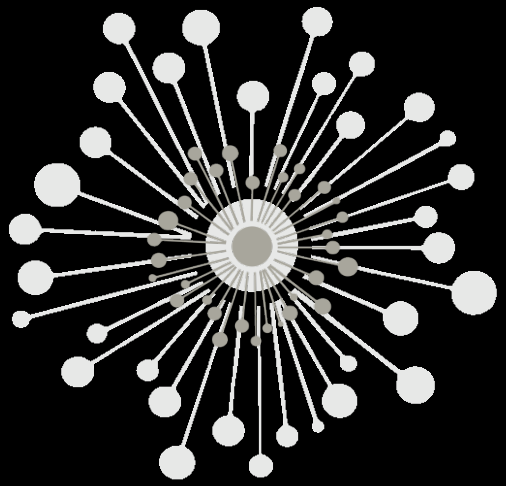


Safer Software

through Better Abstractions and Static Analysis

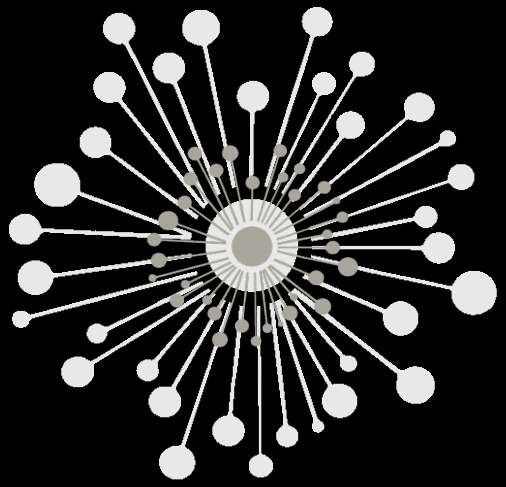
Markus Völter
voelter@acm.org
www.voelter.de
@markusvoelter

Bernd Kolb
kolb@itemis.de
www.itemis.de
@berndkolb

voelter { ingenieurbüro für softwaretechnologie // itemis

Safer Software through Better Abstractions and Static Analysis

1



Introduction

Security

Safety

Robustness

Process

Education
Requirements
Reviews
Pen Testing

Architecture

Authentication
Encryption
DMZs
Runtime Monitoring

Implementation

Implementation

Does the system work properly?

Buffer Overruns, NPEs, Dead-Ends, Unhandled Cases/Values

Does it do the right things?

Are the behaviours expressed in requirements implemented faithfully? Are all use cases supported?

Does it only do the right things?

Additional, unspecified behaviors are often loopholes and can be exploited maliciously.

Correct-by-Construction

The language/framework/ API/modeling tool doesn't let devs make a particular class of mistakes.

Analysis-and-Fix

You analyze the code/model after the fact and try to find problems which devs then fix.

Correct-by-Construction

Languages

Analysis-and-Fix

Analysis Tools

Languages

Analysis Tools

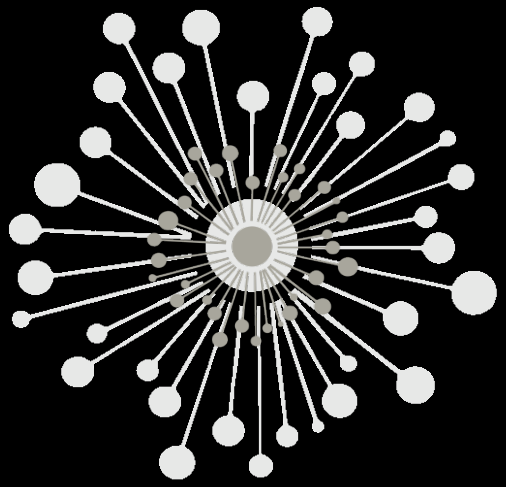
Agenda

Domain-Specificity, Ability to detect interesting problems, (Development Effort)

- Analyze/Understand (domain-specific)
- Extend Language (domain-specific)
- Extend Language (compensate crap)
- Advanced Analyses
- High Q Testing
- Lint & Co
- Use a reasonable language

Safer Software through Better Abstractions and Static Analysis

2



Verification: Correct by Construction

and very simple checks

BK

Additional Constraints

Prevent the use of insecure functions

Detect insecure C functions:

```
char* strcpy ( char* dest, const char* src);
```

Mark vetted secure functions as *SECURE API* and only use those.

