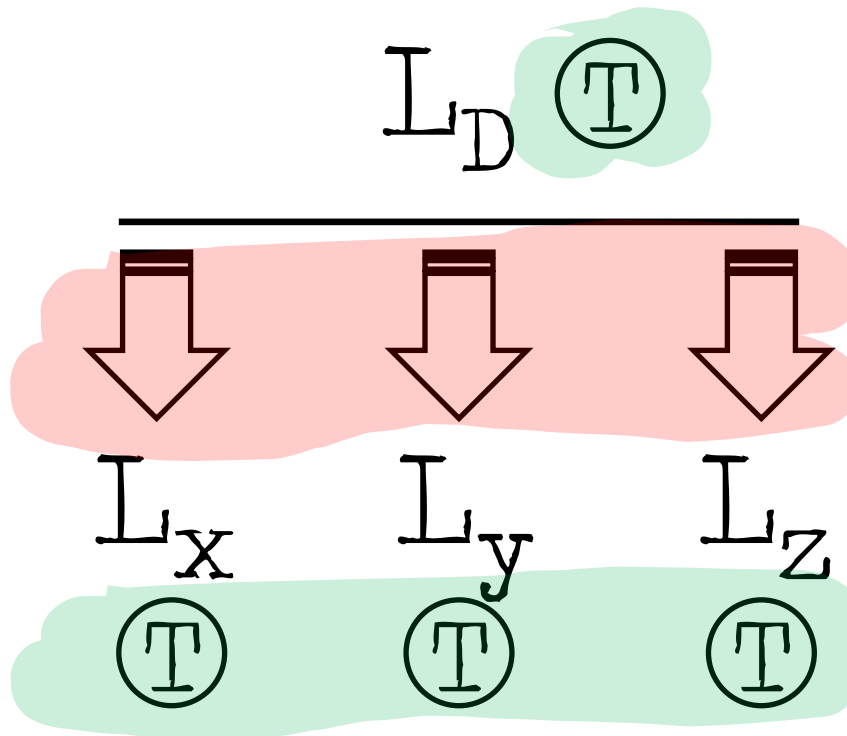


Similar
Semantics?

all green!











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

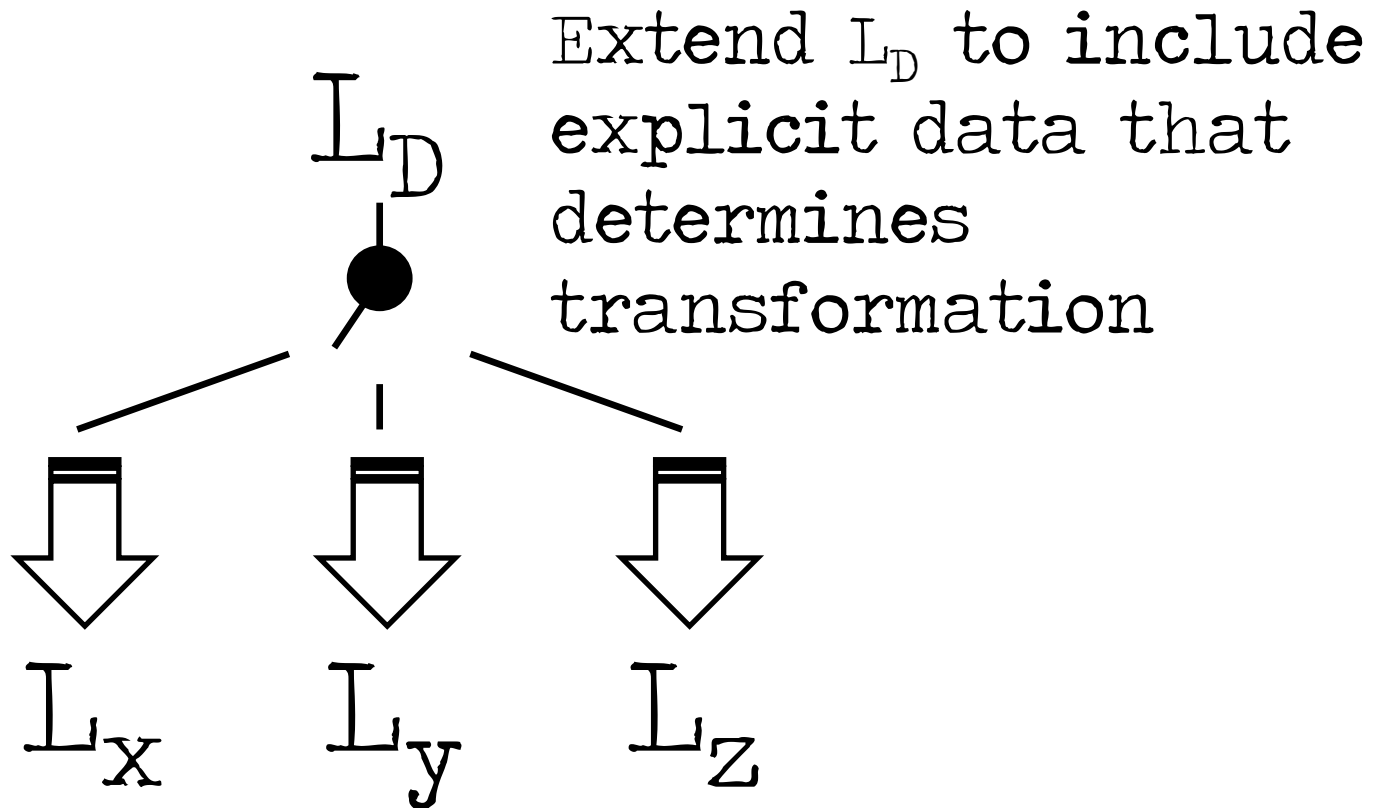
```

Example

Refrige
rators

Multiple Mappings

... alternatively, selectably



Multiple Mappings

... alternatively, selectably

Extend L_D to include explicit data that determines transformation

L_D

