DSL Design

A conceptual framework for building good DSLs

Markus Voelter
independent/itemis
voelter@acm.org
http://www.voelter.de
@markusvoelter

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.

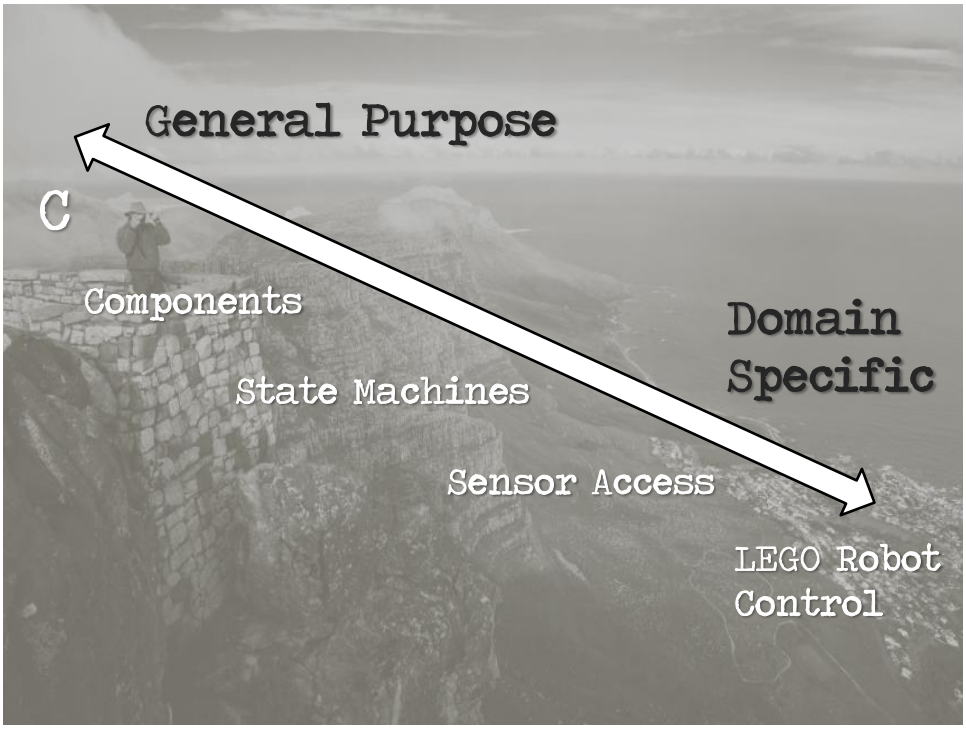
Concepts (abstract syntax)

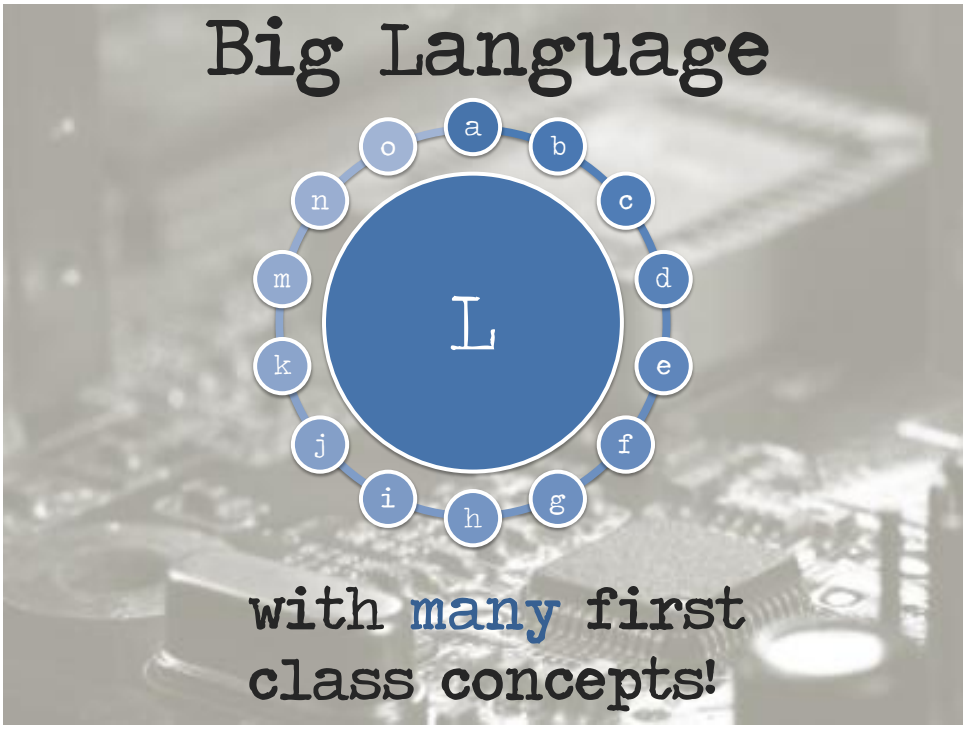
(concrete) Syntax

semantics (generators)

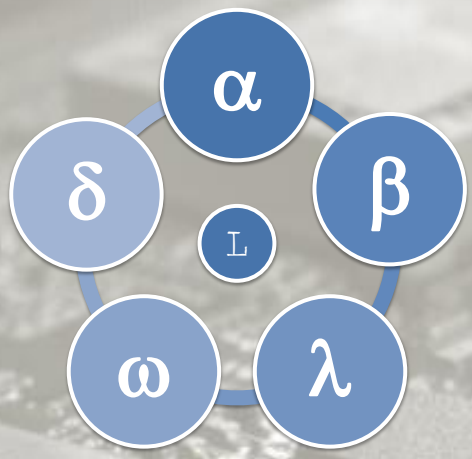
Tools and IDE

	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

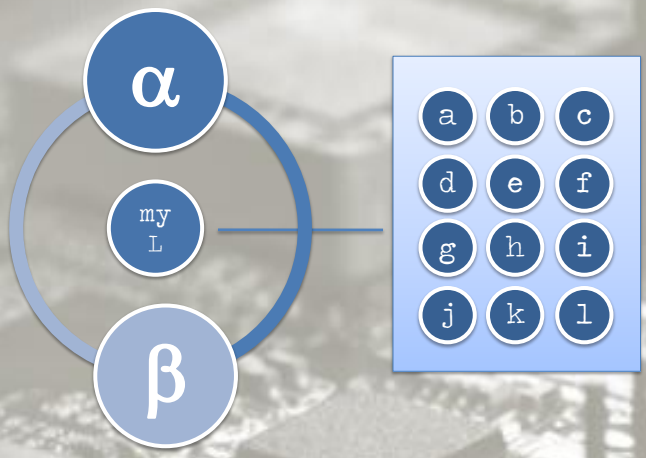

Big Language
L
a
b
c
d
e
f
g
h
i
j
k
m
n
o
with many first class concepts!

Small Language



with a few, orthogonal and poweful concepts

Modular Language



with many optional, composable concepts

Case Studies

Components

```
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 …
  }
}
```

Components

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

Refrigerators

```
appliance KIR {

    compressor compartment cc {
        static compressor c1
        fan ccfan
    }

    ambient tempsensor at

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

}
```

Refrigerators

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

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->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
        }
    }
```

Refrigerators

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

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

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


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

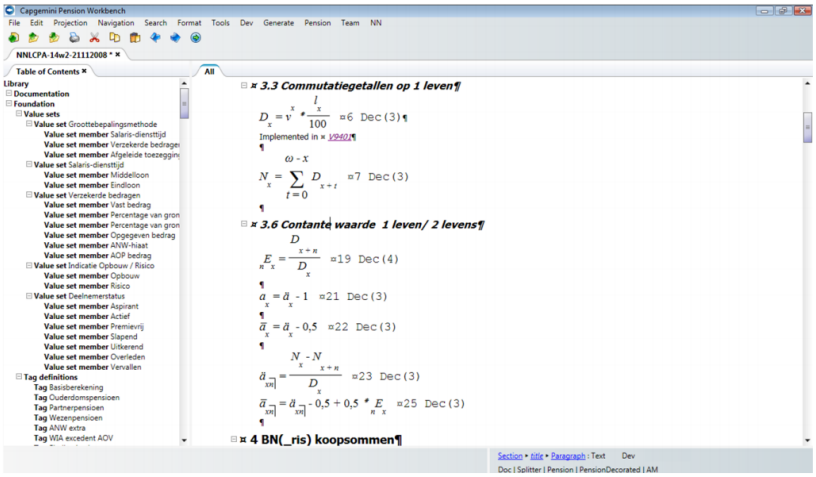
```
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