Check this with tool.

Same Syntax, Different Semantics

Timing Side-Channel Attacks

Guess based on execution time

```
for (i = 0; i < length; ++i) {
     if password[i] != input[i]
          send("bad password");
}
```

Ensure constant or random time

```
/* Returns 1 if a and b are equal, in constant time */
int s2n_constant_time_equals(string, string, len);
```

Amazon‘s s2n library

Same Syntax, Different Semantics

Timing Side-Channel Attacks

```
/* Returns 1 if a and b are equal, in constant time */
int s2n_constant_time_equals(string, string, len);
```

Amazon‘s s2n library

Naming conventions not checkable.

Semantic Annotations better.

Then insert **busy wait** at end (to randomize timing) through preprocessor.

Same Syntax, Different Semantics

Sniffing sensitive Data

```
char* encryptData(char* k_enc, char* data) {
    char k_clr[256];
    decryptKey(k_enc, k_clr);
    char* encryptedData = // encrypt with k_clr
    return encryptedData;
}
```

When leaving the function, **k_clr** still on the stack – only stack pointer moved.

Better: As variables leave scope, wipe their memory. Use Generator.

Apple‘s Got Fail Bug

Apple‘s Intention:

```
if (validateStep1(data, ...) != 0) goto fail;
if (validateStep2(data, ...) != 0) goto fail;
if (validateStep3(data, ...) != 0) goto fail;
fail: handleFailedValidation(data, errorcode, ...);
```

Actual Code

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
       ... other checks ...
fail:
        ... buffer frees (cleanups) ...
        return err;
```

Language Extension

Apple‘s Got Fail Bug

Apple‘s Intention:

```
if (validateStep1(data, ...) != 0) goto fail;
if (validateStep2(data, ...) != 0) goto fail;
if (validateStep3(data, ...) != 0) goto fail;
fail: handleFailedValidation(data, errorcode, ...);
```

More robust notation:

```
trysequentially {
  validateStep1(data, ...);
  validateStep2(data, ...);
  validateStep3(data, ...);
} on fail (errorcode) {
  handleFailedValidation(data, errorcode, ...);
}
```

Aren‘t Macros enough?

```
#define GUARD(x) if ((x)<0) return -1
```

Amazon‘s s2n library

Danger:

```
GUARD( do_something ) + 1;
```

Macros have no protection against wrong use; Poor man‘s LE. Real LE is more robust/safer.

State Machines (in C)

Typically implemented via `switch`, function pointer tables or macro-based libraries.

This is error-prone:

- **Hard to read and understand**
- **Hard to analyze b/c of „hidden“ semantics**
- **Easy to forget a `break;`**
- **Hierarchical Machines a syntactic nightmare**

Language Extension

State Machines in C

```
exported statemachine HierarchicalFlightAnalyzer initial = beforeFlight {
  macro stopped(next) = tp->speed == 0 mps
  macro onTheGround(next) = tp->alt == 0 m
  in event next(Trackpoint* tp) <no binding>
  in event reset() <no binding>
  out event crashNotification() => raiseAlarm
  readable var int16 points = 0
  state beforeFlight {
    entry { points = 0; }
    on next [tp->alt > 0 m] -> airborne
    exit { points += TAKEOFF; }
  } state beforeFlight
  composite state airborne initial = flying {
    on reset [ ] -> beforeFlight { points = 0; }
    on next [onTheGround && stopped] -> crashed
    state flying (airborne.flying) {
      on next [onTheGround && tp->speed > 0 mps] -> landing
      on next [tp->speed > 200 mps] -> flying { points += VERY_HIGH_SPEED; }
      on next [tp->speed > 100 mps] -> flying { points += HIGH_SPEED; }
    } state flying
    state landing (airborne.landing) {
      on next [stopped] -> landed
      on next [ ] -> landing { points--; }
    } state landing
    state landed (airborne.landed) {
      entry { points += LANDING; }
    } state landed
  } state airborne
  state crashed {
    entry { send crashNotification(); }
  } state crashed
}
```

First-class notation does not have those drawbacks.

Language Extension

State Machines in C