```
exported component AnotherDriver extends Driver {
  ports:
    provides IDiagnostic diag
    requires optional ILogger logger
    requires ILowLevel lowlevel restricted to LowLevelCode.ll
  contents:
    field int8_t count = 0
}
```

X y Z

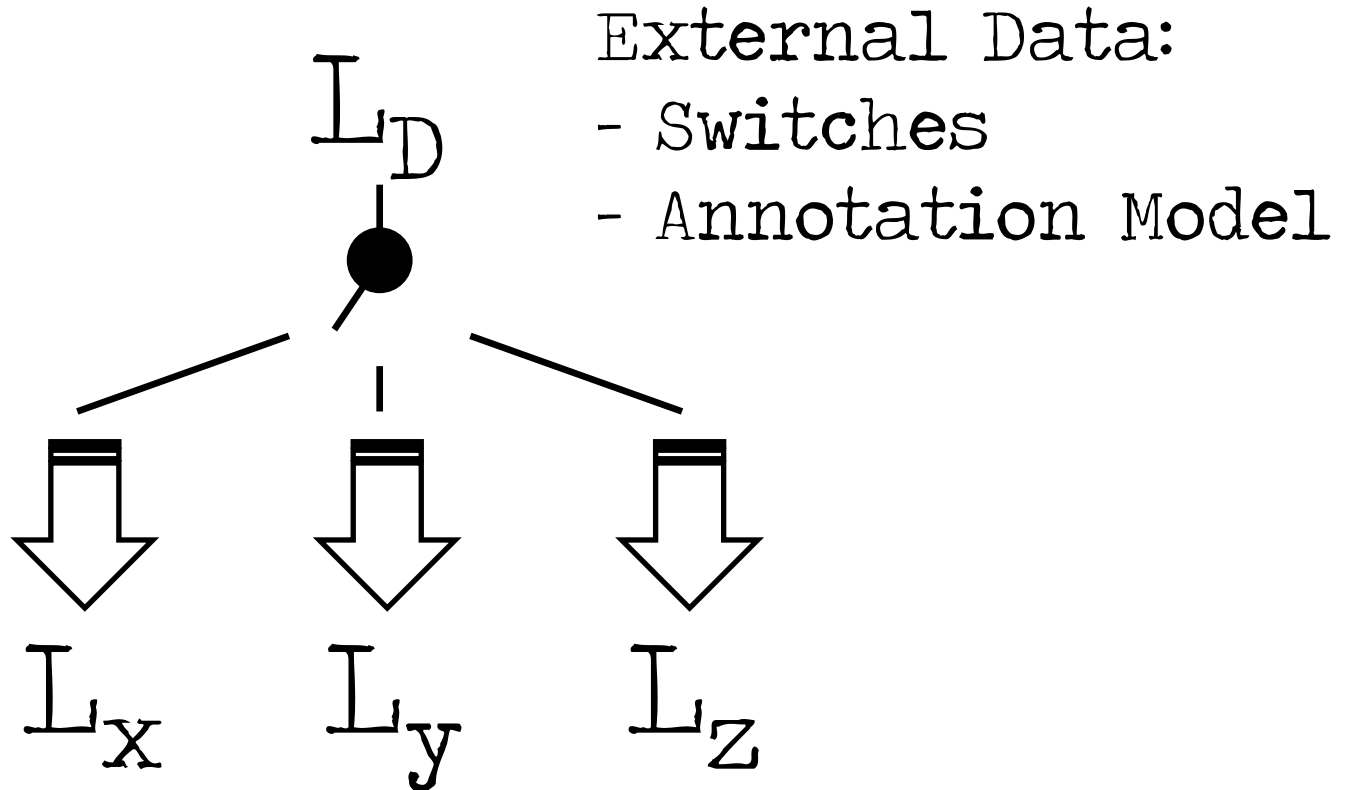
Restricted Port leads to reduced overhead C

Example

Extended C

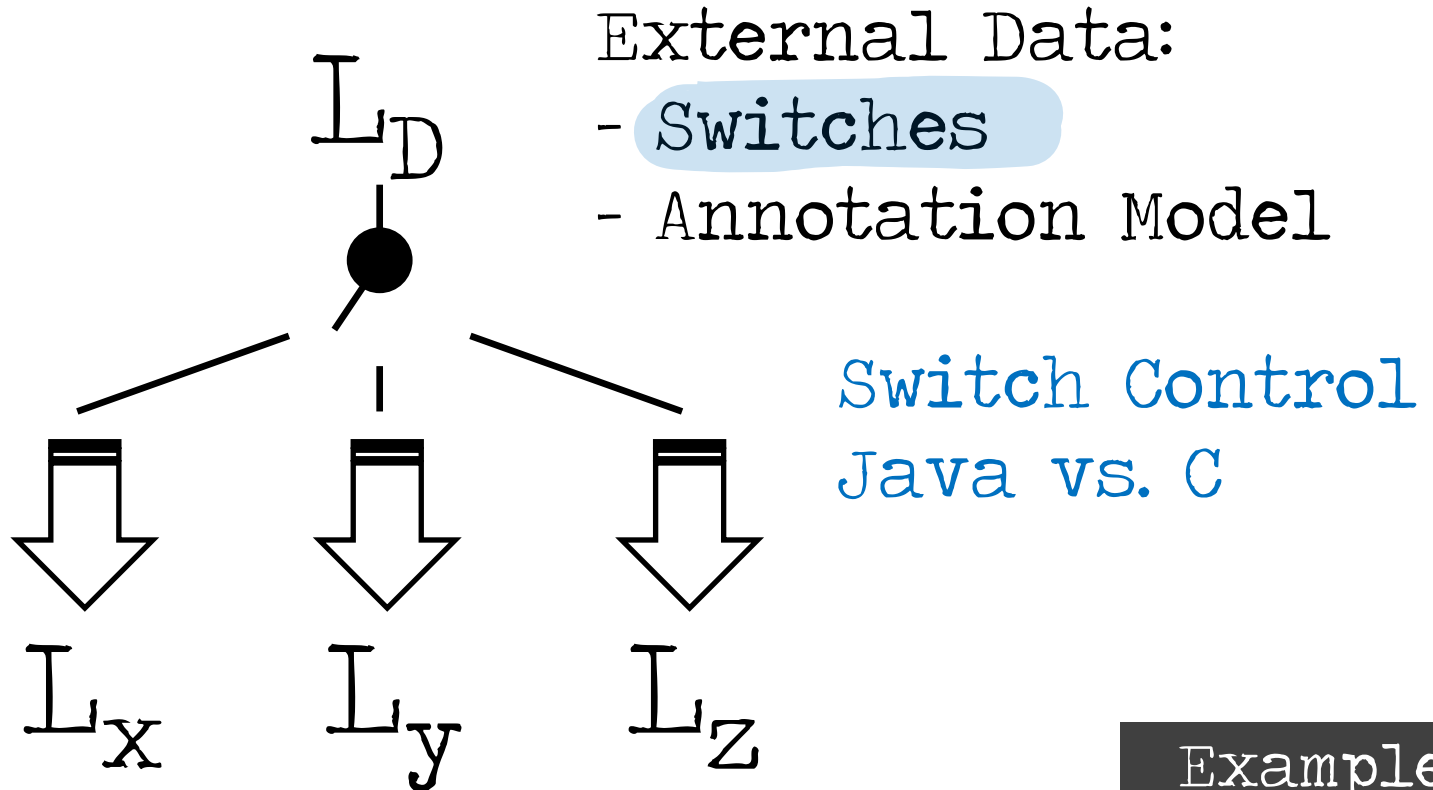
Multiple Mappings

... alternatively, selectably



Multiple Mappings

... alternatively, selectably

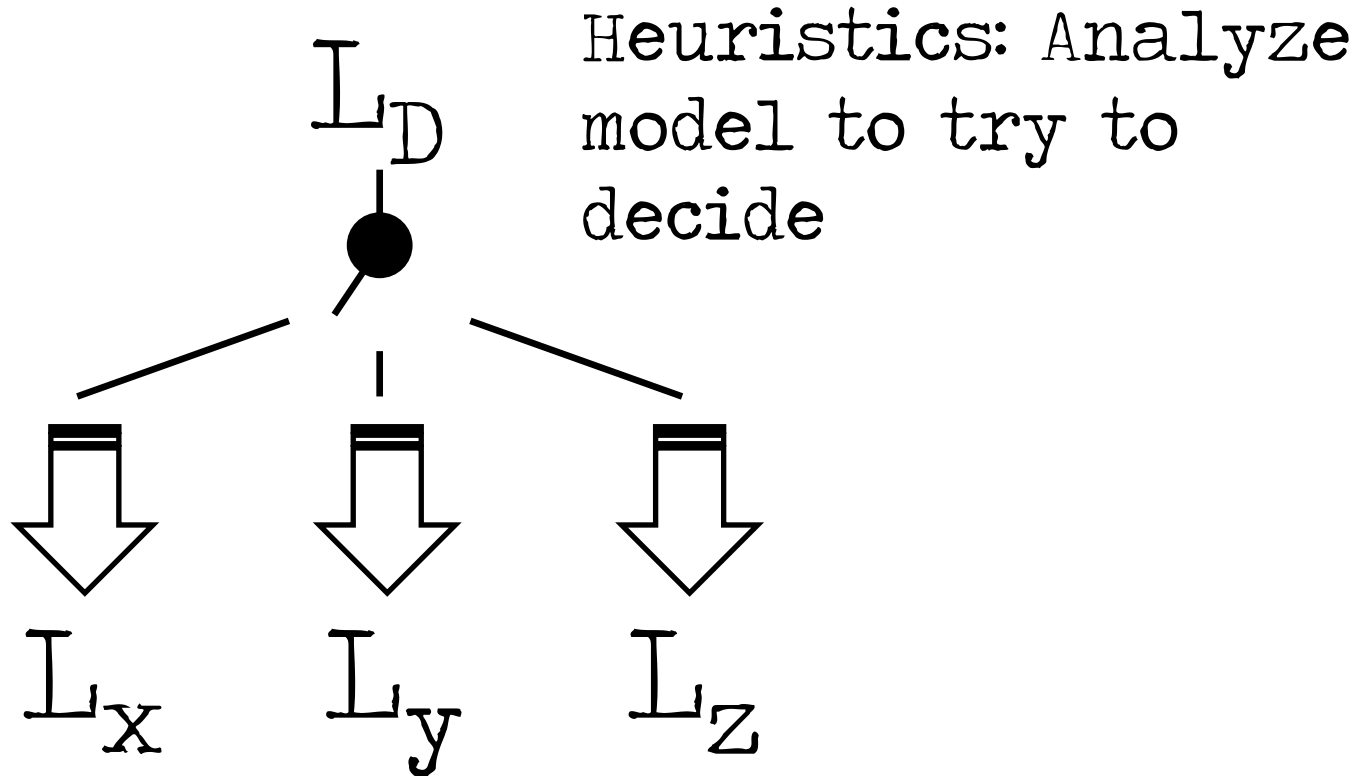


Example

Pension
Plans

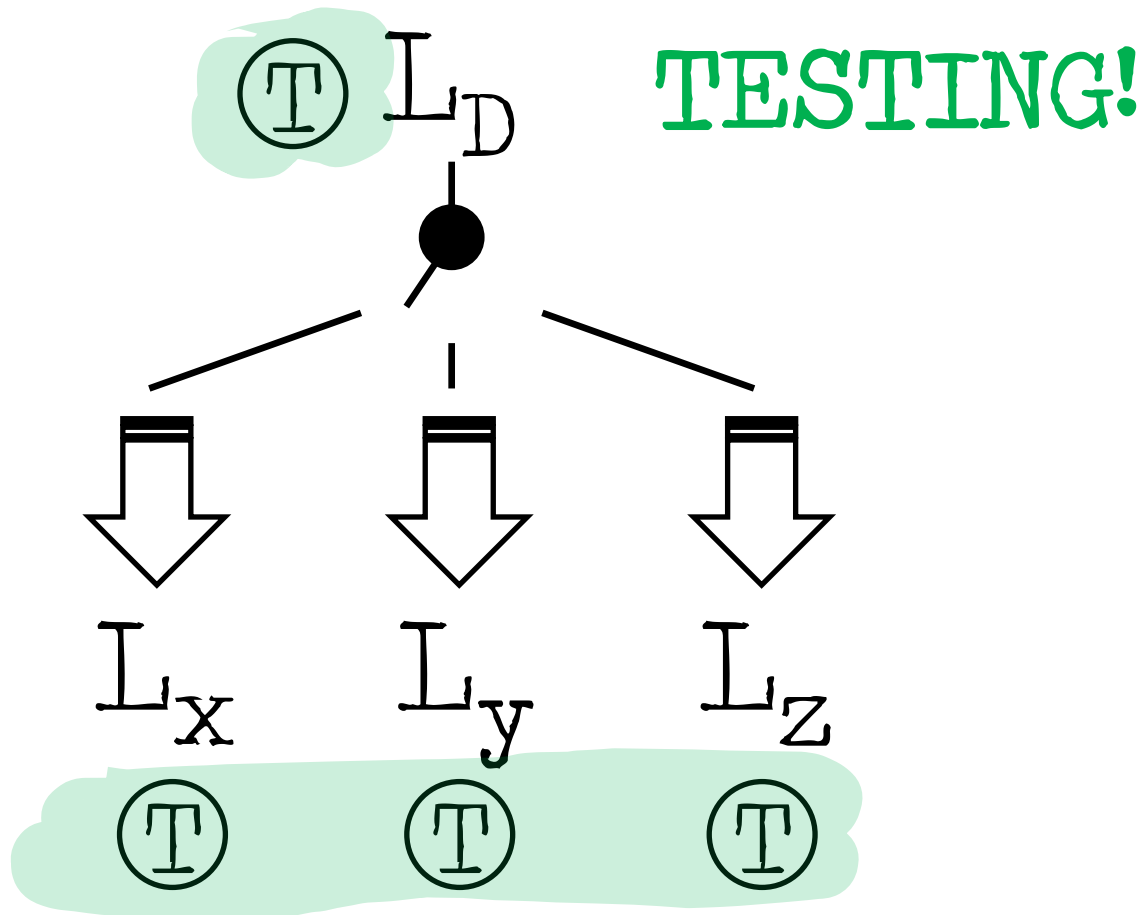
Multiple Mappings

... alternatively, selectably



Multiple Mappings

... alternatively, selectably



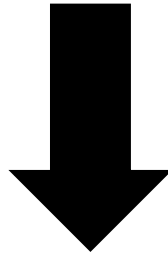
Reduced Expressiveness

bad?

maybe.

good?

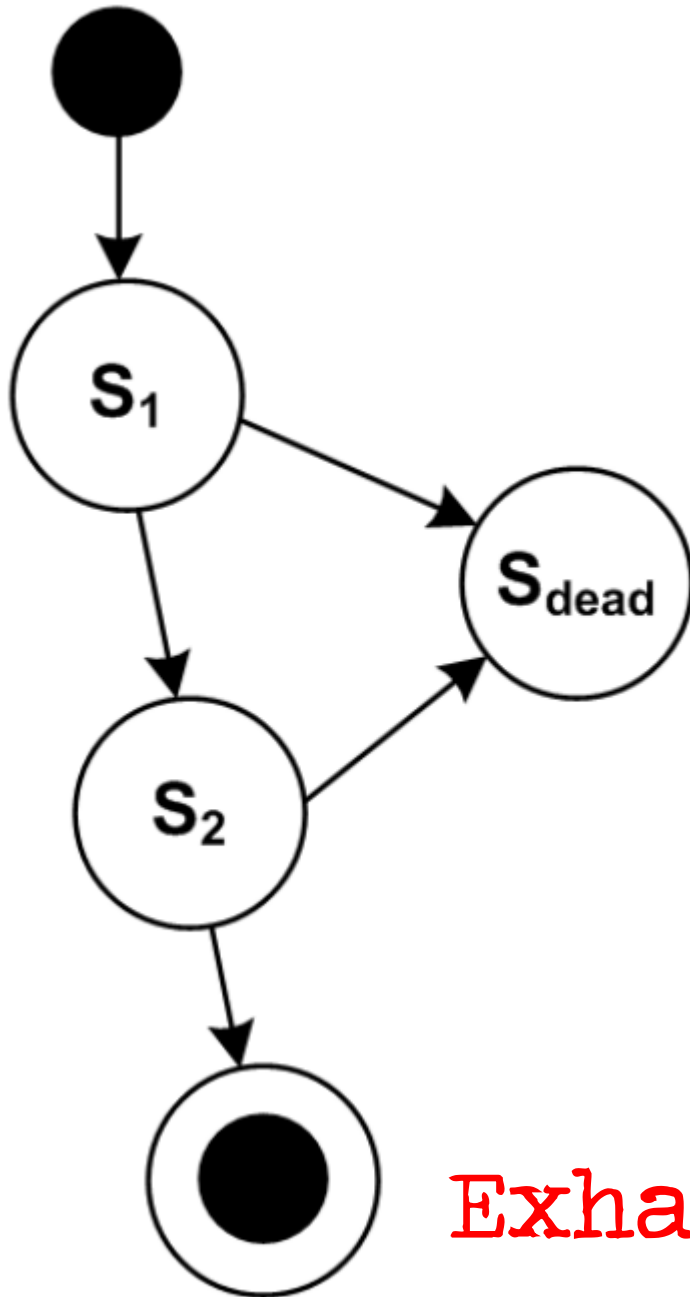
maybe!



Model Checking

SAT Solving

Exhaustive Search, Proof!



Unique State Names

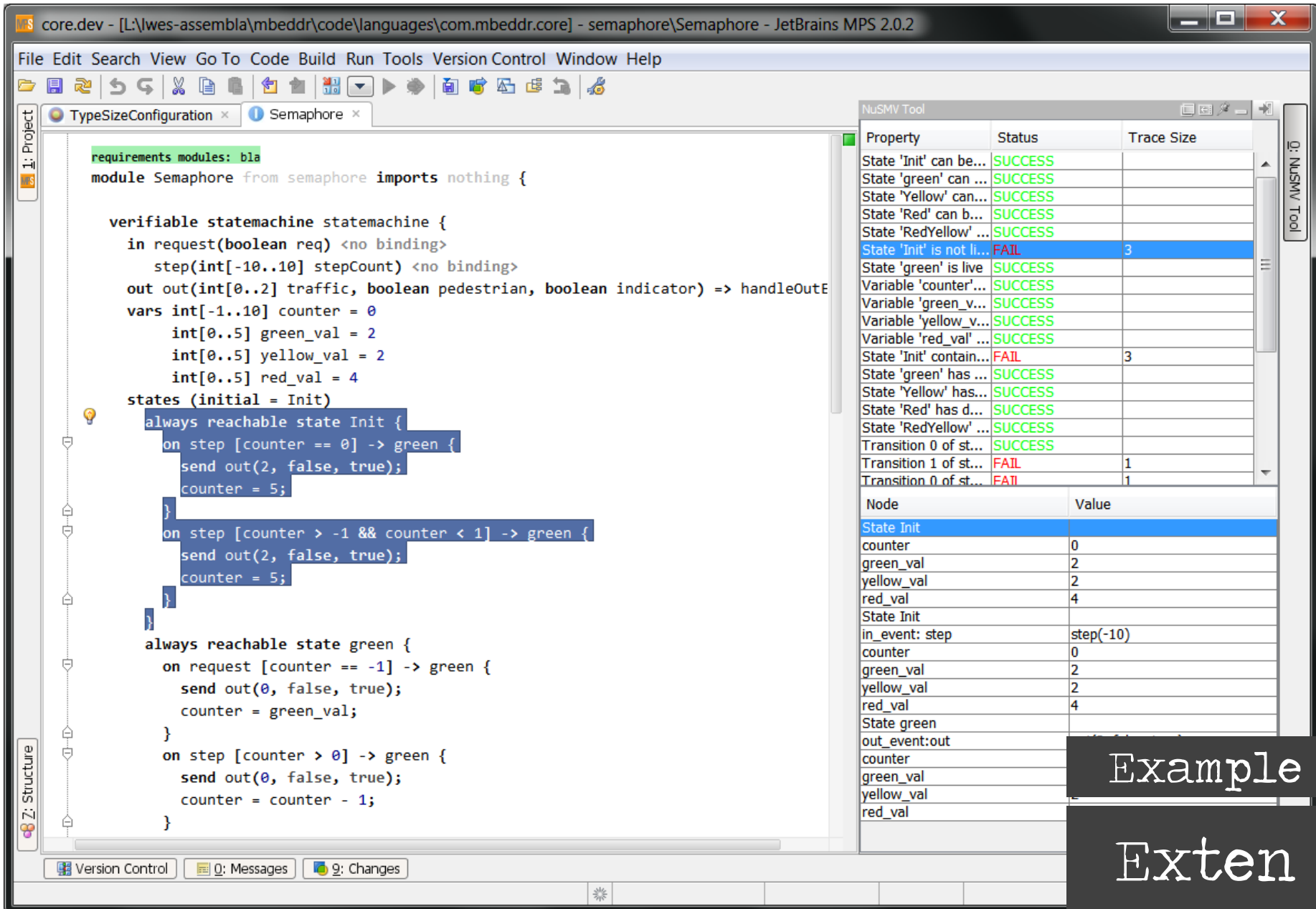
Unreachable States

Dead End States

Guard Decidability

Reachability

Exhaustive Search, Proof!



Example

Extended 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

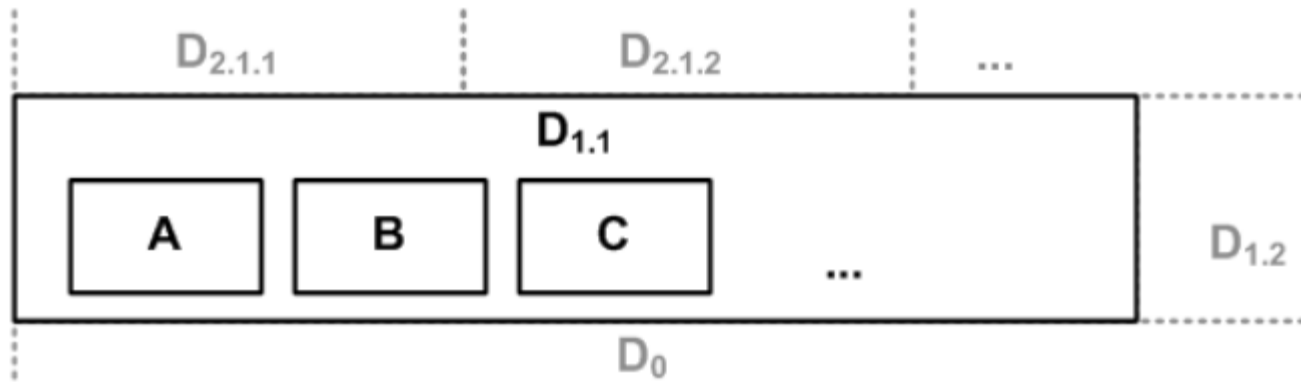
expressivity
coverage
semantics
separation of
concerns

completeness
paradigms
modularity
concrete
syntax

process

Several Concerns