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

Extended C

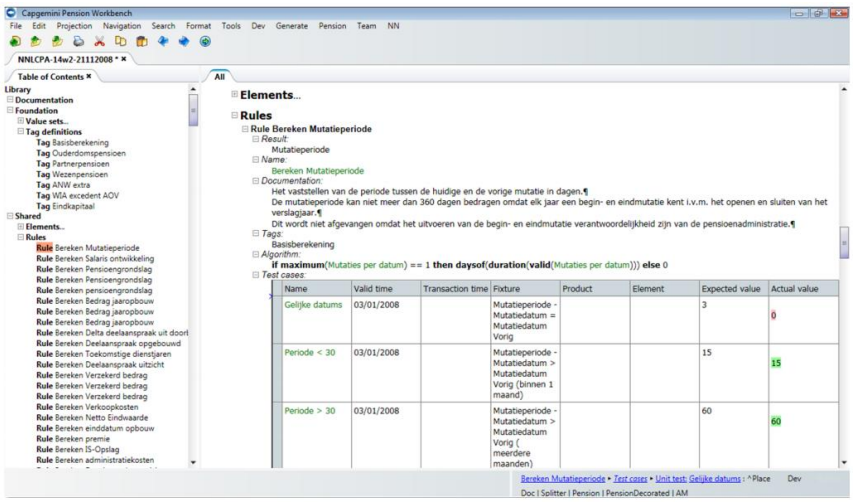
```
module main imports OsekKernel, EcAPI, BitLevelUtilies {

  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);
  }
}
```

Pension Plans



Pension Plans



WebDSL

```
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 }
  }
```

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

www.answers.com/topic/model

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

www.ichnet.org/glossary.htm

Model

an abstraction or simplification of reality

ecosurvey.gmu.edu/glossary.htm

Model

which ones?

an abstraction or simplification of reality

what should we leave out?

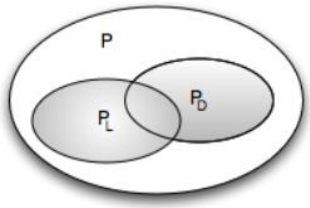
ecosurvey.gmu.edu/glossary.htm

Model Purpose

… code generation
… analysis and checking
… platform independence
… stakeholder integration
…

drives design of language!

Programs
Languages
Domains



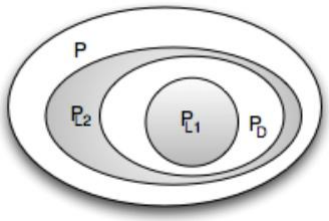
body of knowledge in the real world

deductive top down

Domain

existing software (family)

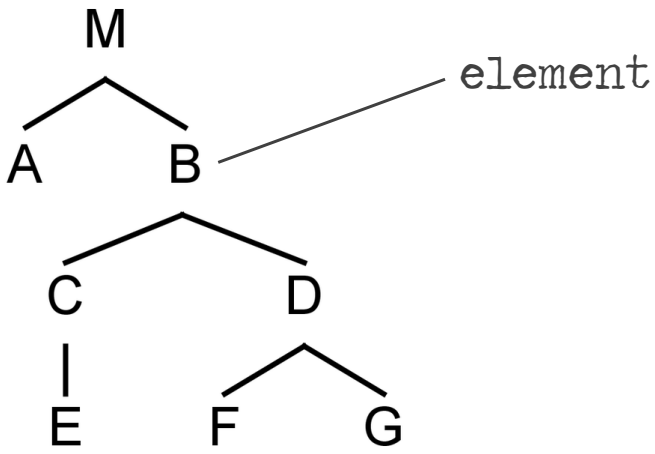
inductive bottom up



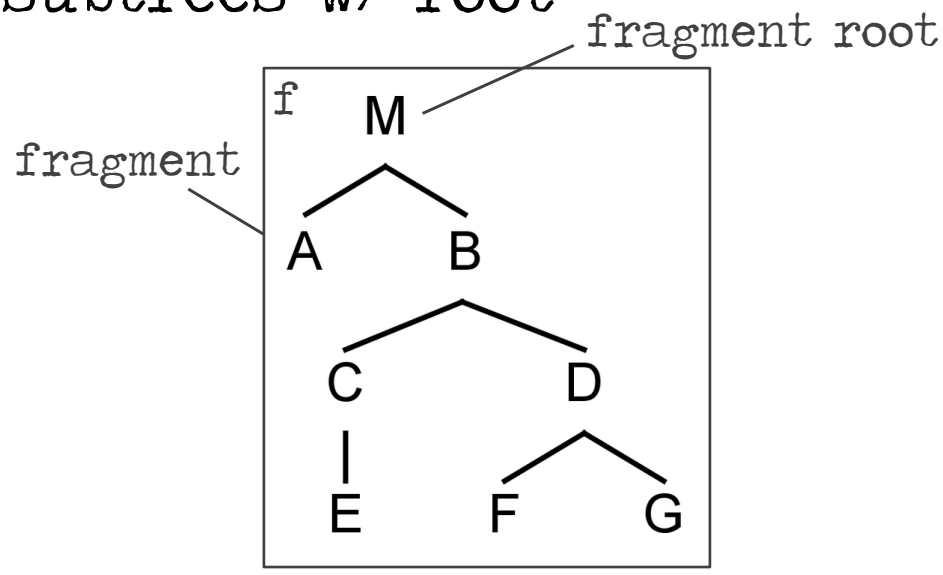
A DSL L_D for D is a language that is **specialized** to encoding P_D programs.