```
exported statemachine FlightAnalyzer initial = beforeFlight {
```

		Events	
		next(Trackpoint* tp)	reset()
States	beforeFlight	[tp->alt > 0 m] -> airborne	
	airborne	[tp->alt == 0 m && tp->speed == 0 mps] -> crashed [tp->alt == 0 m && tp->speed > 0 mps] -> landing [tp->speed > 200 mps && tp->alt == 0 m] -> airborne { points += VERY_HIGH_SPEED; } [tp->speed > 100 mps && tp->speed <= 200 mps && tp->alt == 0 m] -> airborne { points += HIGH_SPEED; }	[ ] -> beforeFlight
	landing	[tp->speed == 0 mps] -> landed [tp->speed > 0 mps] -> landing { points--; }	[ ] -> beforeFlight
	landed		[ ] -> beforeFlight
	crashed		
	NextRound <j>		

```
}
```

Various Table-based notations or even diagrams can make the situation even better.

A set of C extensions optimized for embedded software dev.

Research Project at itemis and Fortiss between 2011 and 2013. Since then open sourced, used in projects.

DLT645ProtocolImpl

```
Events
dataReceived(uint8 data)
States
idle        [data == DLT645_FRONT_LEADING_BYTE] -> awaking
Error: type boolean is not a subtype of uint8
awaking     [data == DLT645_FRAME_STARTFLAG] -> recvFrame
  {
    send startFrame(idx, data);
    idx++;
  }
recvFrame   [idx < length - 1 && idx != DLT645_HEADER_DATALENGTH_OFFSET]
  idx ^DLT645ProtocolImpl.DLT645ProtocolImpl.DLT645RxFrameParser.DLT645RxFrameParser.idx (StatemachineVariableDeclaration)
  rxQueueHeadIdx        ^DLT645ProtocolImpl.DLT645ProtocolImpl.rxQueueHeadIdx (Field)
  rxQueueTailIdx        ^DLT645ProtocolImpl.DLT645ProtocolImpl.rxQueueTailIdx (Field)
  idx++;
}
[idx == DLT645_HEADER_DATALENGTH_OFFSET] -> rec
  {
```

UnitDeclarations

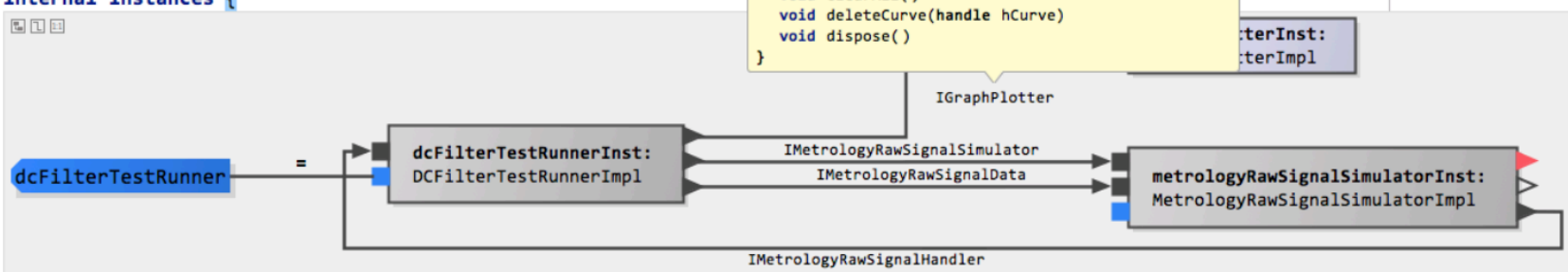
```
section AngleUnits {
  exported unit ° := <no spec> for Angle
  exported unit rad := <no spec> for Angle
  exported unit rotation := <no spec> for Angle

  exported #constant FULL_ROTATION = 2 * M_PI rad;