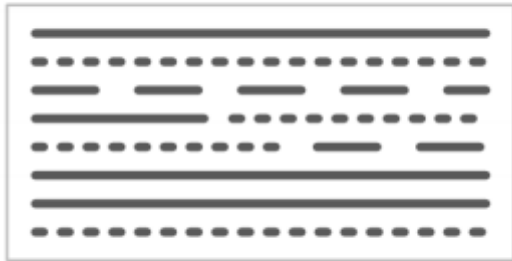
... in one domain



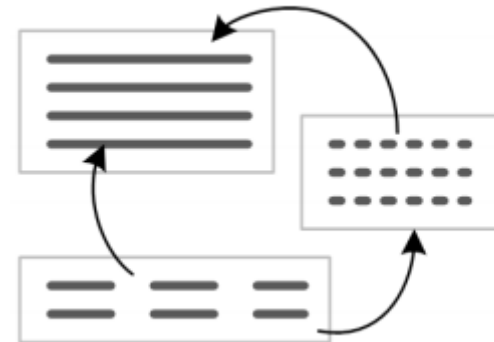
Several Concerns

... in one domain

integrated into
one fragment



separated into
several fragments

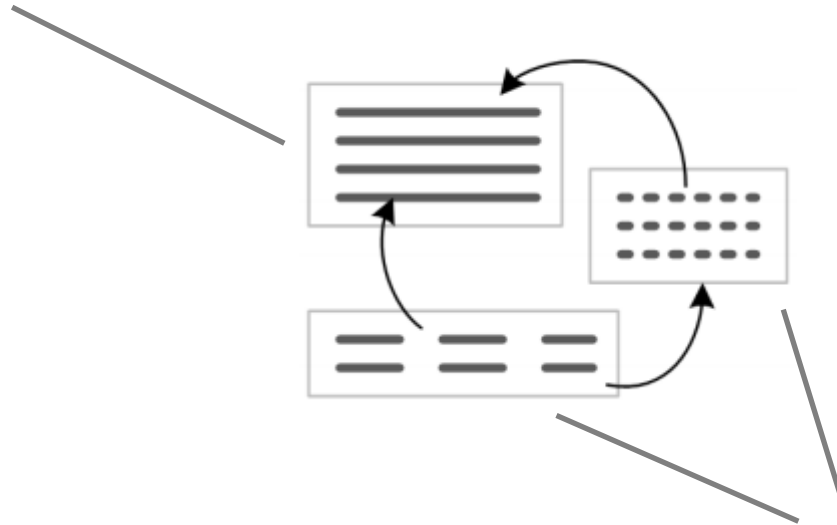


Viewpoints

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

independent

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



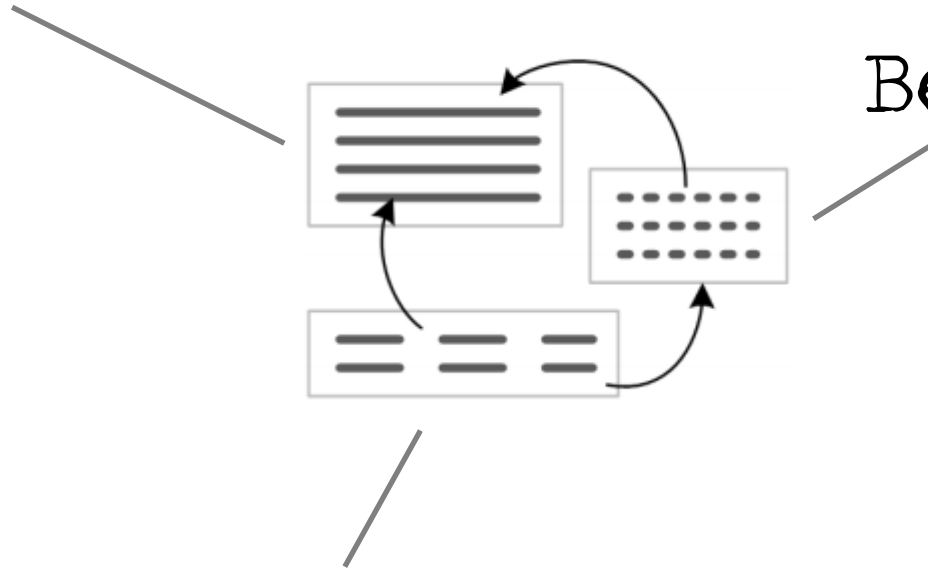
dependent

Viewpoints

Hardware

Behaviour

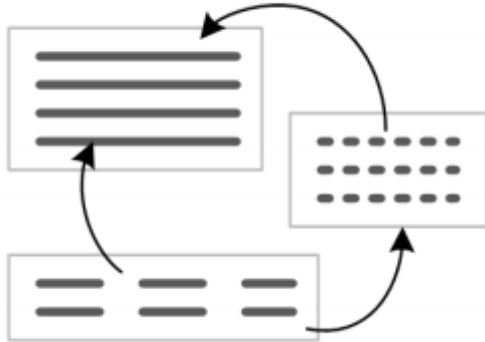
Tests



Example

Refrige
rators

Viewpoints: Why?



Sufficiency

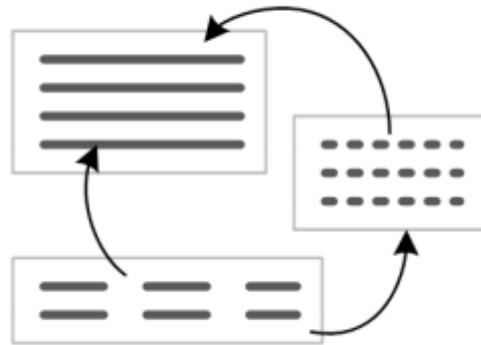
Different Stakeholders

Different Steps in
Process - VCS unit!

Viewpoints

Hardware

Product
Management



Behaviour

Thermo-
dynamics-
Experts

Tests

Example

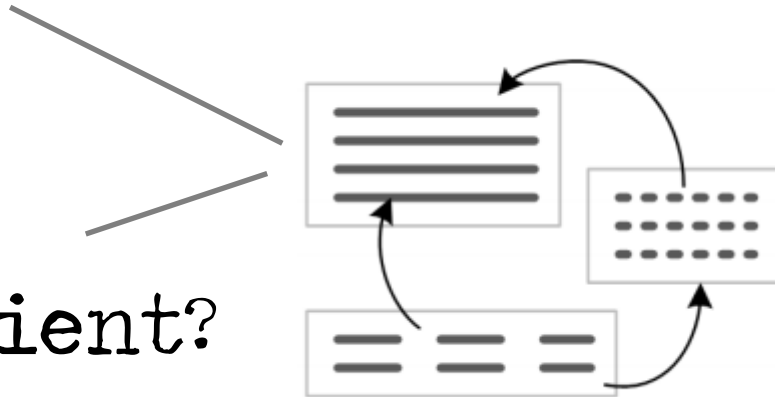
Refrige-
rators

Viewpoints

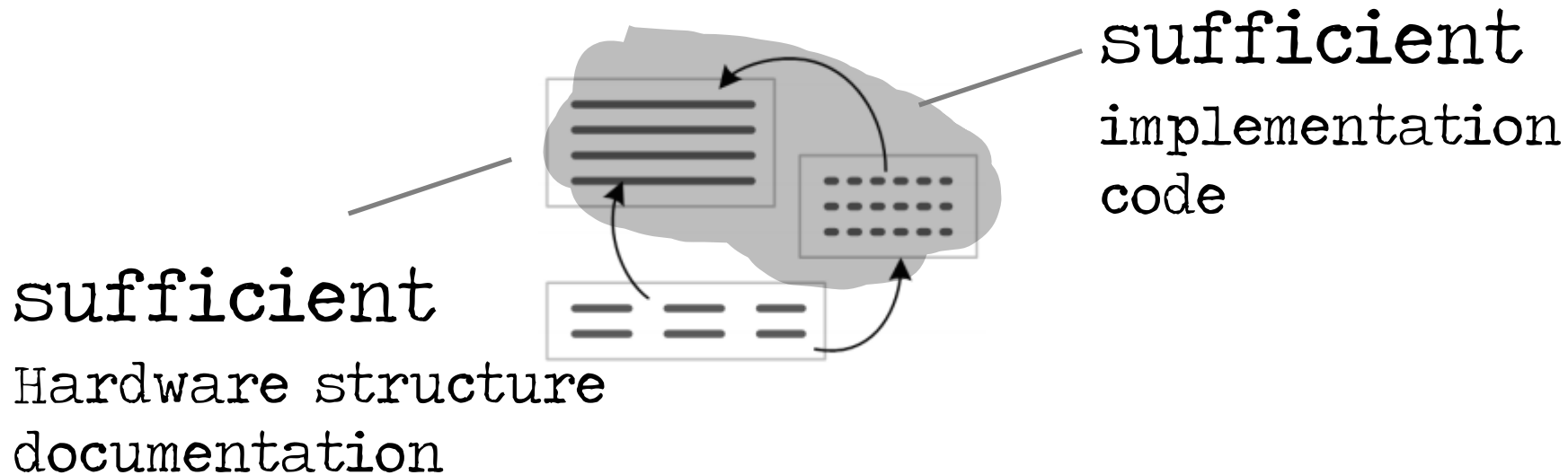
independent

sufficient?

contains all
the data for
running a
meaningful
transformation



Viewpoints

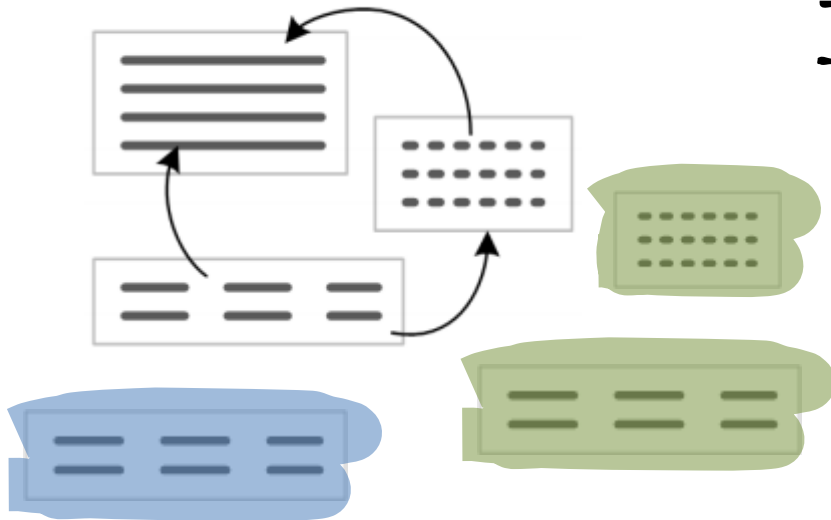


Example

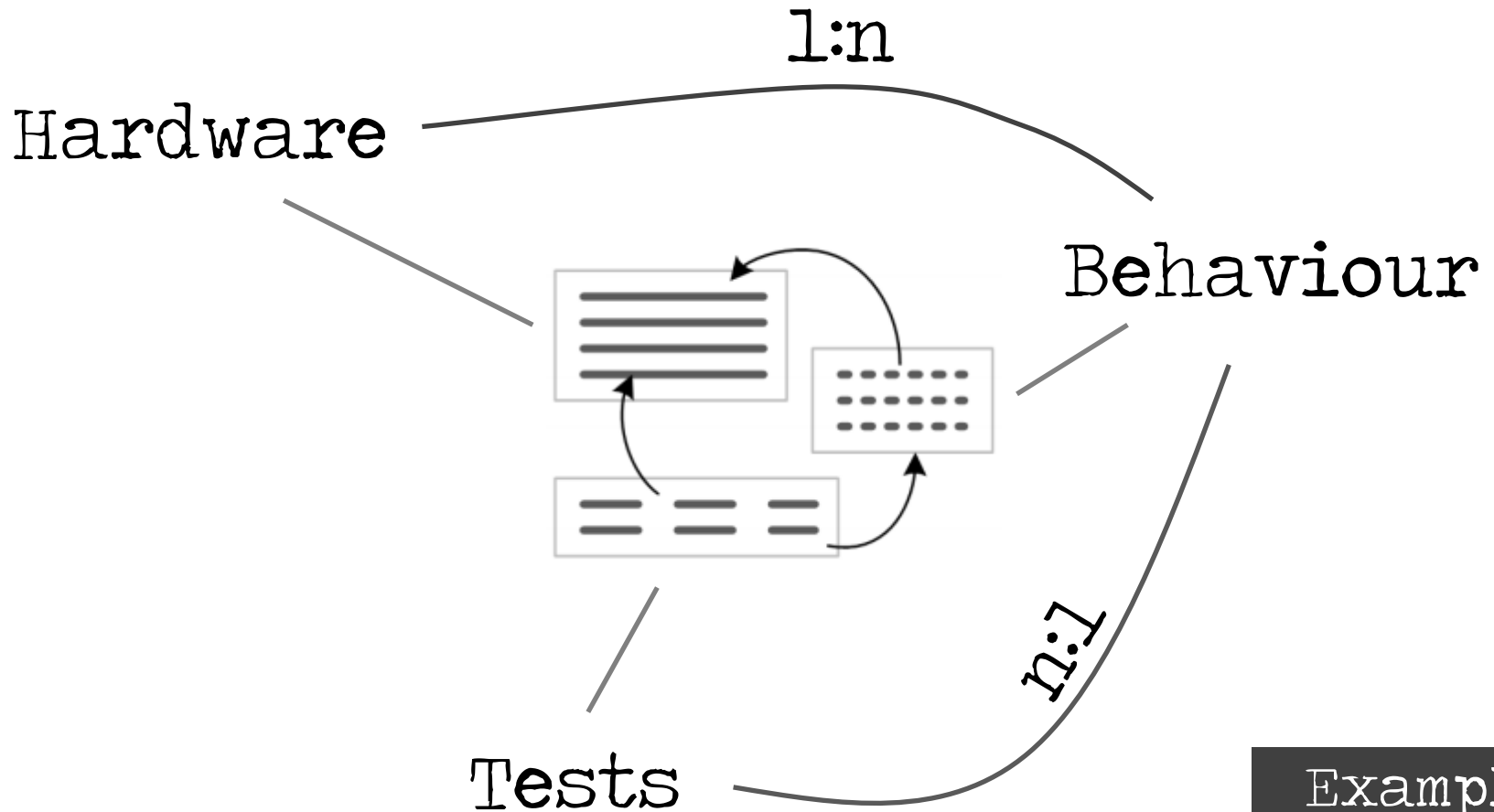
Refrige
rators

Viewpoints: Why?

1:n Relationships



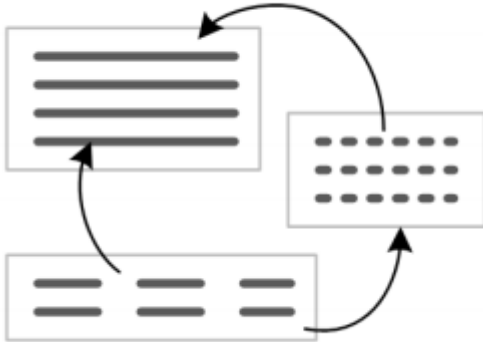
Viewpoints



Example

Refrige
rators

Viewpoints

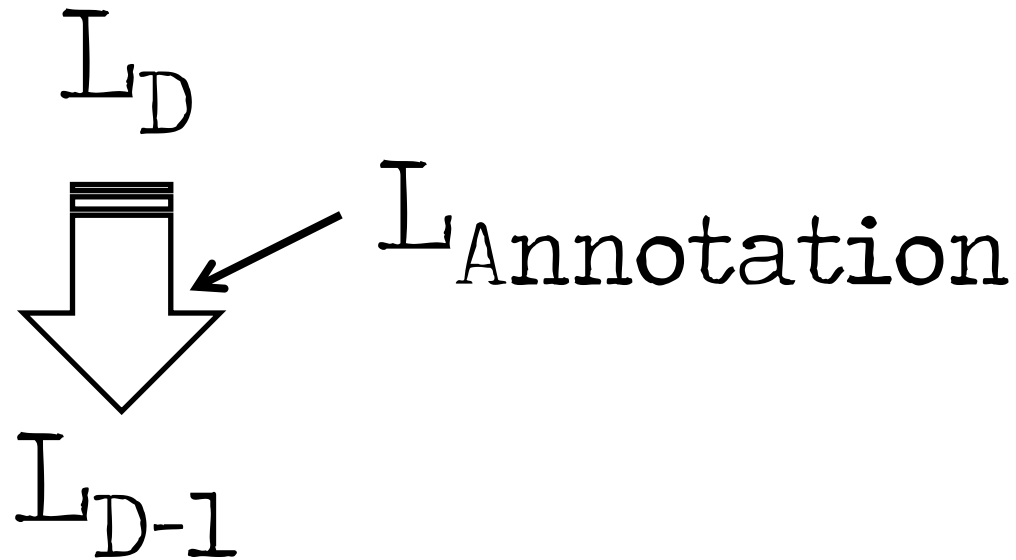


Well-defined
Dependencies