more efficient
smaller

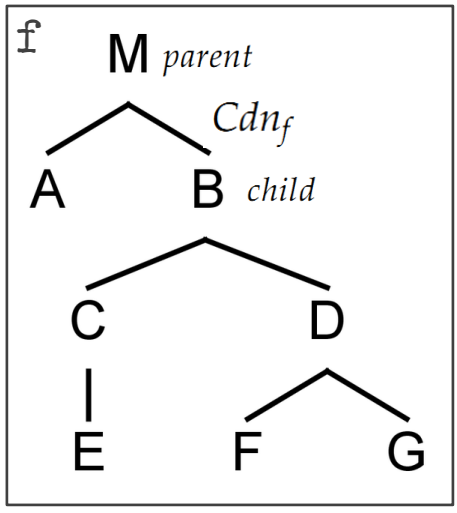
Programs are trees



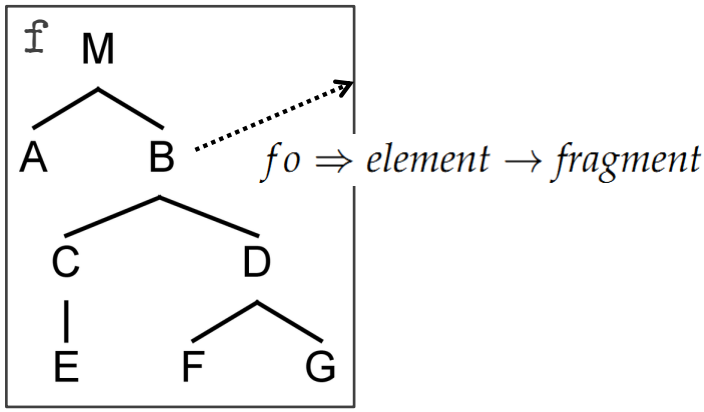
Fragments are subtrees w/ root



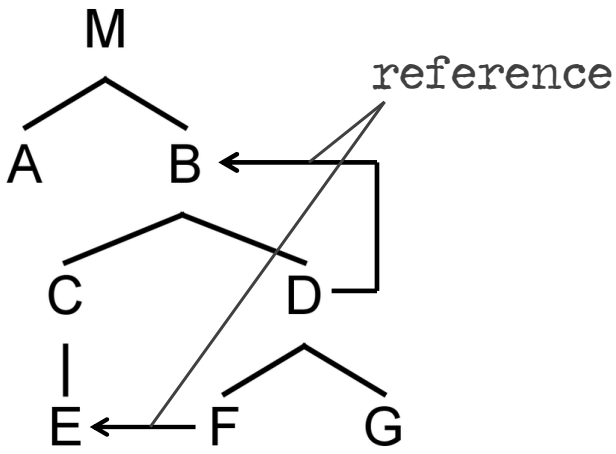
Parent-Child Relation



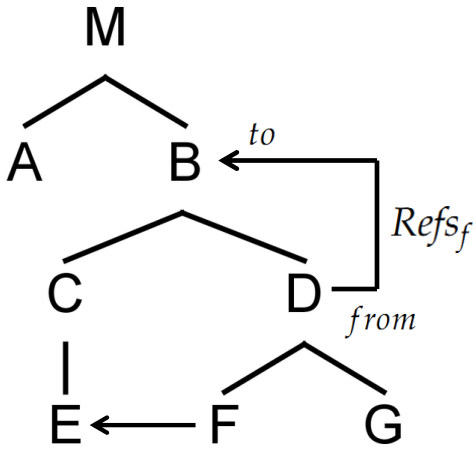
Programs and Fragments



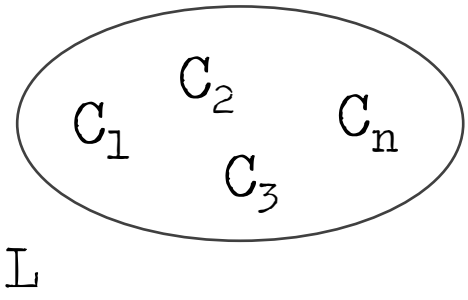
Programs are graphs, really.



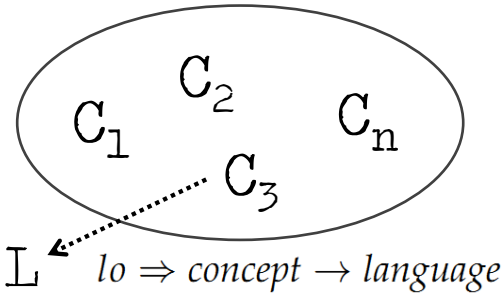
Programs are graphs, really.



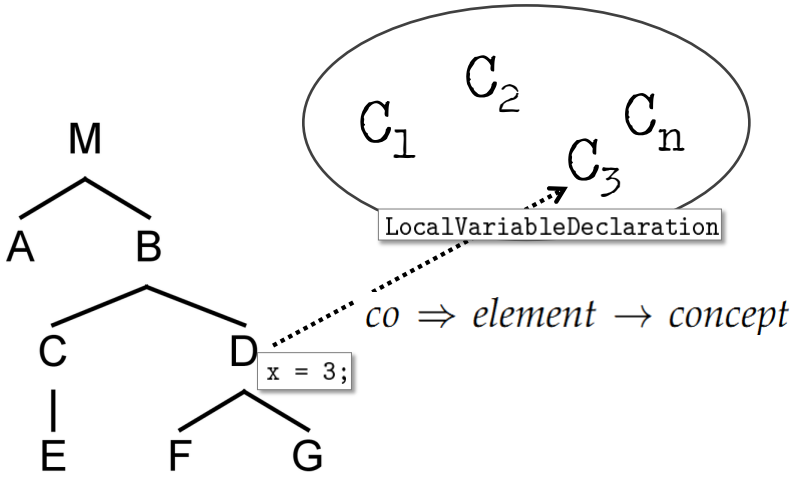
Languages are sets of concepts



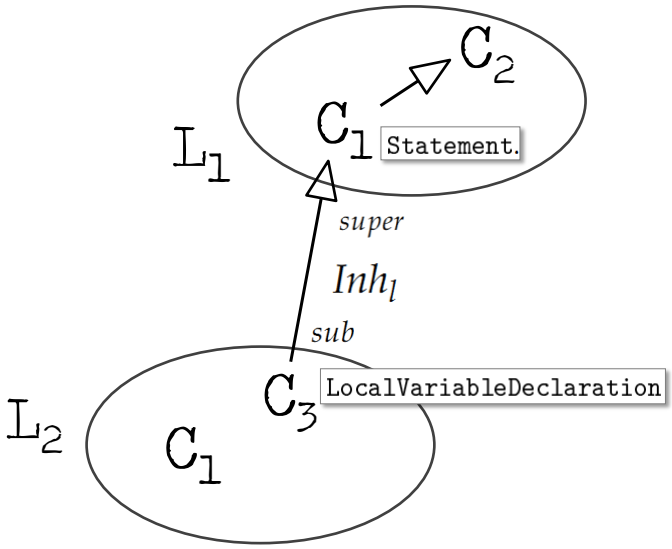
Languages are sets of concepts



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
    }
}
```

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

Heterogeneous

```
module CounterExample from counterd imports nothing {

  var int theI;

  var boolean theB;

  var boolean hasBeenReset;

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

  var Counter c1;

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

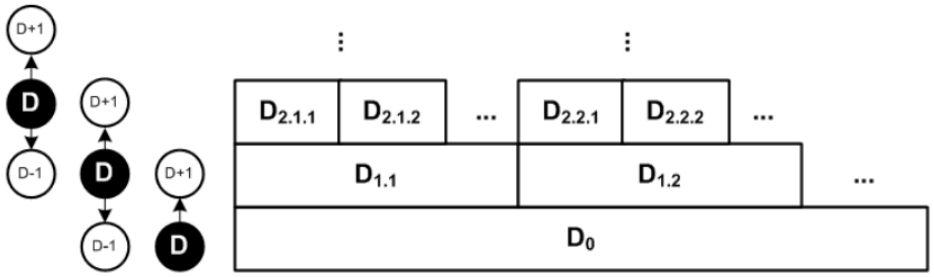
C

Statemachines

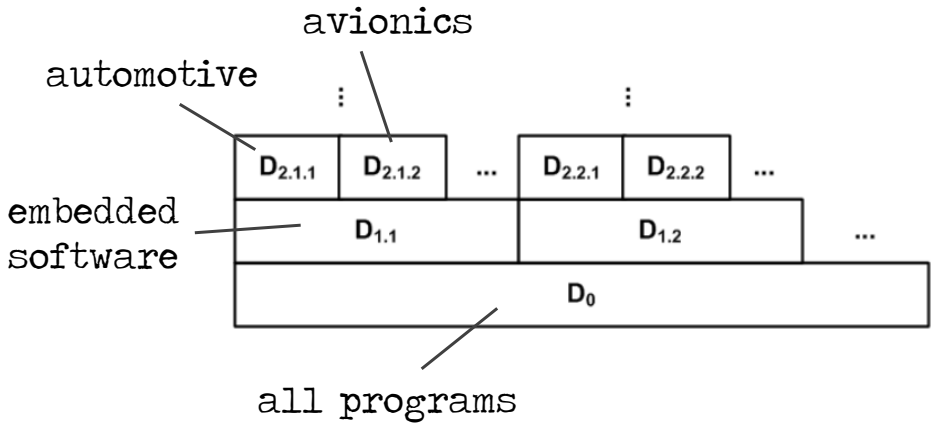
Testing

Heterogeneous

Domain Hierarchy



Domain Hierarchy



Design Dimensions

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

Expressivity

expressivity	completeness
coverage	paradigms
semantics	modularity
separation of concerns	concrete syntax

Shorter Programs

More Accessible Semantics

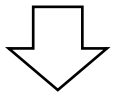
For a limited Domain!

Domain Knowledge encapsulated in language

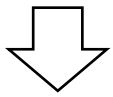
Def: Expressivity

A language L_1 is *more expressive in domain D* than a language L_2 if for each $p \in P_D \cap P_{L_1} \cap P_{L_2}$, $|p_{L_1}| < |p_{L_2}|$

Smaller Domain



More Specialized Language



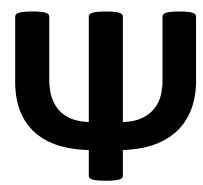
Shorter Programs

The do-what-I-want language

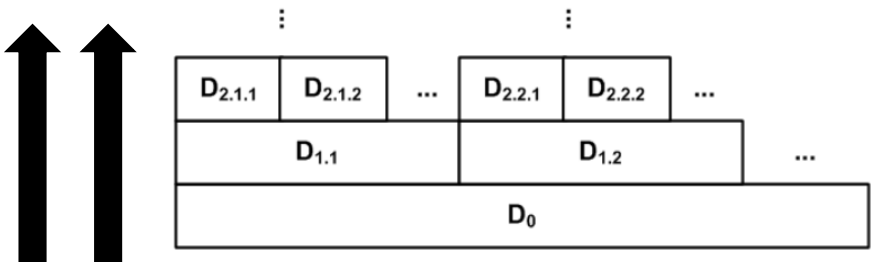
Ψ

Single Program vs. Class/Domain

No Variability!