  exported eager conversion ° -> rad {
   val as <no type> -> val/°/ * M_PI rad / 180 °
}
ger conversion rad -> rotation {
 type> -> val/rad/ * 1 rotation / FULL_RO
```

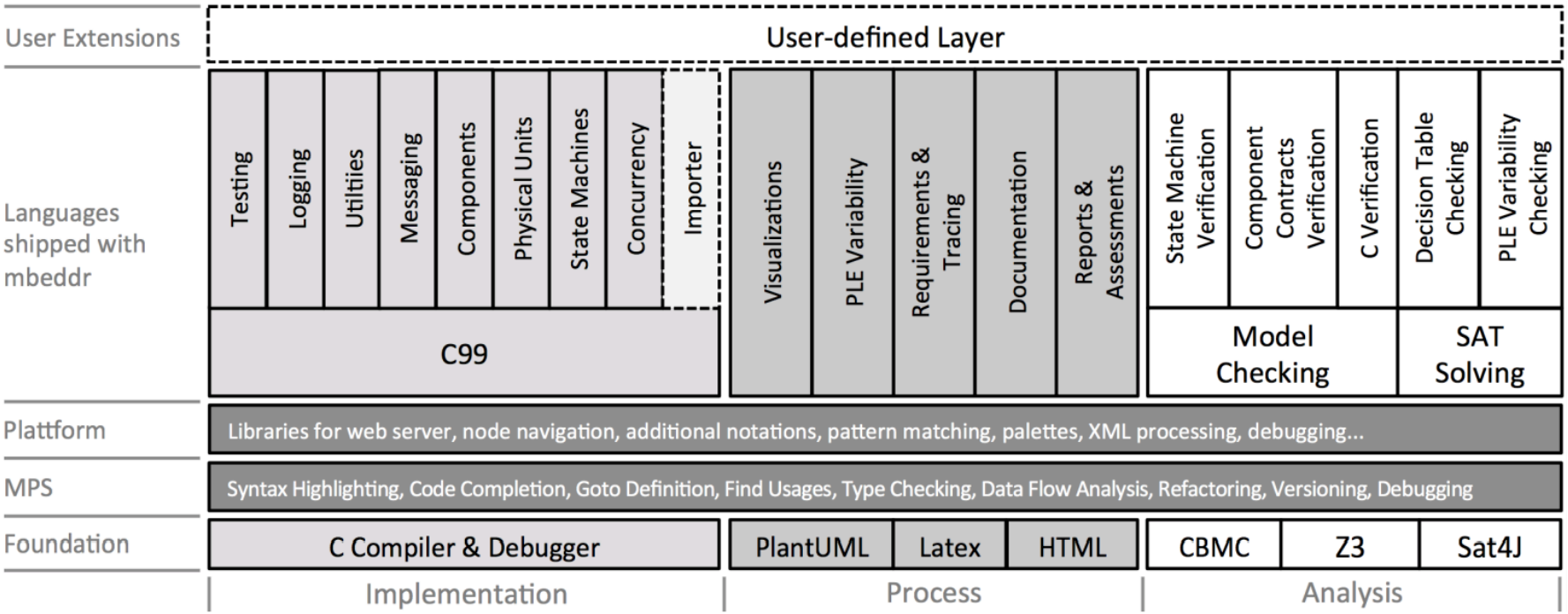
```
exported cs interface IGraphPlotter {
  handle newCurve(Color color, string const label)
  void addPoint(handle hCurve, double x, double y)
  void plot(string const title, handle hCurve)
  void plotAll(string const title)
  void clear(handle hCurve)
  void clearAll()
  void deleteCurve(handle hCurve)
  void dispose()
}
```

DCFilterTestHarness



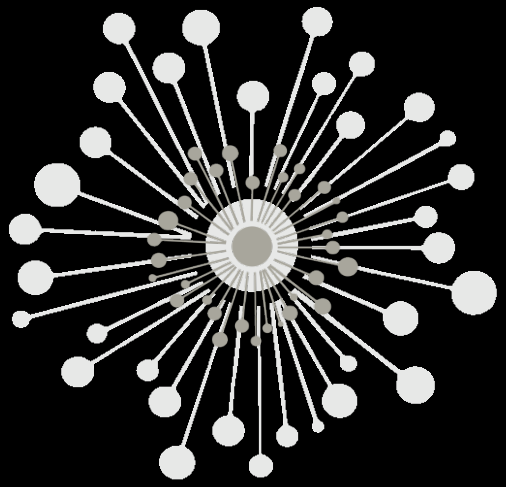
A set of C extensions optimized for embedded software dev.

Many different extensions for embedded programming, program analysis and verification as well as process aspects. User-defined extensions seamlessly supported.



Safer Software through Better Abstractions and Static Analysis

3



Verification: Type Systems

BK

Advanced Type Checking

Preventing Overflow