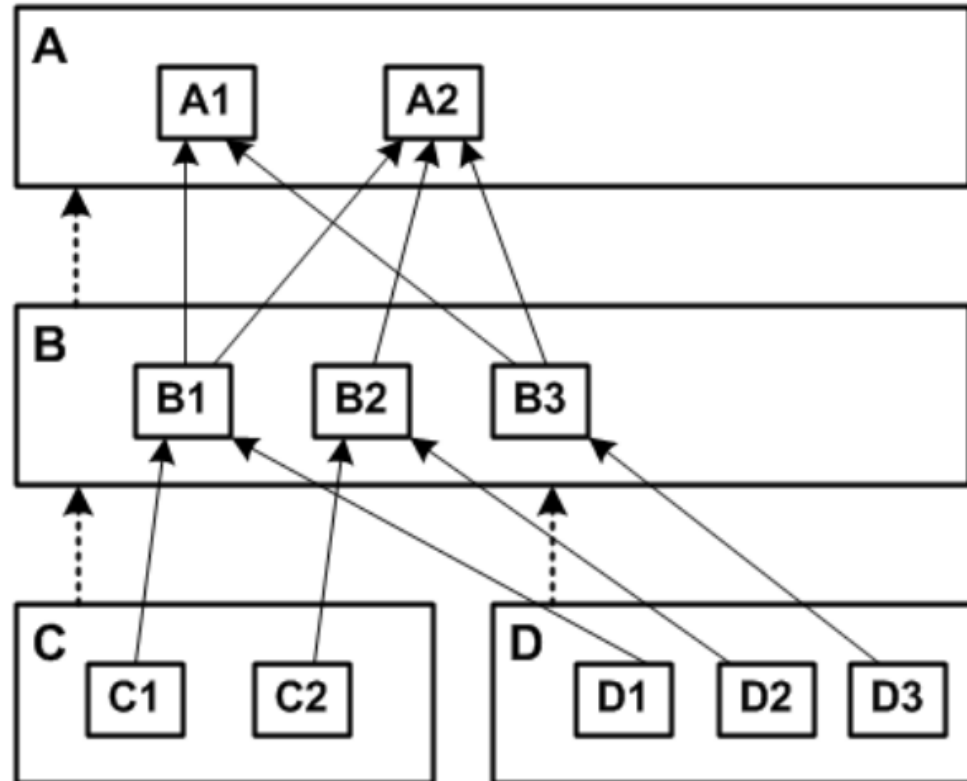
No Cycles!

Avoid Synchronization!
(unless you use a projectional editor)

Viewpoints: Why?



Progressive Refinement



Views on Programs

Achmea demo plan

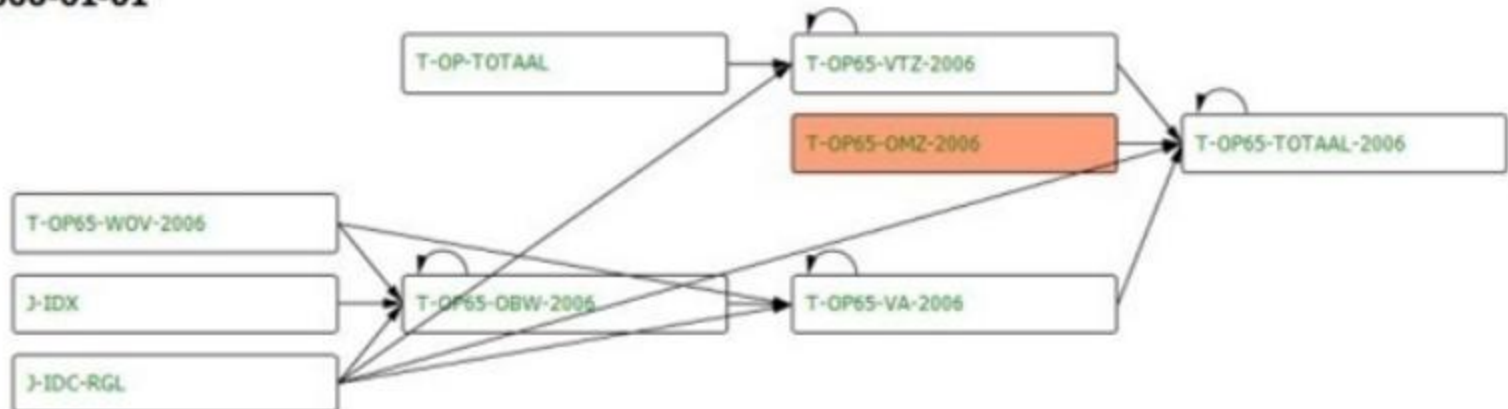
▣ T-OP65-TOTAAL-2006

Het totale ouderdomspensioen, opgebouwd in de oude of de nieuwe regeling

▣ 1999-01-01



▣ 2006-01-01



Example

Pension
Plans

Completeness

expressivity
coverage
semantics
separation of
concerns

completeness
paradigms
modularity
concrete
syntax

process

NOTICE



THIS PART OF THE
SLIDE DECK
HAS BEEN SKIPPED.

PLEASE REFER TO THE
DSL ENGINEERING BOOK.

Fundamental Paradigms

expressivity
coverage
semantics
separation of
concerns

completeness
paradigms
modularity
concrete
syntax

process

NOTICE



THIS PART OF THE
SLIDE DECK
HAS BEEN SKIPPED.

PLEASE REFER TO THE
DSL ENGINEERING BOOK.

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:

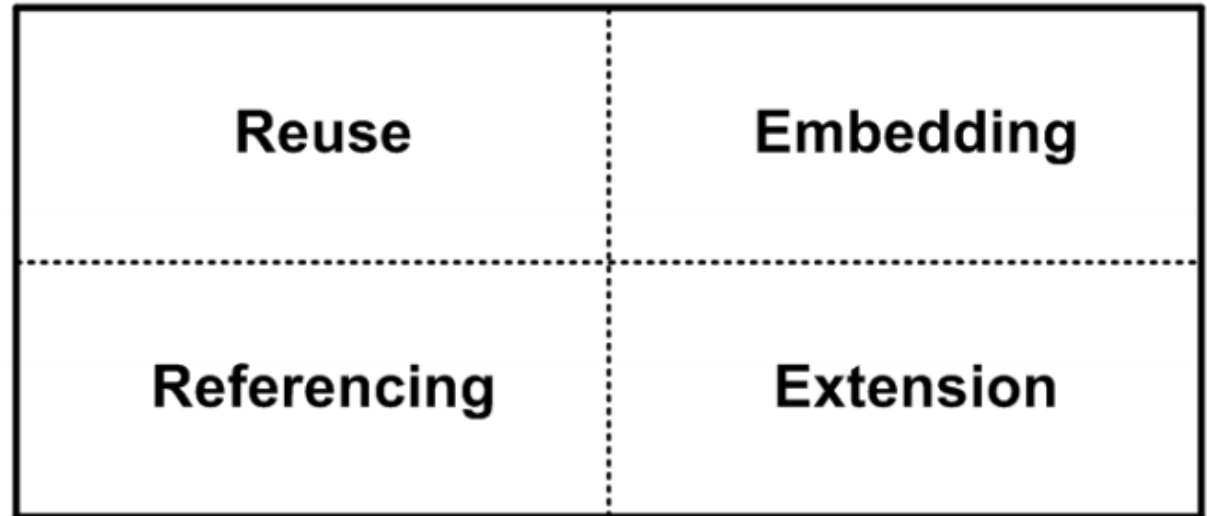
do we have to know about the reuse when designing the languages?

Fragment Structure:

homogeneous vs. heterogeneous
(„mixing languages“)

Dependencies & Fragment Structure:

independent
languages
dependencies
dependent



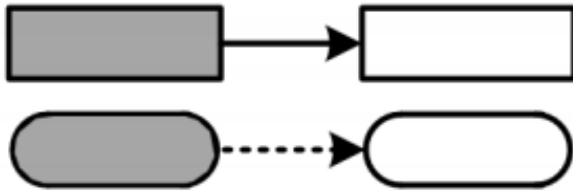
homogeneous

heterogeneous

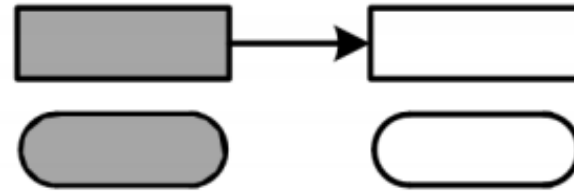
fragment structure