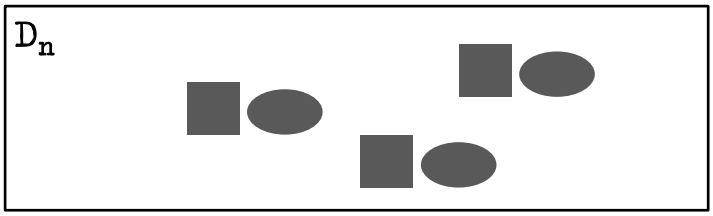


Domain Hierarchy

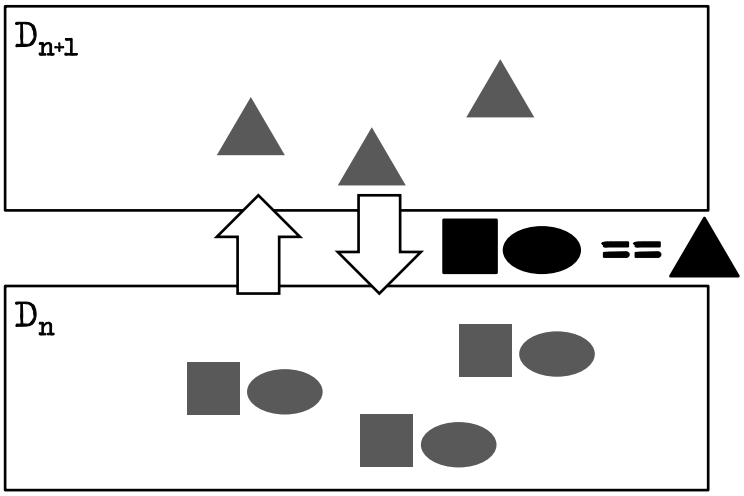


more specialized domains
more specialized languages

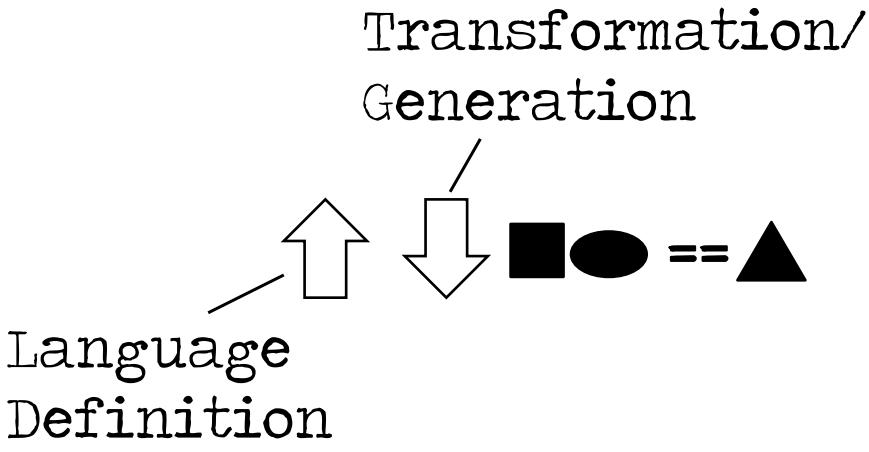
Reification



Reification



Reification




```
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

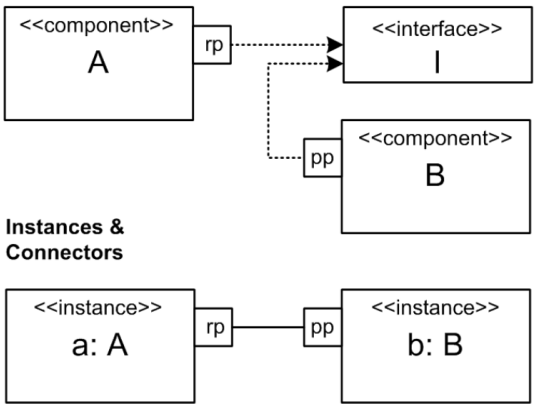
A DSL is a **language** at D that provides **linguistic abstractions** for **common patterns and idioms** of a language at D-1 when used within the domain D.

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.

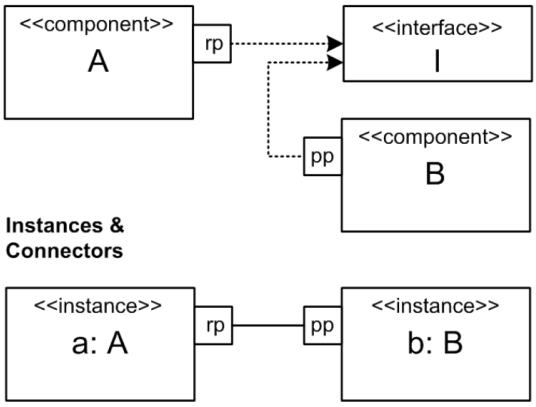
Declarative!

Another Example




```
if (isConnected(port)) {
    port.doSomething();
}
```

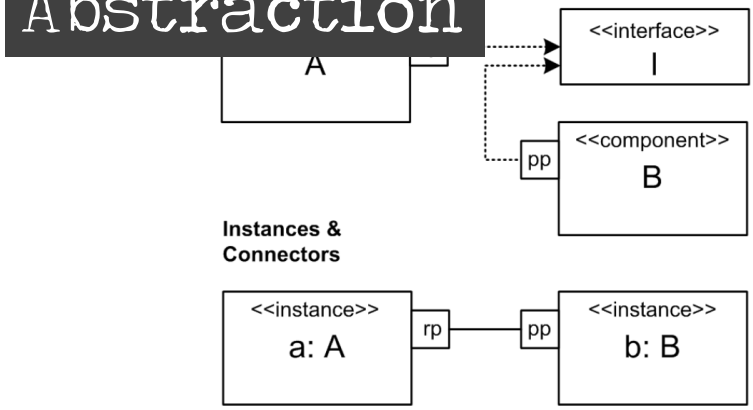
Another Example




```
if (isConnected(port) || true) {
    port.doSomething();
}
```

Turing Complete!
Requires Semantic Analysis!

Linguistic Abstraction

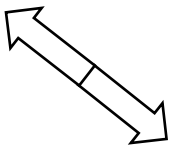



```
with port (port) {
   port.doSomething();
}
```

Linguistic Abstraction

```
exported component AnotherDriver extends Driver {
  ports:
    requires optional ILogger logger
    provides IDriver cmd
  contents:
    field int count = 0

    int setDriverValue(int addr, int value) <- op cmd.setDriverValue {
      with port (logger) {
        logger.log("some error message");
      } with port
      return 0;
    }
}
```

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

}
```

Language Evolution Support

Customization
vs.
Configuration

Precision
vs.
Algorithmics

Coverage

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

Domain D_L defined inductively by L

(the domain that can be expressed by L)

$C_L(L) == 1$ (by definition)

not very interesting!

Def: Coverage

to what extend can a language L cover a domain D

$$C_D(L) = \frac{number\ of\ P_D\ programs\ expressable\ by\ L}{number\ of\ programs\ in\ domain\ D}$$

Def: Coverage

why would $C_D(L)$ be != 1?