```
// integer type, unlimited range
number                  => number[-inf|inf]{0}
// positive integer
number[0|inf]           => number[0|inf]{0}
// integer type, range as specified
number[10|20]          => number[10|20]{0}
// decimal type with 2 decimal places, unlimited range
number{2}              => number[-inf|inf]{2}
// range as specified, precision derived from range decimals
number[3.3|4.5]        => number[3.3|4.5]{1}
```

```
type temperature: number[36|42]{1}
type measuredTemp: number[35|43]{2}

val T_measured: measuredTemp = 42.22
val T_calibrated: temperature = T_measured * 0.93
```

Error: type number[32.55|39.99]{4} is not a subtype of number[36|42]{1}

Advanced Type Checking

Preventing Script Injection

```
fun getData(url: string): string<tainted> { val v1 = taint[url] }
fun storeInDB(data: string<!tainted>): boolean { true }

fun createBlogPost(postID: number) {
  val inferred = getData("my/input/form")
  val explicit: string<tainted> = getData("my/input/form")
  val invalid: string<!tainted> = getData("my/input/form")

  // This does not work be
  val v2 = storeInDB(inferred)

  // tainted always wins
  val dirty: string<tainted> = sanitize[inferred] + "Hello"

  // works after sanitizing.
  val stored: string<!tainted> = sanitize[inferred]
  val success = storeInDB(stored)
}
```

Error: type string<tainted> is not a subtype of string<!tainted>

Advanced Type Checking

Segregation of Different Confidentialities

```
val somethingUnclassified: string = "hello"
val somethingConfidential: string<confidential> = "hello"
val somethingSecret: string<secret> = "hello"
val somethingTopSecret: string<topsecret> = "hello"

fun publish(data: string) = data
val p1 = publish(somethingUnclassified)
val p2 = publish(somethingConfidential)
val p3 = publish(somethingSecret)
val p4 = publish(somethingTopSecret)

fun putIntoCIAArchive(data: string<confidential+>) = data
val a2 = putIntoCIAArchive(somethingConfidential)
val a1 = putIntoCIAArchive(somethingUnclassified)
val a3 = putIntoCIAArchive(somethingTopSecret)
val a4 = putIntoCIAArchive(somethingSecret)

fun putIntoCIAArchive(data: string<confidential+>) = data
val a2 = putIntoCIAArchive(somethingConfidential)
val a1 = putIntoCIAArchive(somethingUnclassified)
val a3 = putIntoCIAArchive(somethingTopSecret)
val a4 = putIntoCIAArchive(somethingSecret)