Dependencies & Fragment Structure:

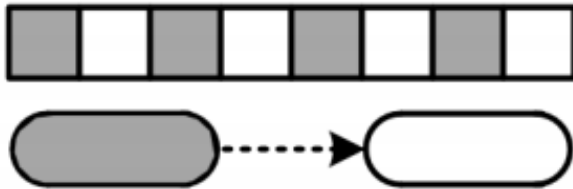
Referencing



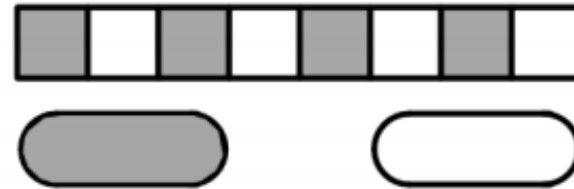
Reuse



Extension



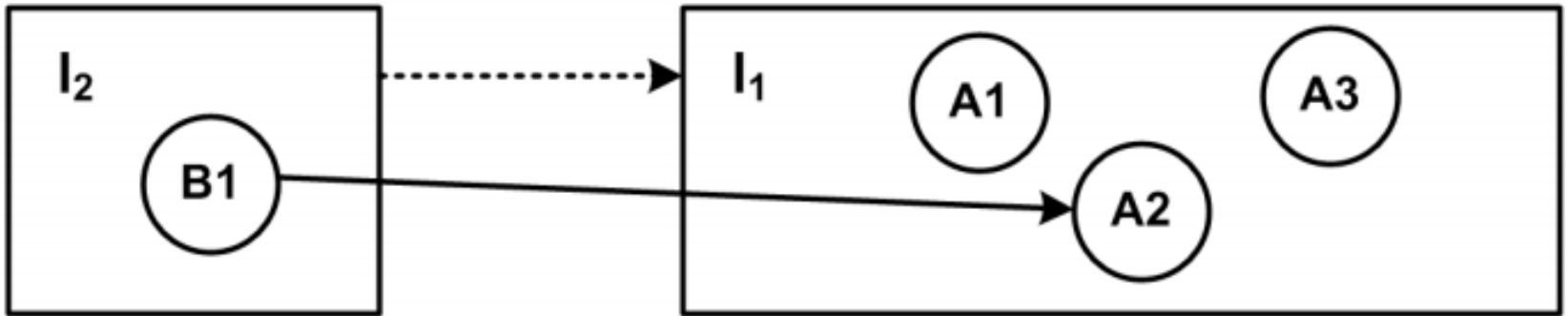
Embedding



Referencing

independent
languages
dependencies
dependent

Reuse	Embedding
Referencing	Extension
homogeneous	heterogeneous
fragment structure	



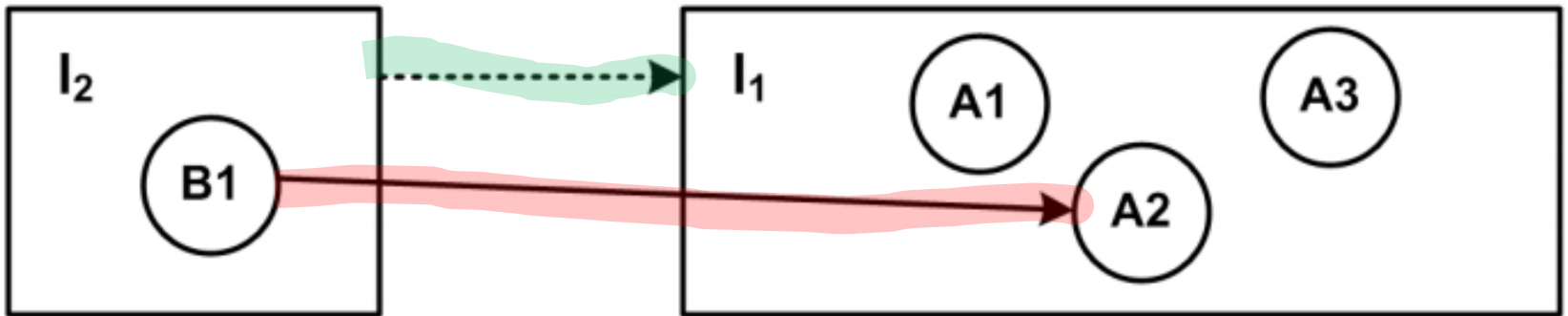
Referencing

independent
languages
dependencies
dependent

Reuse	Embedding
Referencing	Extension

homogeneous heterogeneous
fragment structure

Dependent



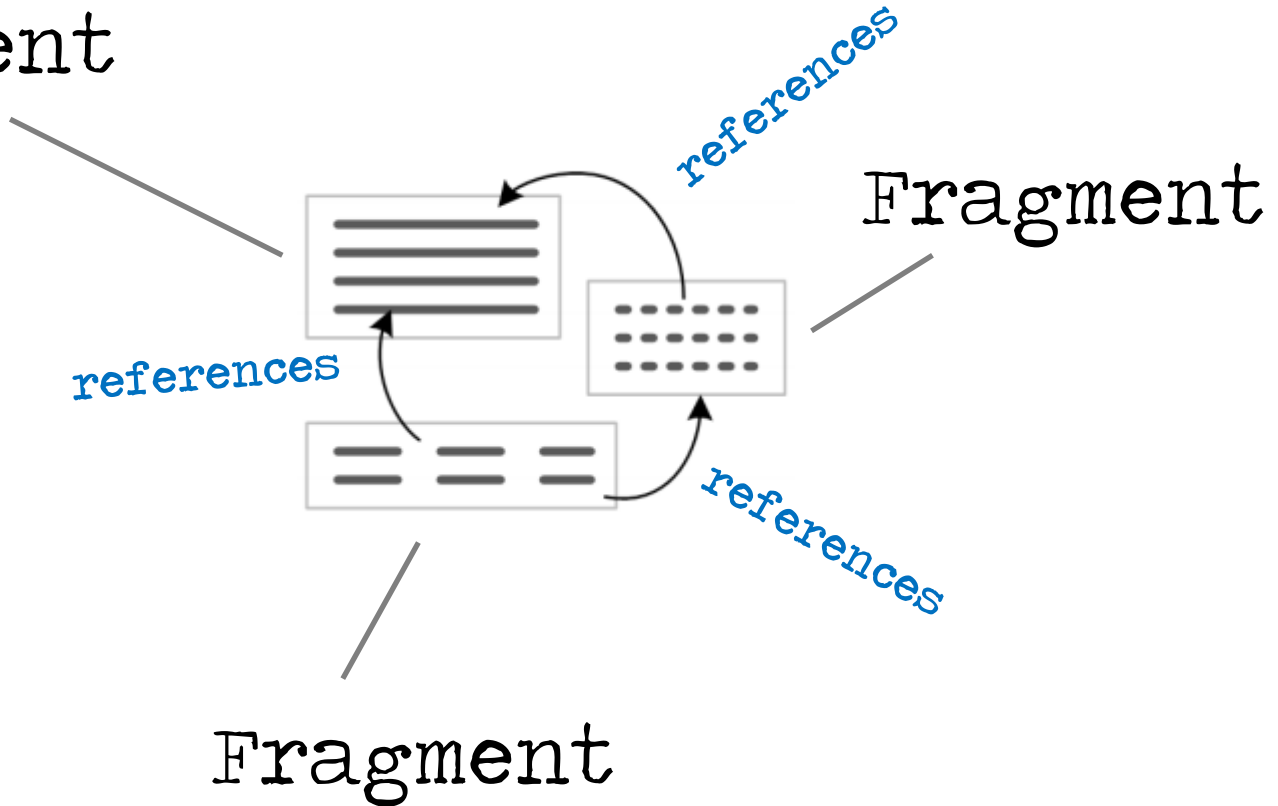
No containment

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

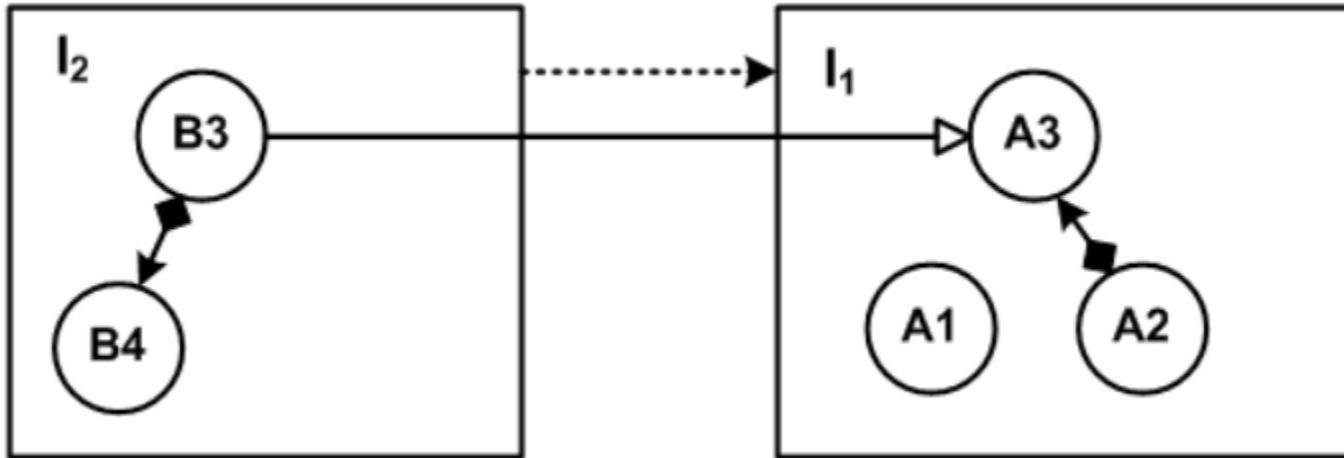
Refrige
rators

Extension

independent
languages
dependencies
dependent

Reuse	Embedding
Referencing	Extension

homogeneous heterogeneous
fragment structure



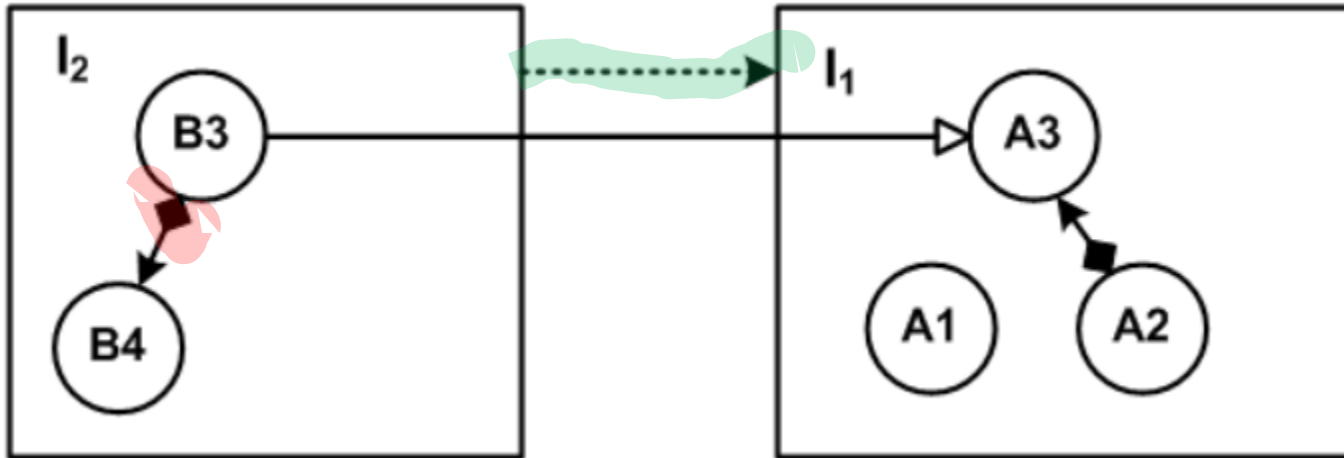
Extension

independent
languages
dependencies
dependent

Reuse	Embedding
Referencing	Extension

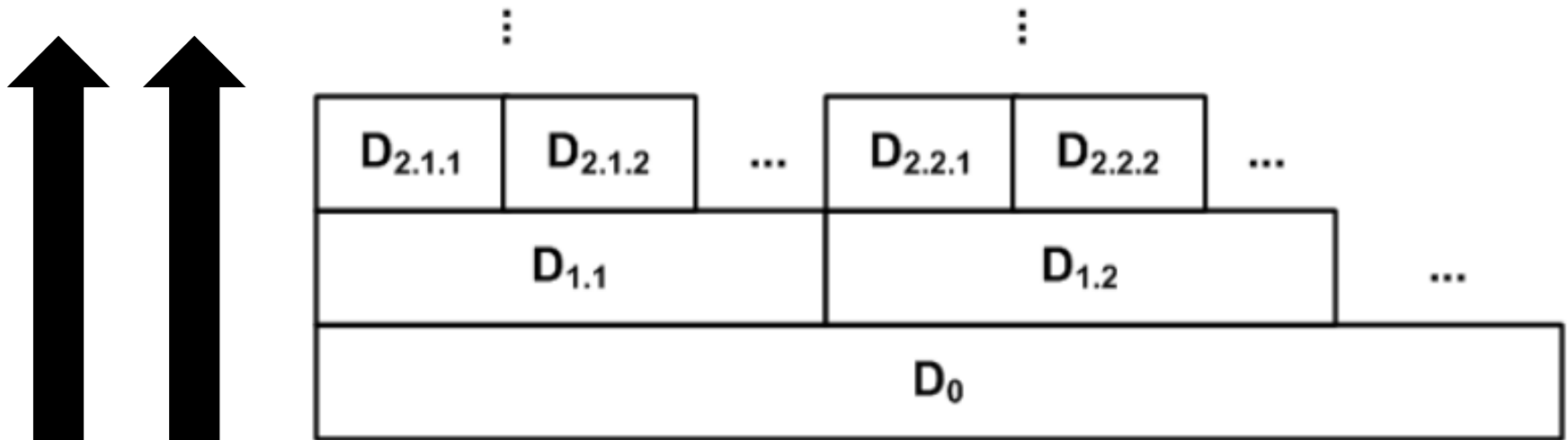
homogeneous heterogeneous
fragment structure

Dependent



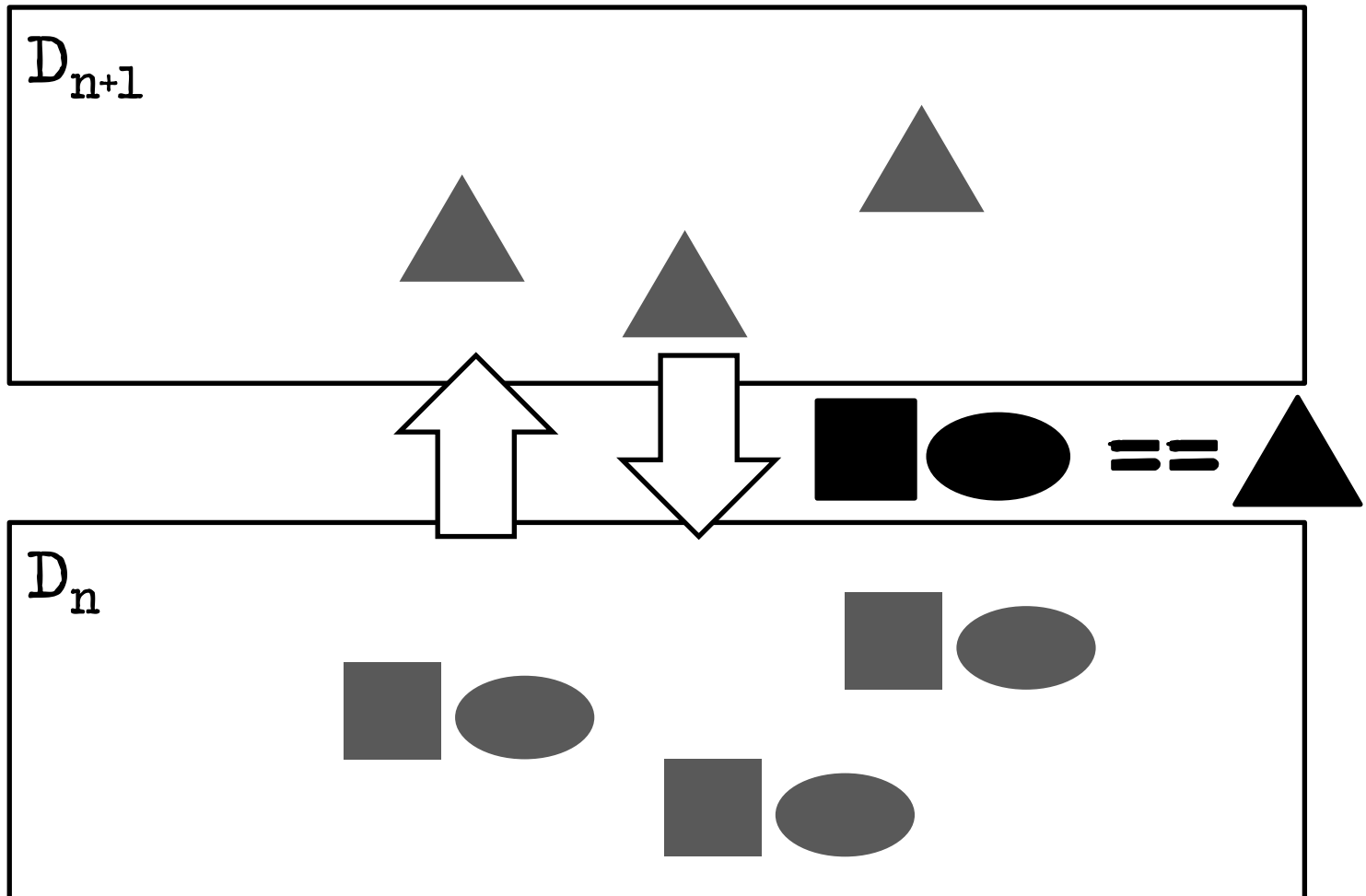
Containment

Extension

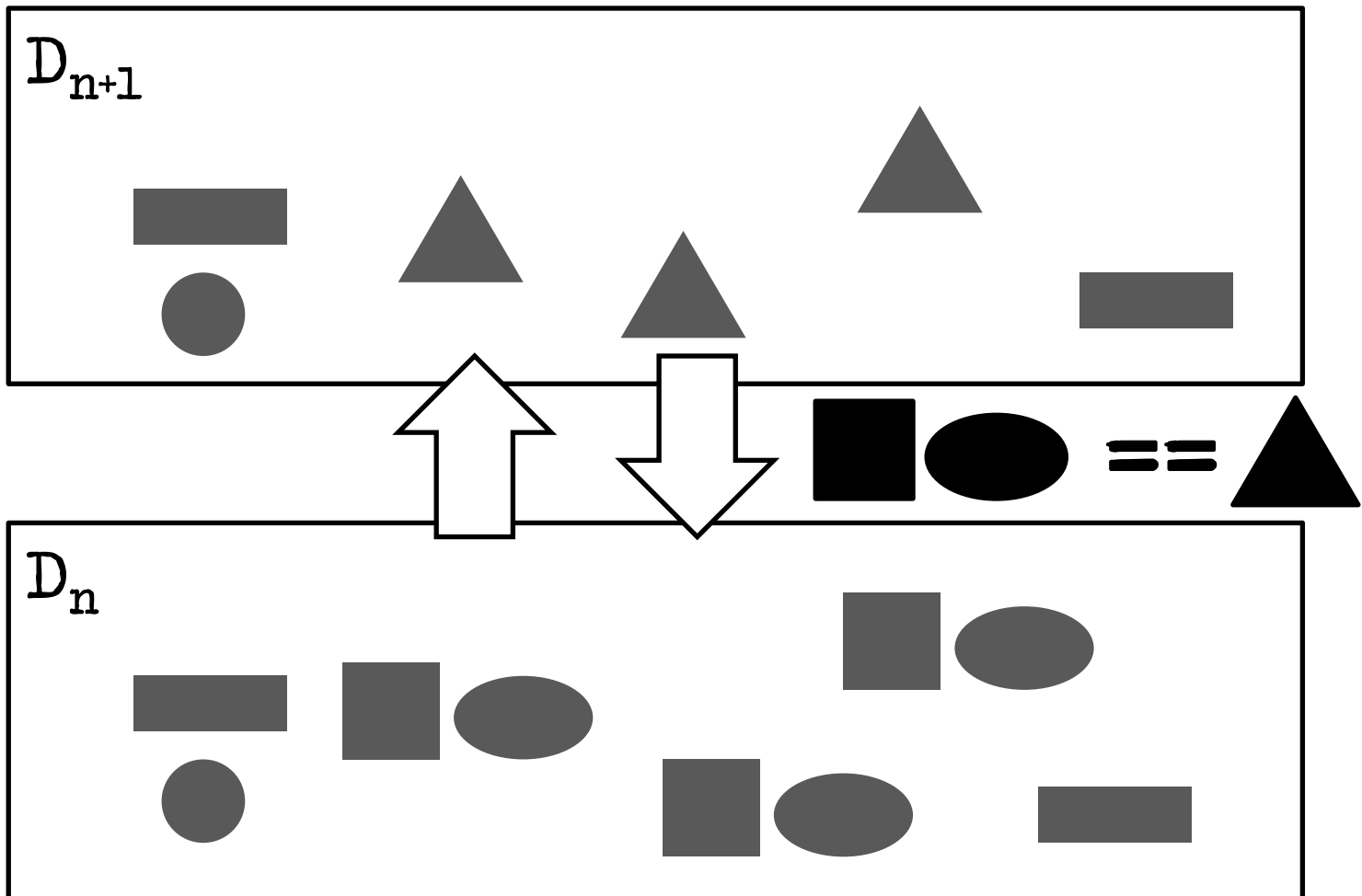


more specialized domains
more specialized languages

Extension

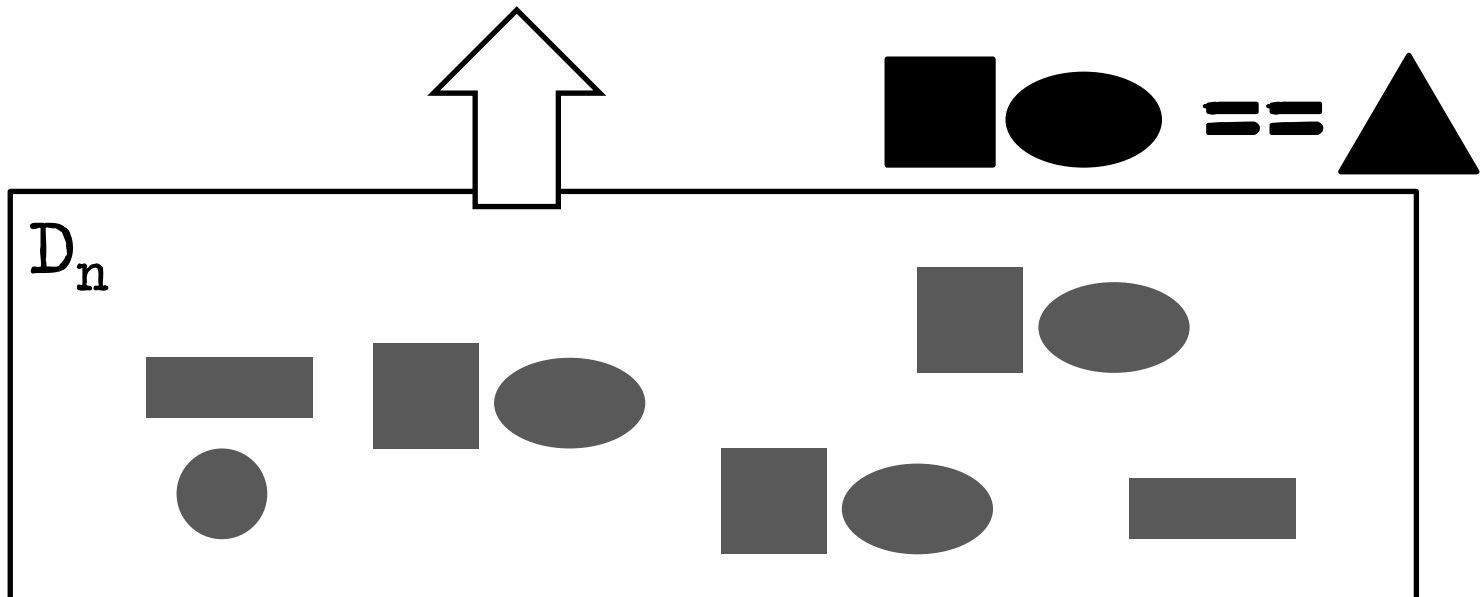