1) L is deficient
2) L is intended to cover only a subset of D, corner cases may make L too complex

Rest must be expressed in D_{-1}

Semantics & Execution

expressivity	completeness
coverage	paradigms
semantics	modularity
separation of concerns	concrete syntax

Static Semantics

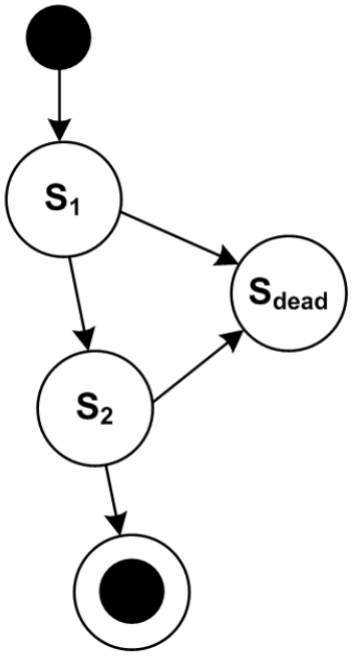
Execution Semantics

Static Semantics

Execution Semantics

Static Semantics

Constraints
Type Systems

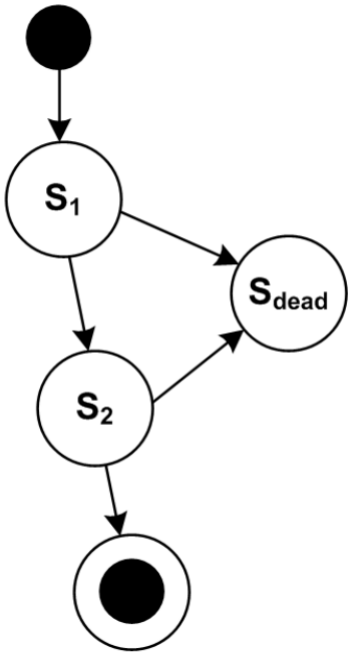


Unique State Names

Unreachable States

Dead End States

…



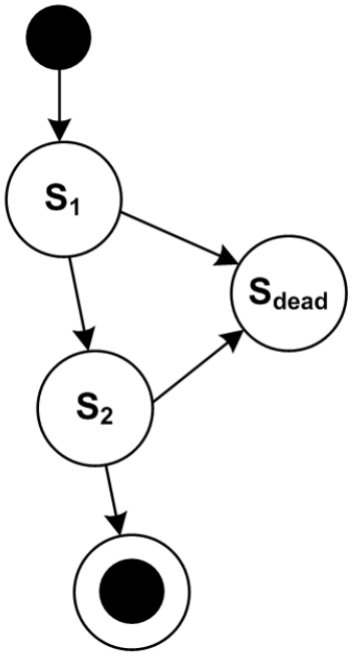
Unique State Names

Unreachable States

Dead End States

…

Easier to do on a declarative Level!



Unique State Names

Unreachable States

Dead End States

…

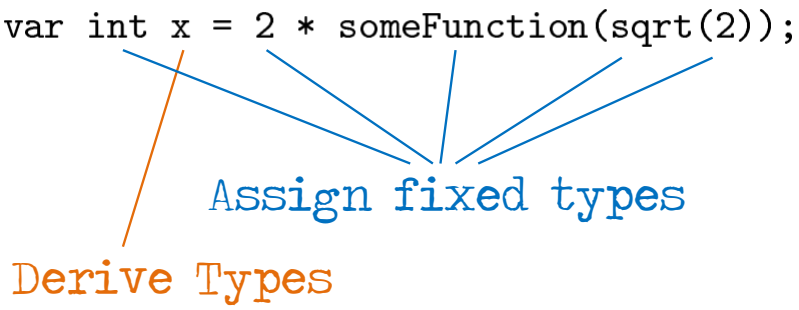
Easier to do on a declarative Level!

Thinking of all constraints is a coverage problem!

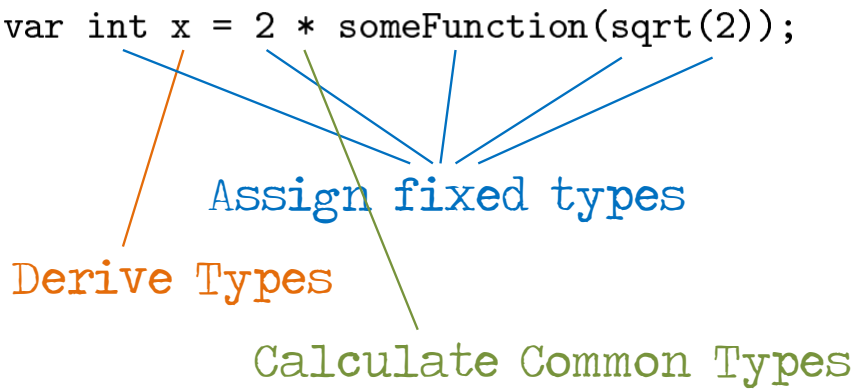
```
var int x = 2 * someFunction(sqrt(2));
```

Assign fixed types

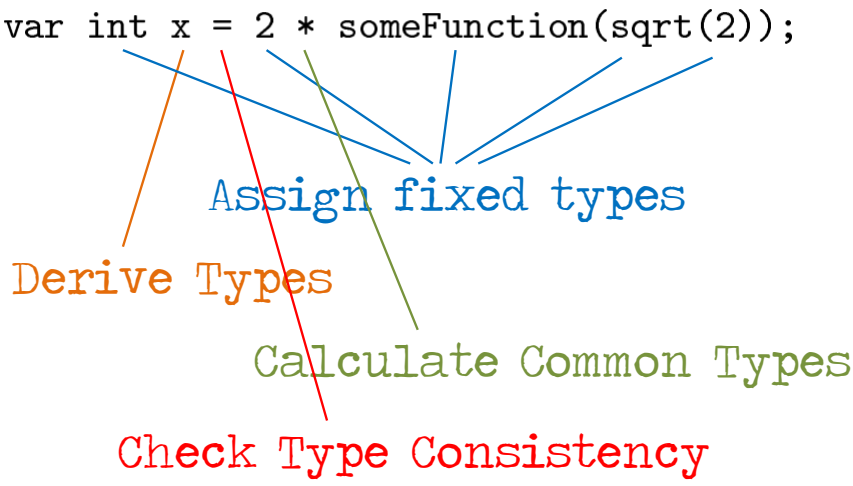
What does a type system do?



What does a type system do?



What does a type system do?



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

What does it all mean?

Execution Semantics

Def: Semantics

… via mapping to lower level

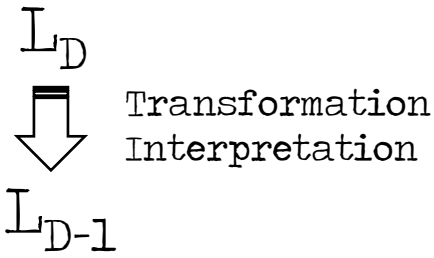
$$semantics(p_{L_D}) := q_{L_{D-1}}$$
$$where\ OB(p_{L_D}) == OB(q_{L_{D-1}})$$

OB: Observable Behaviour (Test Cases)

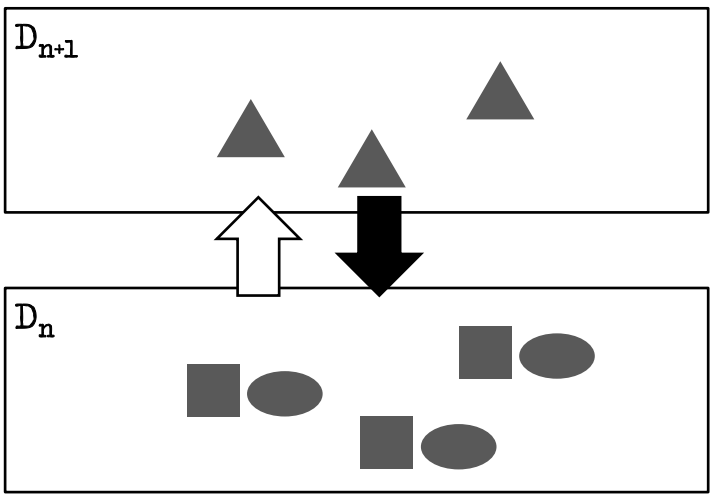
Def: Semantics

… via mapping to lower level

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



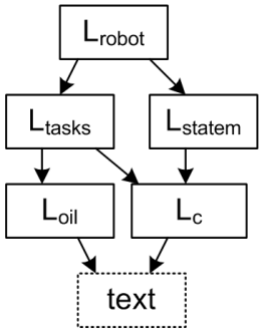
Transformation



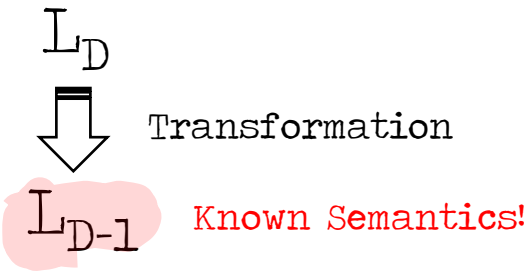
multi-stage

Transformation

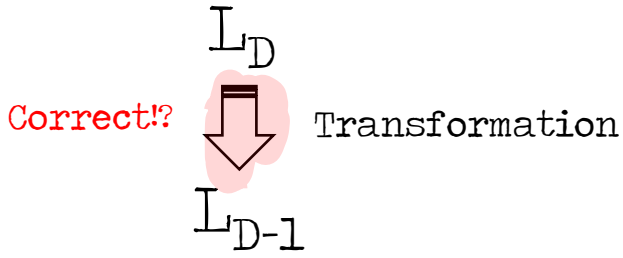
```
module impl imports <<imports>> {

  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

}
```