Extension



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);

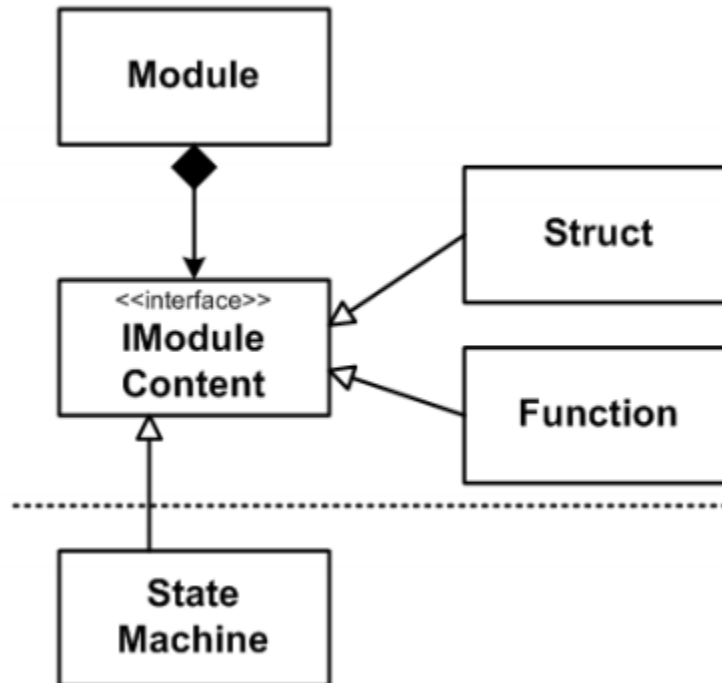
        task run cyclic prio = 1 every = 2 {
            stateswitch linefollower
                state running
                    int32 light = 0;
                    light = ecrobot_get_light_sensor
                        (SENSOR_PORT_T::NXT_PORT_S1);
                    if ( light < ( WHITE + BLACK ) / 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

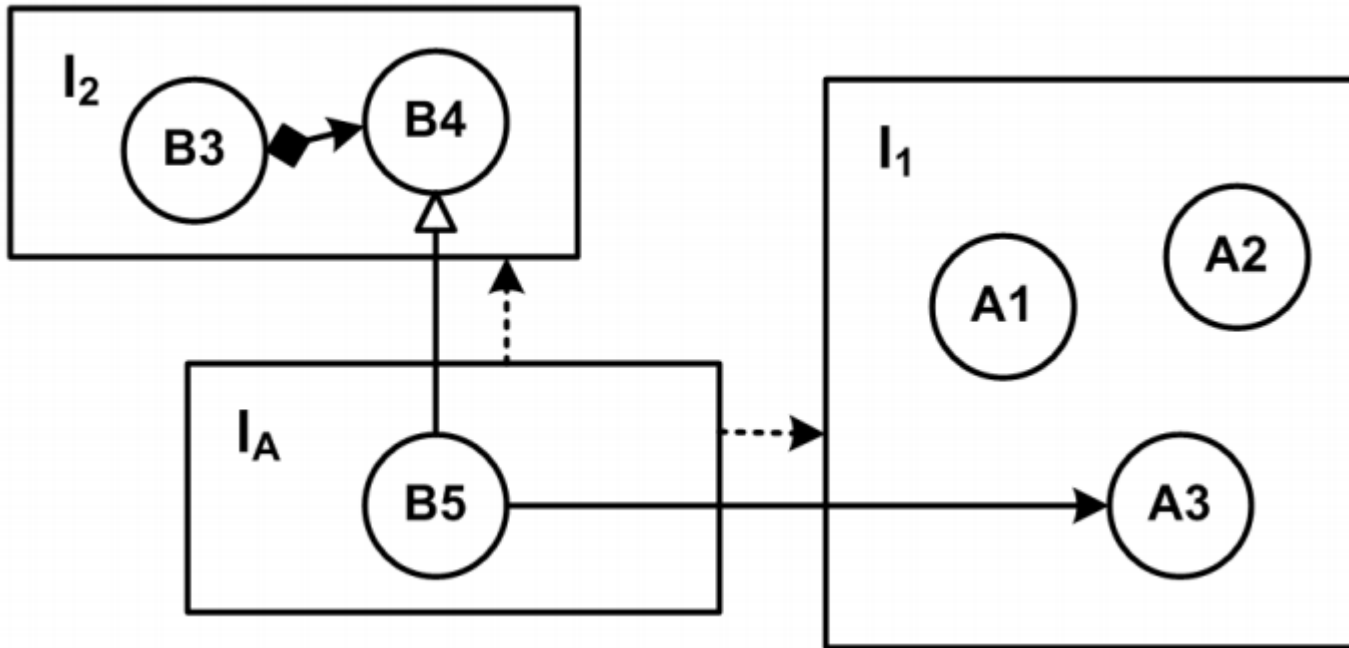
Extended C

Reuse

independent
languages
dependencies
dependent

Reuse	Embedding
Referencing	Extension

homogeneous heterogeneous
fragment structure



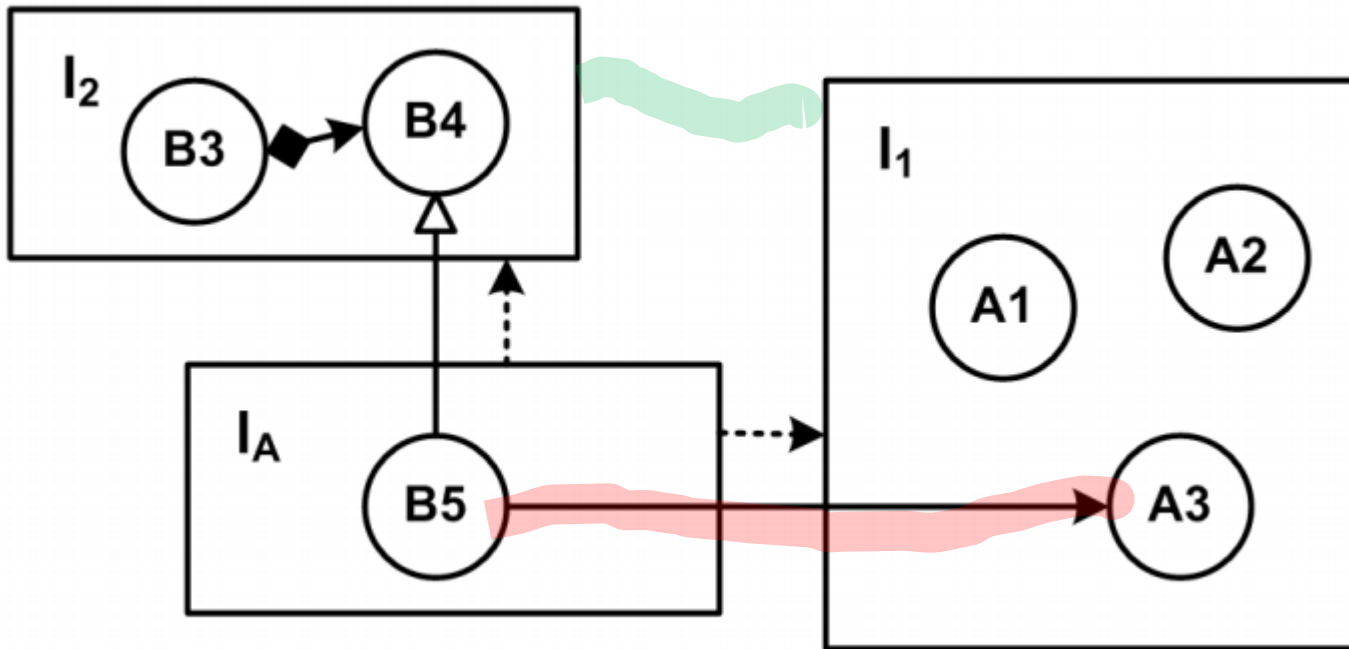
Reuse

independent
languages
dependencies
dependent

Reuse	Embedding
Referencing	Extension

homogeneous heterogeneous
fragment structure

Independent



No containment

Reuse

Often the referenced language is built expecting it will be reused.

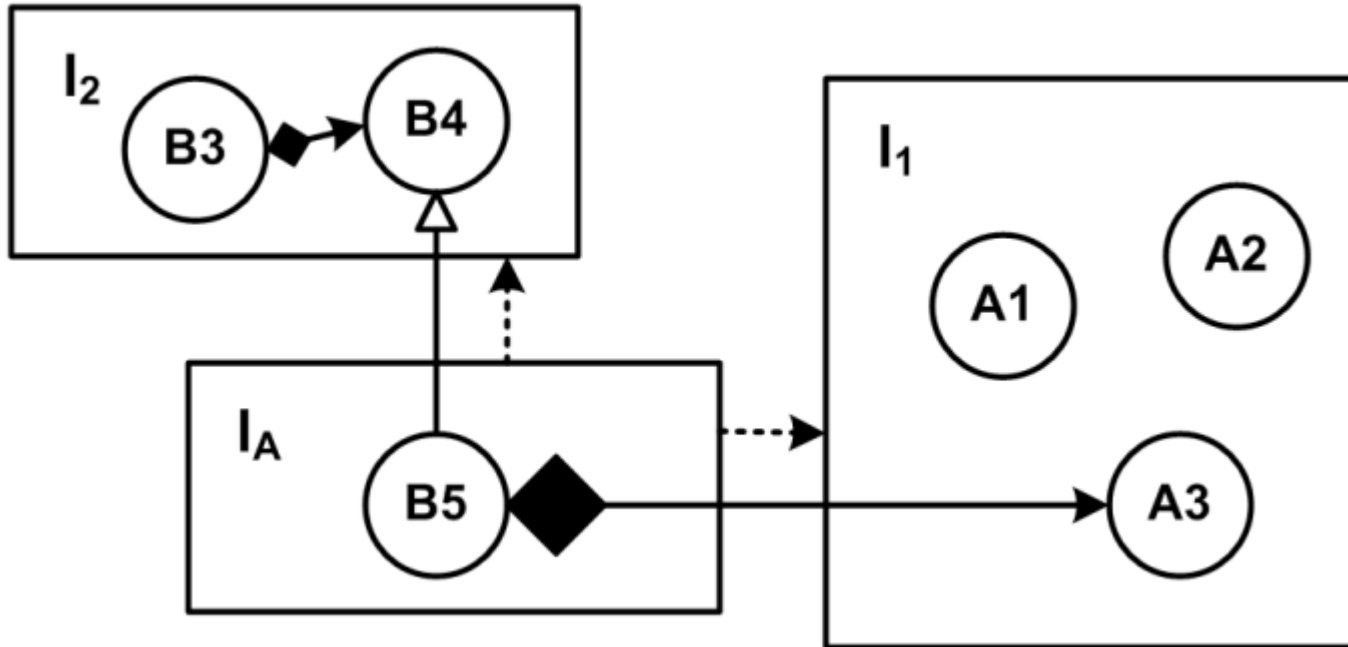
Hooks may be added.

Embedding

independent
languages
dependencies
dependent

Reuse	Embedding
Referencing	Extension

homogeneous heterogeneous
fragment structure

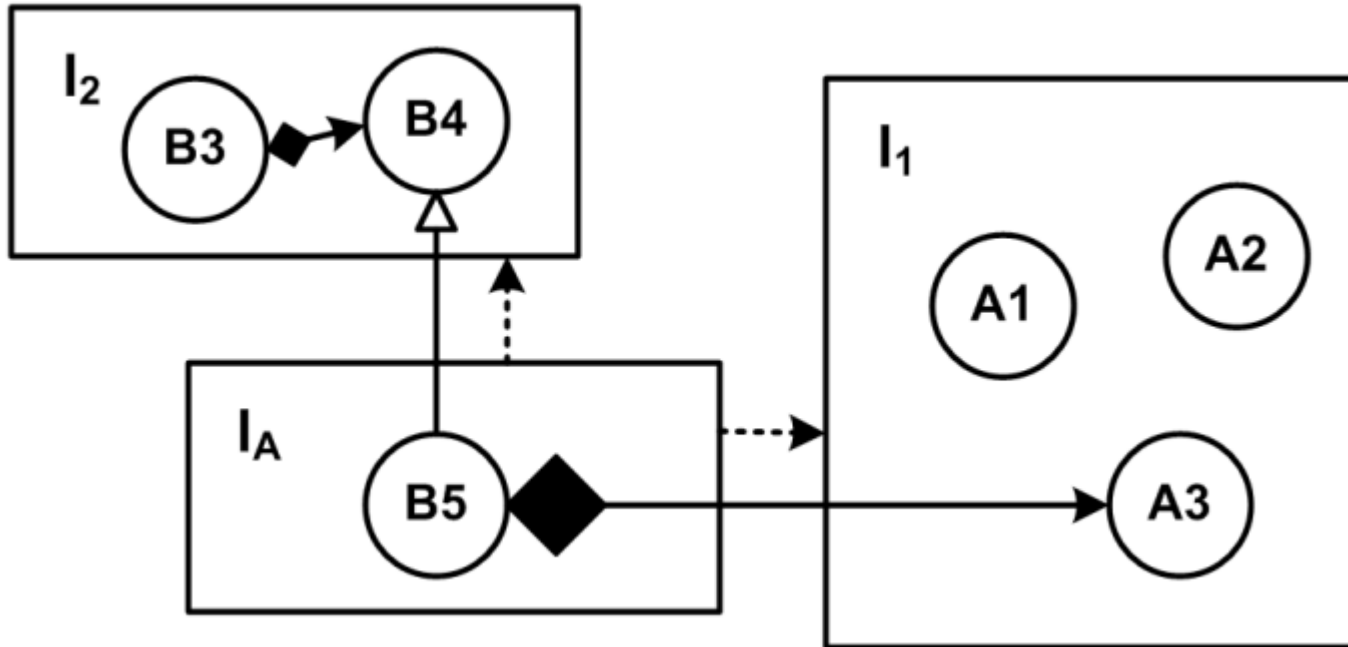


Embedding

independent
languages
dependencies
dependent

Reuse	Embedding
Referencing	Extension

homogeneous heterogeneous
fragment structure



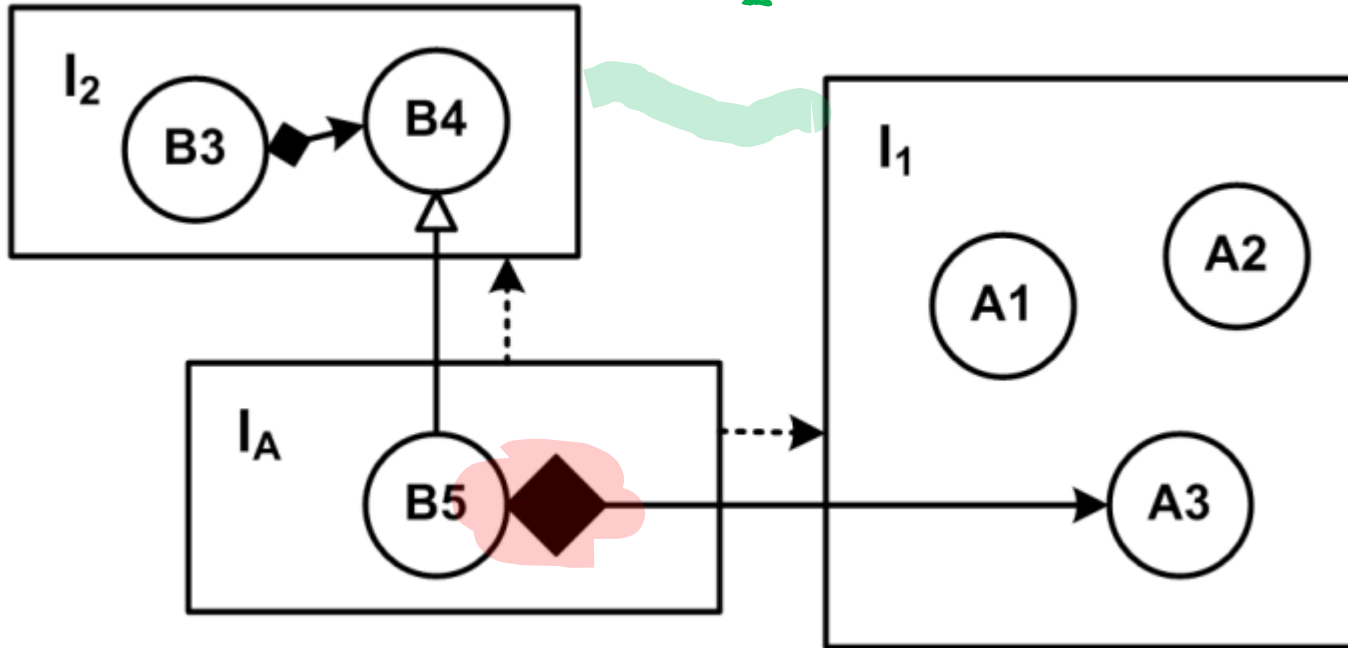
Embedding

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 documentation states: '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.'

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 'Library' section containing various rules and tag definitions.

Example

Pension
Plans

Embedding

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

NOTICE



THIS PART OF THE
SLIDE DECK
HAS BEEN SKIPPED.

PLEASE REFER TO THE
DSL ENGINEERING BOOK.

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

... then add other forms,
if really necessary

Default: Text

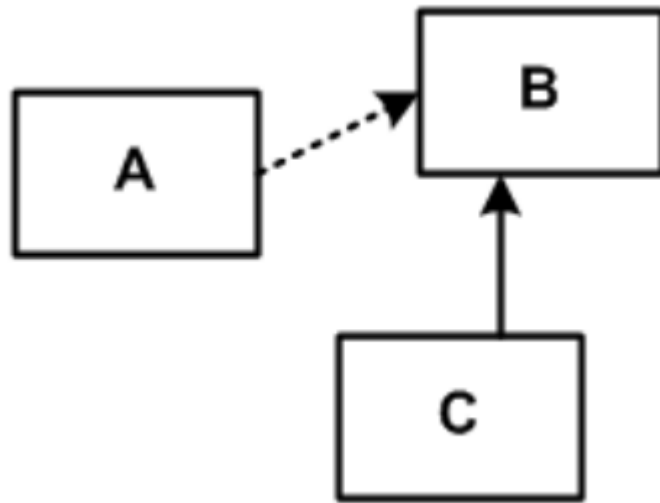
Editors simple to build

Productive

Easy to integrate w/ tools

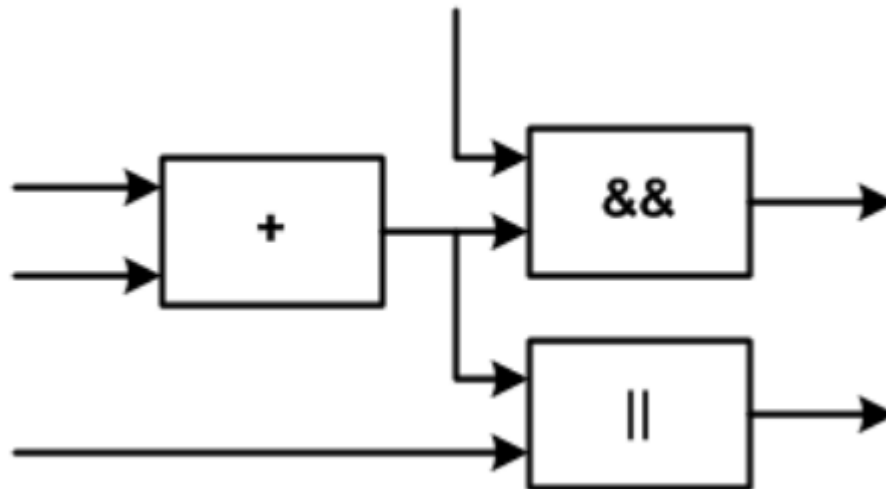
Easy to evolve programs

Graphical in case...



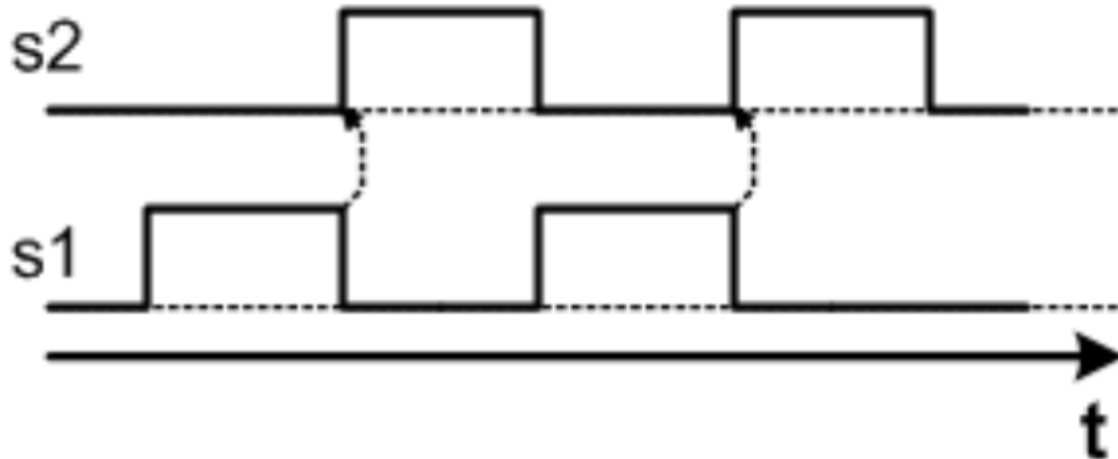
Relationships

Graphical in case..



Flow and
Dependency

Graphical in case...



Causality
and Timing

Symbolic

Either
Mathematical,
or often
highly
inspired by
domain

▣ ✕ 3.3 *Commutatiegetallen op 1 leven* ¶

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

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

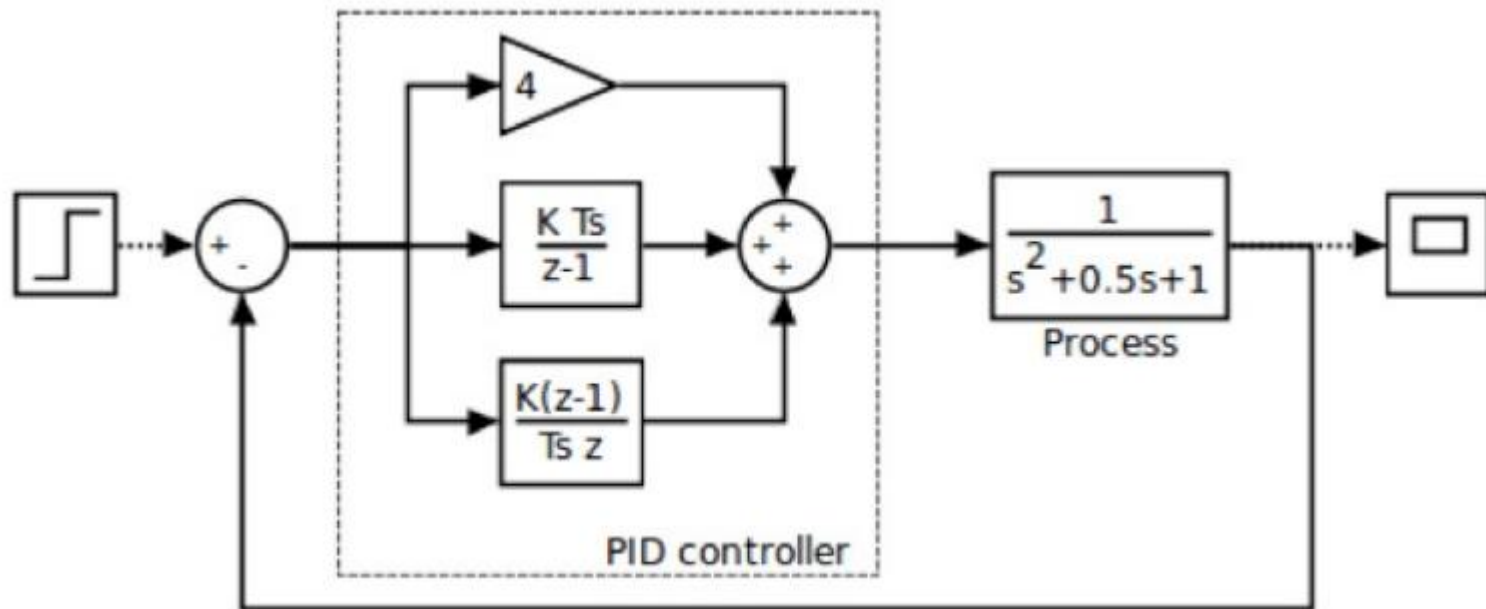
```
c/s interface Decider {
  int decide(int x, int y) pre
}