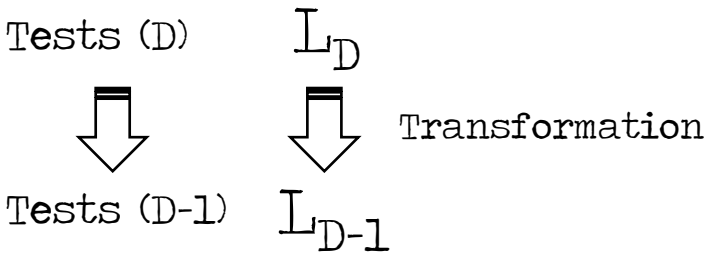
Transformation



Transformation



Transformation

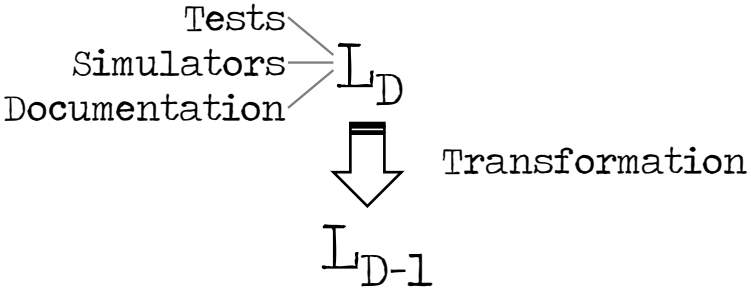


Run tests on both levels; all pass.
Coverage Problem!

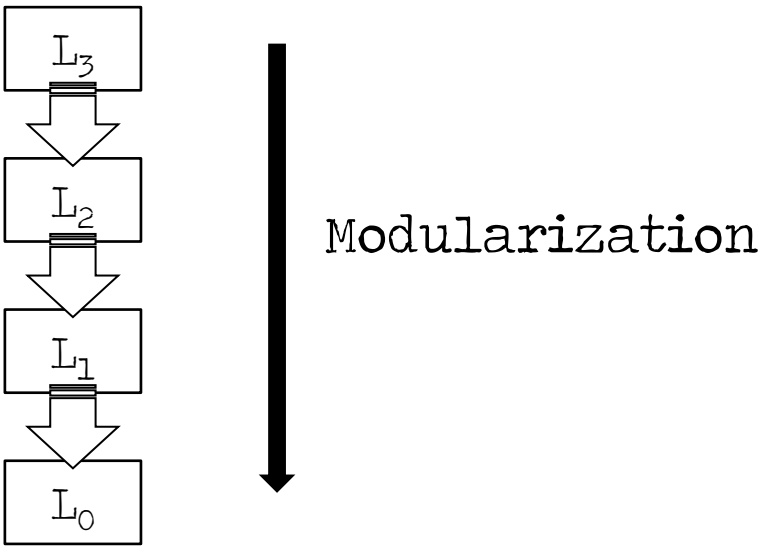
Transformation

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	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

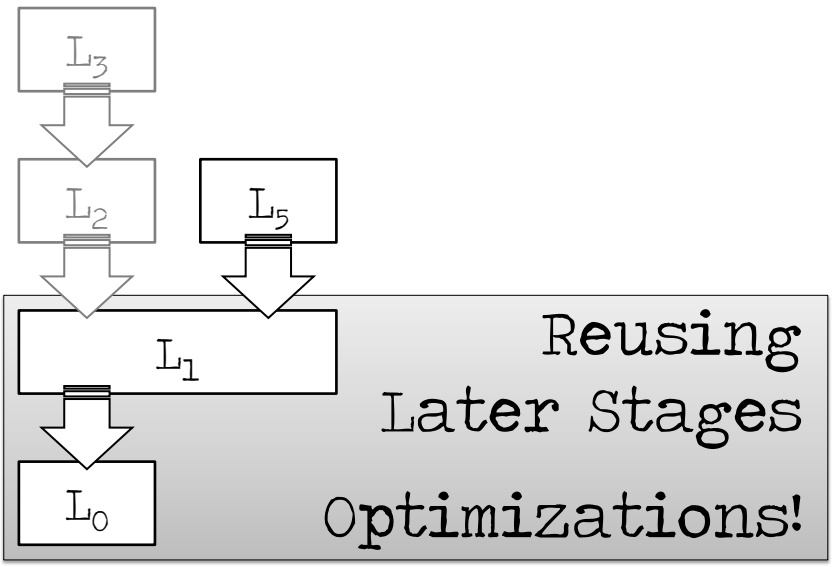
Transformation



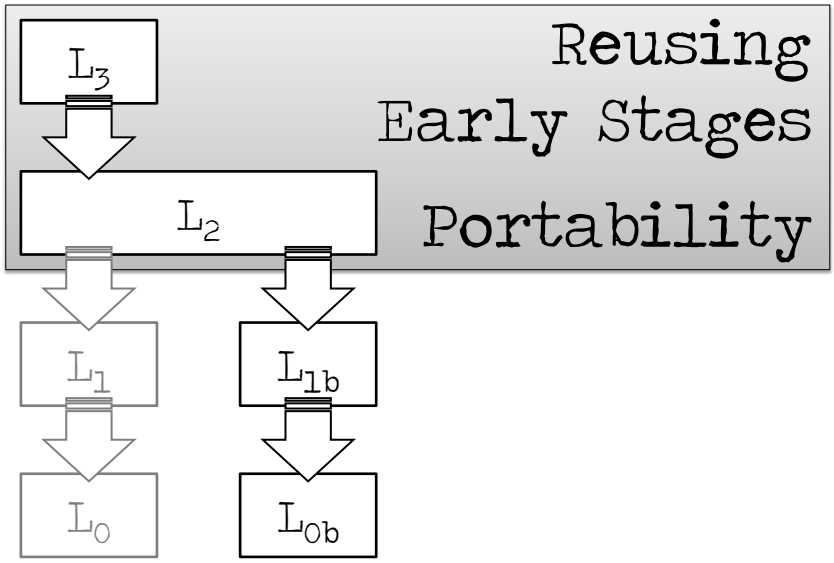
Multi-Stage



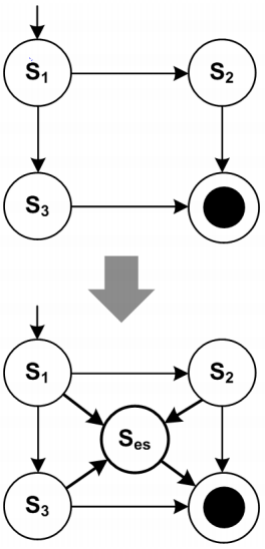
Multi-Stage: Reuse



Multi-Stage: Reuse

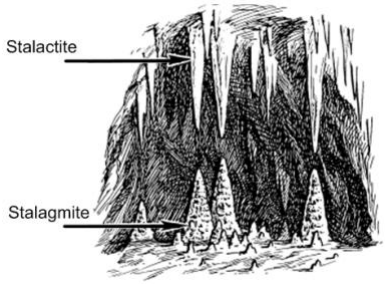


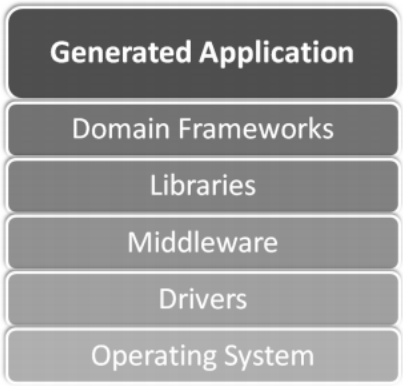
Multi-Stage: Preprocess



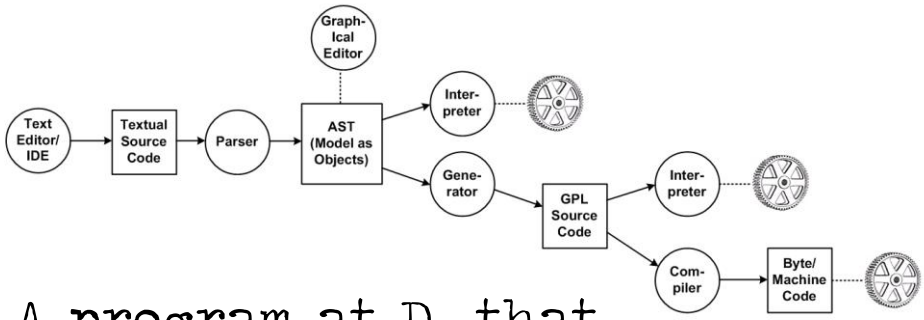
Adding an optional, modular emergency stop feature

Platform





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_0

imperative → step through
functional → eval recursively
declarative→ ? solver ?

Transformation	Interpretation

Transformation

+ Code Inspection

Interpretation

Transformation

+ Code Inspection
+ Debugging

Interpretation

Transformation

- Code Inspection
- Debugging
- Performance & Optimization

Interpretation

Transformation

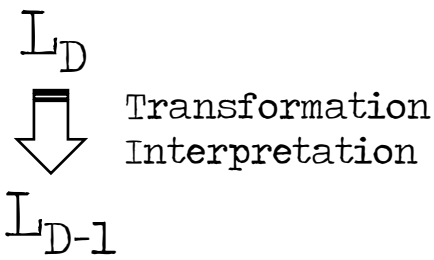
- Code Inspection
- Debugging
- Performance & Optimization
- Platform Conformance

Interpretation

Transformation	Interpretation
+ Code Inspection	
+ Debugging	
+ Performance & Optimization	
+ Platform Conformance	
	+ Turnaround Time
	+ Runtime Change

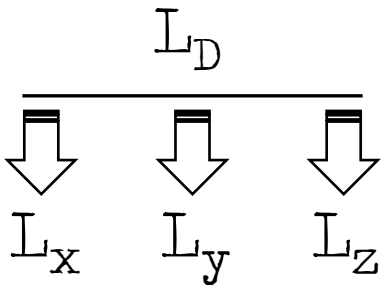
Def: Semantics

… via mapping to lower level



Multiple Mappings

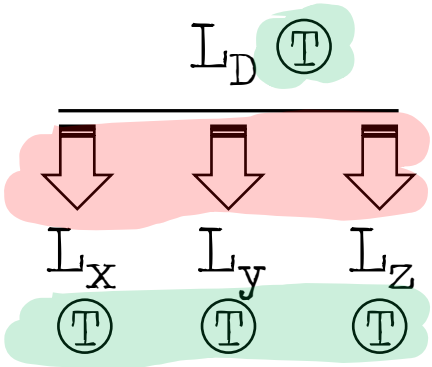
… at the same time



Similar Semantics?

Multiple Mappings

… at the same time

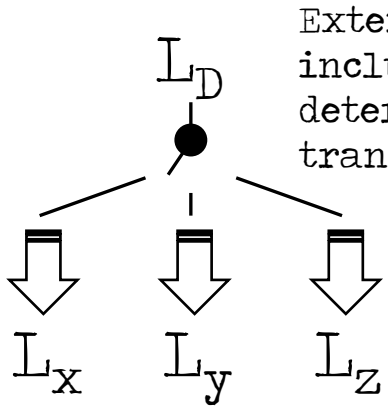


Similar Semantics?

all green!

Multiple Mappings

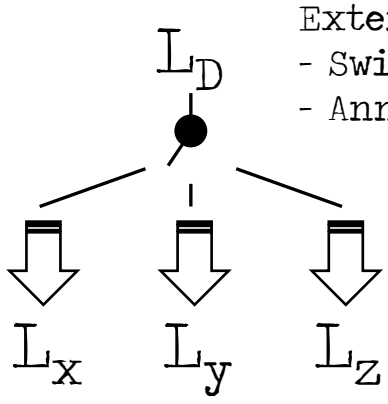
… alternatively, selectably



Extend L_D to include data that determines transformation

Multiple Mappings

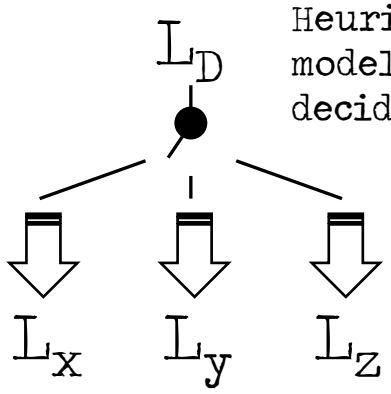
… alternatively, selectably



External Data:
- Switches
- Annotation Model

Multiple Mappings

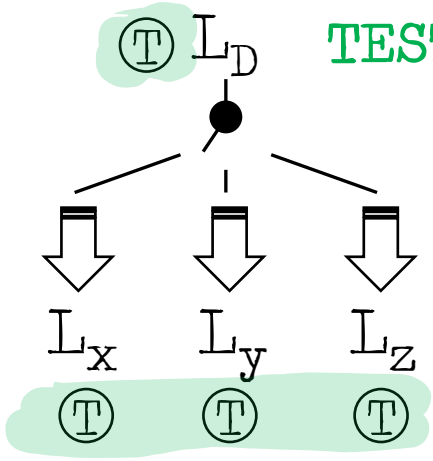
… alternatively, selectably



Heuristics: Analyze model to try to decide

Multiple Mappings

… alternatively, selectably

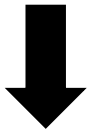


TESTING!

Reduced Expressiveness

bad? maybe.

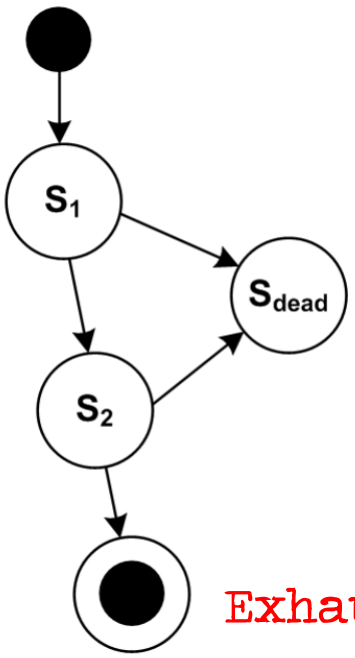
good? maybe!



Model Checking

SAT Solving

Exhaustive Search, Proof!