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

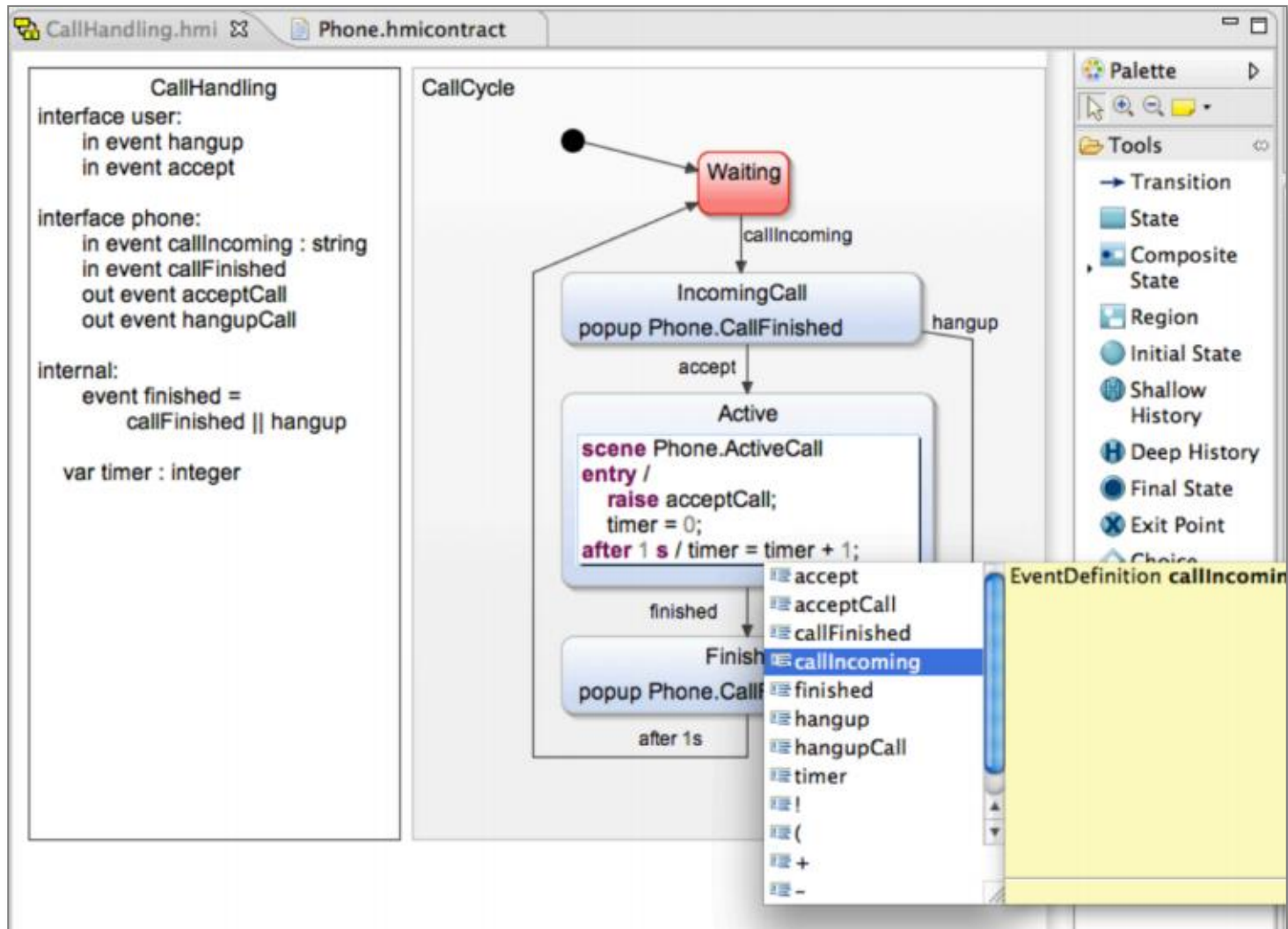
The screenshot shows a software window titled "Specification Document" with two tabs: "*RIF Model.rif-xmi" and "*Specification Document". The main content area is a table with two columns: "Description" and "Link".

	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, 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



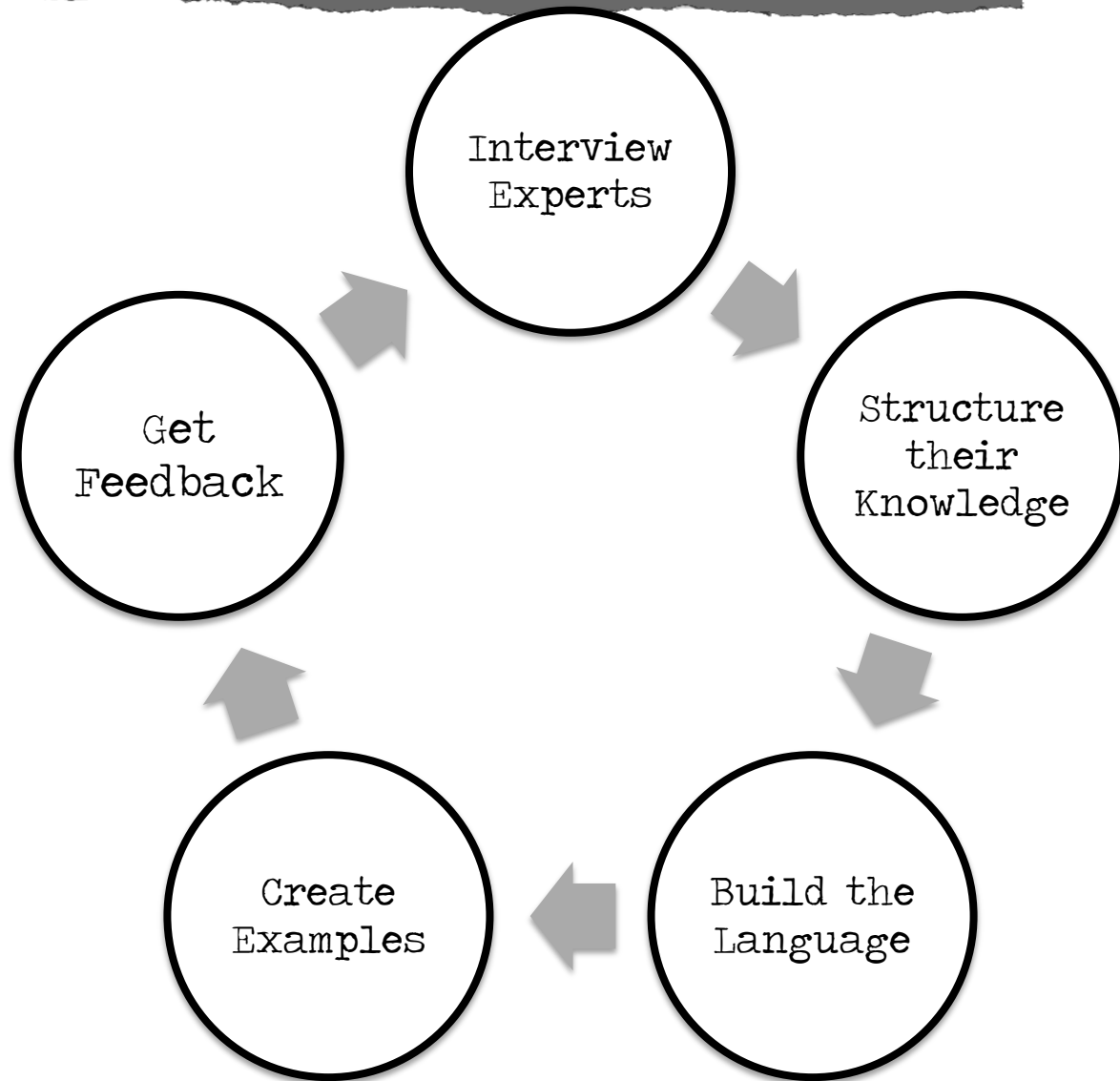
Process

expressivity
coverage
semantics
separation of
concerns

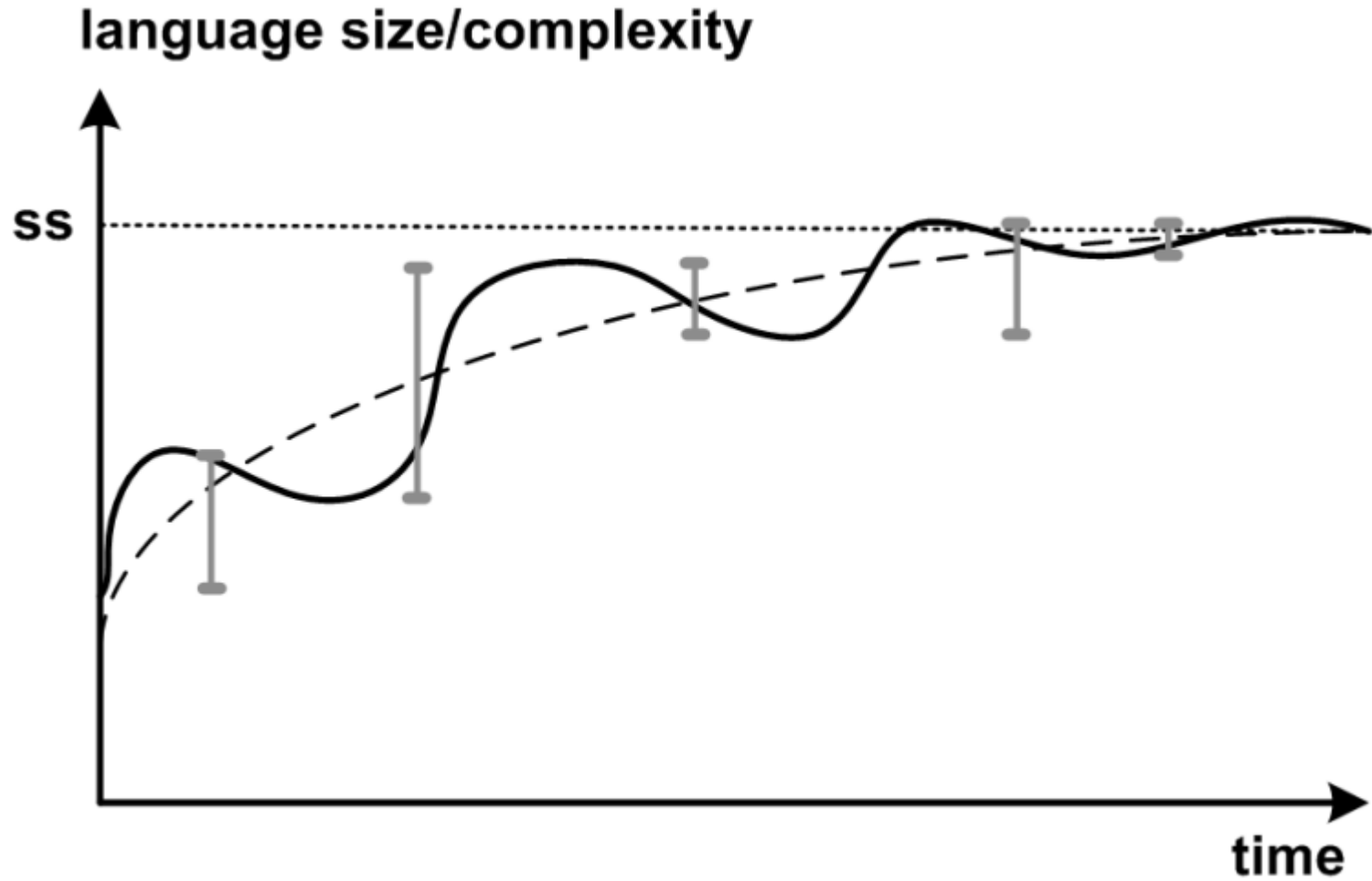
completeness
paradigms
modularity
concrete
syntax

process

Domain Analysis



Iterate to goal



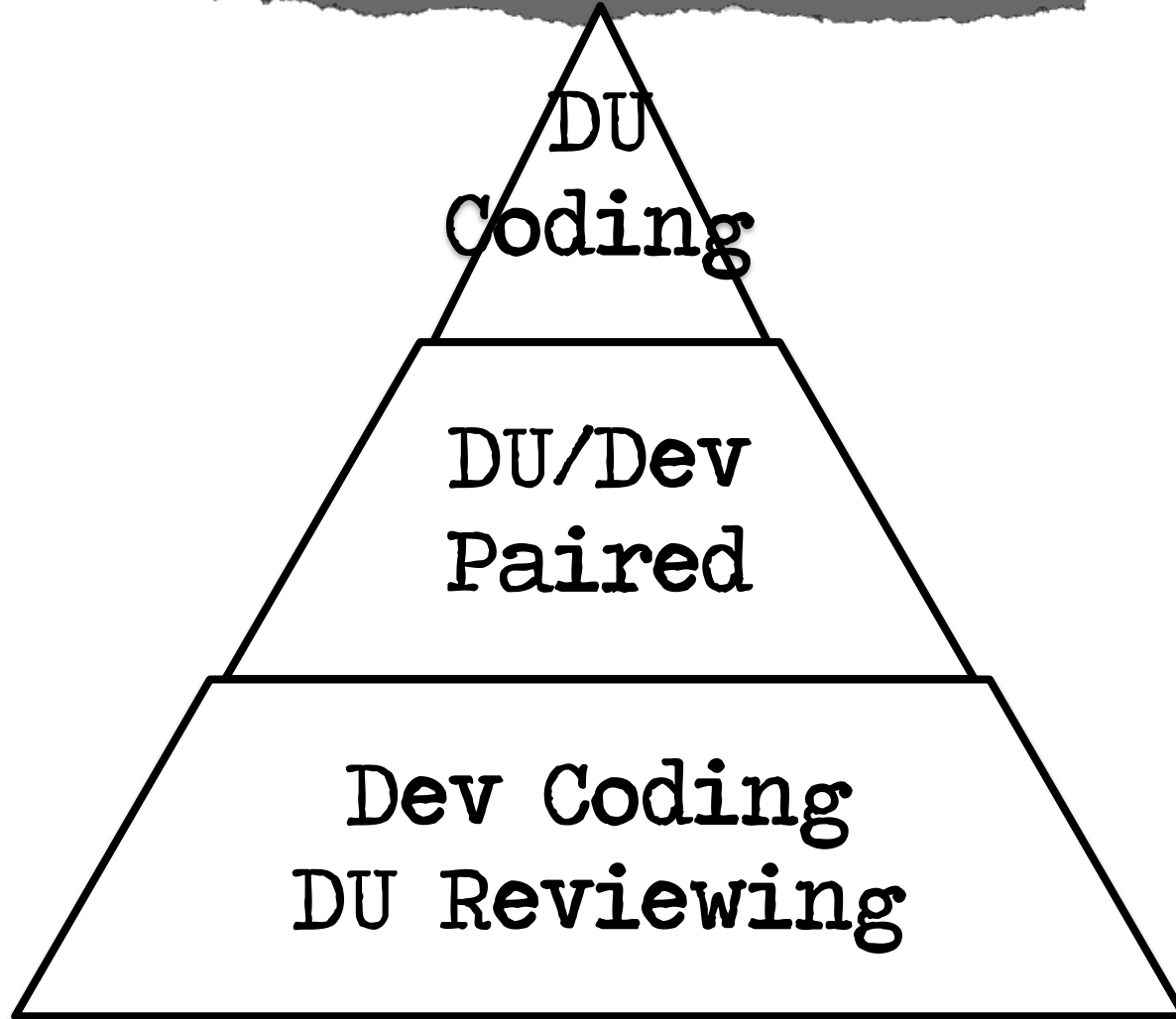
Documentation

Create example-based tutorials!

Domain Folks Programming?

Precision vs.
Algorithmics!

Domain Folks Programming?



DSL as a Product

Release Plan

Bug Tracker

Testing!

Support

Documentation

Reviews

Reviews become
easier --- less
code, more
domain-specific

The End.

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

This material is
based on this book:

<http://dslbook.org>

available Feb 2013



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