Unique State Names

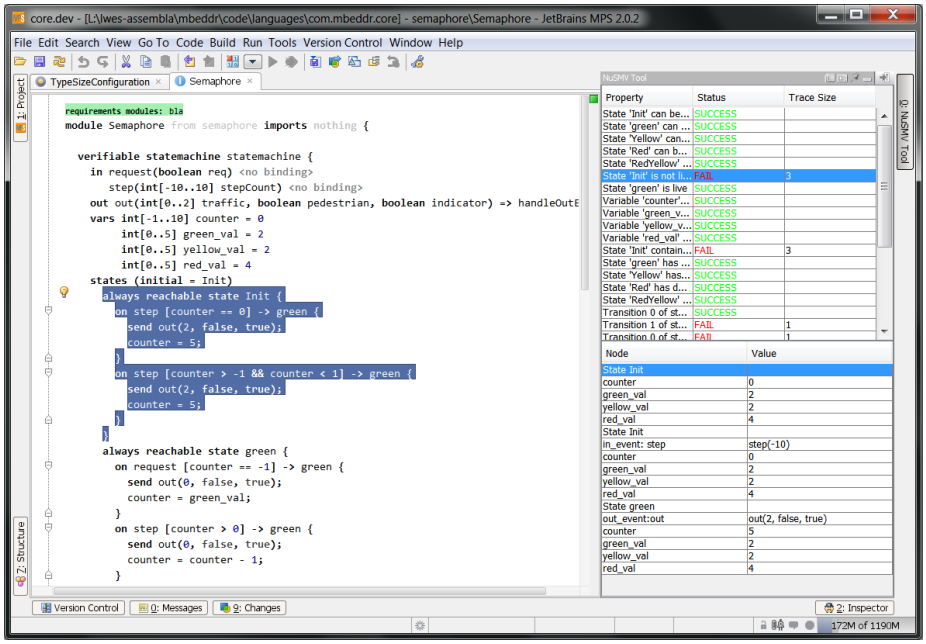
Unreachable States

Dead End States

Guard Decidability

Reachability

Exhaustive Search, Proof!

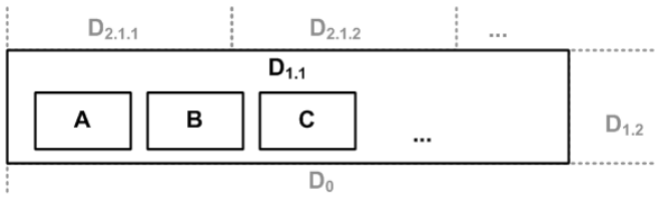

core.dev - [L:\lwes-assembla\mbeddr\code\languages\com.mbeddr.core] - semaphore\Semaphore - JetBrains MPS 2.0.2
File Edit Search View Go To Code Build Run Tools Version Control Window Help
TypeSizeConfiguration
Semaphore
requirements modules: bla
module Semaphore from semaphore imports nothing {
verifiable statemachine statemachine {
in request(boolean req) <no binding>
step(int[-10..10] stepCount) <no binding>
out out(int[0..2] traffic, boolean pedestrian, boolean indicator) => handleOutE
vars int[-1..10] counter = 0
int[0..5] green_val = 2
int[0..5] yellow_val = 2
int[0..5] red_val = 4
states (initial = Init)
always reachable state Init {
on step [counter == 0] -> green {
send out(2, false, true);
counter = 5;
}
on step [counter > -1 && counter < 1] -> green {
send out(2, false, true);
counter = 5;
}
}
always reachable state green {
on request [counter == -1] -> green {
send out(0, false, true);
counter = green_val;
}
on step [counter > 0] -> green {
send out(0, false, true);
counter = counter - 1;
}
NuSMV Tool
Property
Status
Trace Size
State 'Init' can be...
State 'green' can ...
State 'yellow' can...
State 'Red' can b...
State 'RedYellow' ...
State 'Init' is not li...
State 'green' is live
Variable 'counter'...
Variable 'green_v...
Variable 'yellow_v...
Variable 'red_val' ...
State 'Init' contain...
State 'green' has ...
State 'Yellow' has...
State 'Red' has d...
State 'RedYellow' ...
Transition 0 of st...
Transition 1 of st...
Node
Value
State Init
counter
green_val
yellow_val
red_val
State Init
in_event: step
counter
green_val
yellow_val
red_val
State green
out_event:out
counter
green_val
yellow_val
red_val
step(-10)
out(2, false, true)
Version Control
0: Messages
9: Changes
2: Inspector
172M of 1190M

Separation of Concerns

expressivity	completeness
coverage	paradigms
semantics	modularity
separation of concerns	concrete syntax

Several Concerns

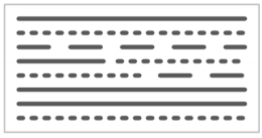
… in one domain



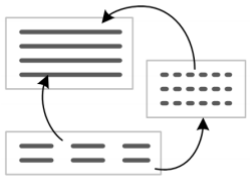
Several Concerns

… in one domain

integrated into one fragment



separated into several fragments

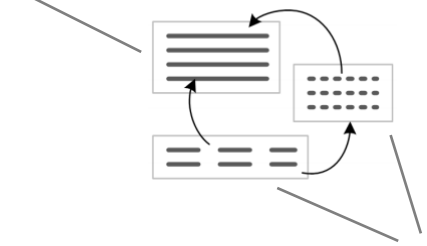


Viewpoints

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

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

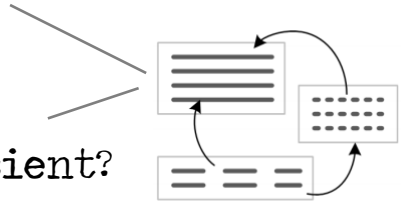
independent



dependent

Viewpoints

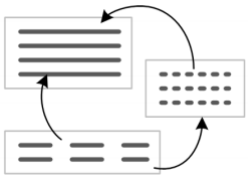
independent



sufficient?

contains all the data for running a meaningful transformation

Viewpoints



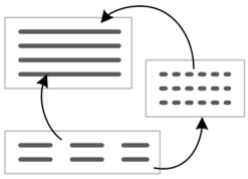
Well-defined Dependencies

No Cycles!

Avoid Synchronization!

(unless you use a projectional editor)

Viewpoints: Why?

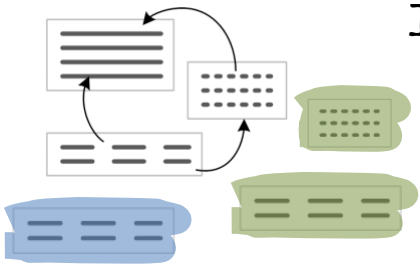


Sufficiency

Different Stakeholders

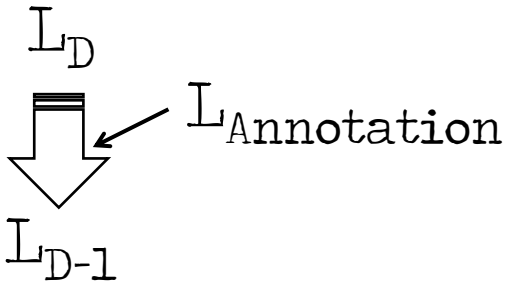
Different Steps in Process - VCS unit!

Viewpoints: Why?

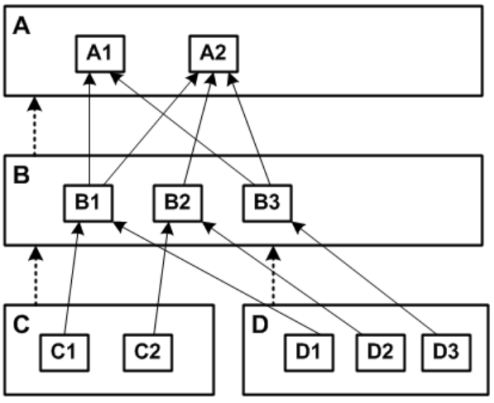


1:n Relationships

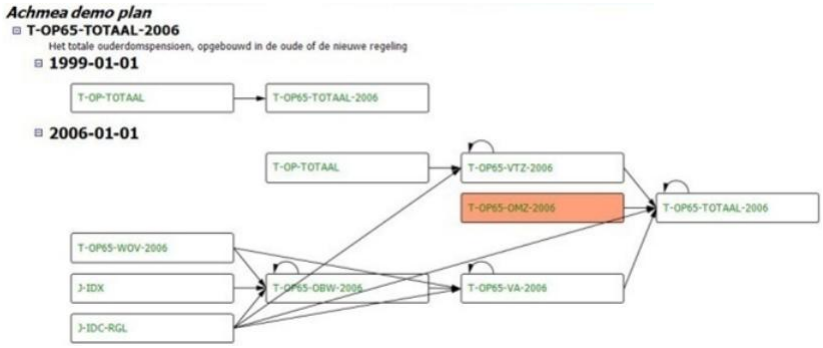
Viewpoints: Why?



Progressive Refinement



Views on Programs



Completeness

expressivity	completeness
coverage	paradigms
semantics	modularity
separation of concerns	concrete syntax