fun tellANavyGeneral(data: string<secret->) = data
val g1 = tellANavyGeneral(somethingConfidential)
val g2 = tellANavyGeneral(somethingSecret)
val g3 = tellANavyGeneral(somethingTopSecret)
val g4 = tellANavyGeneral(somethingUnclassified)
```

Error: type string is not a subtype of string<confidential+>

Safer Software through Better Abstractions and Static Analysis

4

Verification: SMT Solving

MV

What Solvers Do

For a set of equations $e_i \subset E$ with free variables v_i, is there an assignment of values to all v_i such at all e_i are true?

No => UNSAT
Yes => Value Assignments („Model“)

E: $2 * x == 3 * y$ **Model:** $x = 3, y = 2.$

$x + 2 == 5$

Solvers can do this quickly for large sets of equations with many free variables.

Many „Theories“: Booleans, Integers, Reals, Collections, ...

Example: Decision Table

	mode == *MANUAL*	mode == *AUTO*
speed < 30	false	false
speed > 30	false	true
speed > 40	true	true

Overlap-Freedom:

for $i, j \in \{1, .., n\}$,

$$i \neq j \wedge \bigwedge_{i'=1}^{n} i = i' \Rightarrow r_{i'} \wedge \bigwedge_{j'=1}^{n} j = j' \Rightarrow r_{j'}$$

Completeness:

$$\neg(speed < 30) \wedge \neg(speed > 30) \wedge \neg(speed > 40)$$

or: with n row headers $r_1, ..., r_n$, $\bigwedge_{i=0}^{n} \neg r_i$

Typical Solver Tasks

Any

Applicability: Is there a value assignment satisfying all E_i? Examples are any set of boolean expressions, or even a single complex one.

Completeness: For any combination of inputs, does at least one expression E_i match? Examples include conditionals, `switch` statements, `alt`-expressions (see below), decision tables, or transition guards in state machines.

Overlap: For any combination of inputs, does at most one expression E_i match? Examples include any set of Boolean expressions that are *not* ordered, so no two can match any given set of inputs. Often used together with completeness, hence the same examples apply.

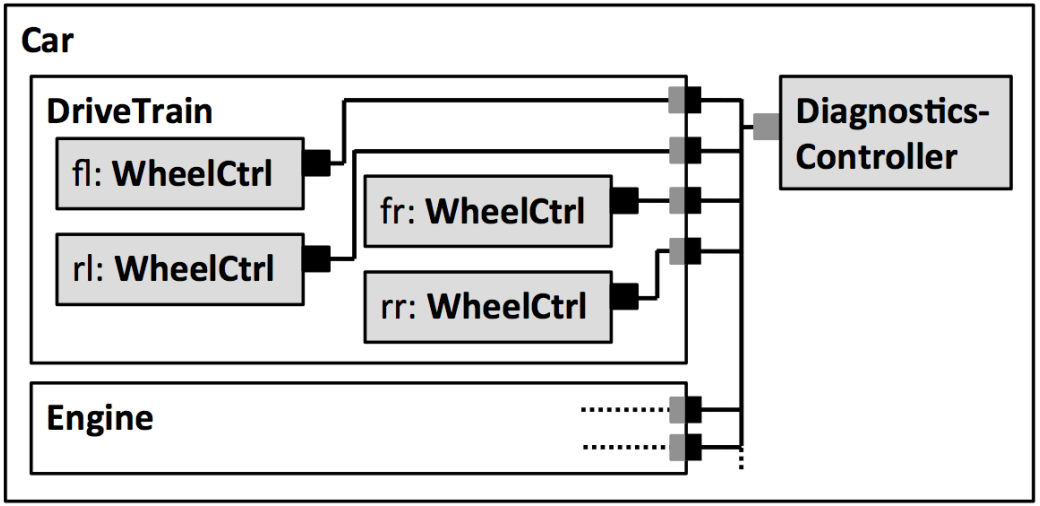
Decision Tables, Guards in SM

Refactorings

Equality: Are the E_i semantically equivalent, even though they differ structurally (think: DeMorgan laws). Examples include refactorings that simplify expressions.

Subset: For any $i \in \{1, ..., n\}$, are the values satisfying E_{i+1} a subset of those satisfying E_i? The canonical example is a list of *ordered* decisions, the earlier one must be narrow to not shadow later ones; any kind of subtyping through constraints such as chained typedefs; producer-consumer relationships where the consumer must be able to consume everything the producer creates, or possibly more.

Producer/Consumer



Solvers can also be used to verify (large subsets of) programs wrt. arbitrary properties. But that's outside our scope.

Verification: Correctness of `switch`

SMT Solvers can help find/fix non-trivial logic errors in programs.

Error: [MANUALLY CHECKED] This alternative can never be true.

```
fun example(x: number) = alt [x < 0 && x > 0 => 1
                              x == 5          => 2]
```

Error: [MANUALLY CHECKED] Alternatives missing.
For instance the following case is not covered:
x = 0

```
fun example(x: number) = alt [x > 0 => 1
                              x < 0 => 2]
```

Error: [MANUALLY CHECKED] Overlapping alternatives.
The following case is covered by multiple alternatives:
x = 0

```
fun example(x: number) = alt [x >= 0 => 1
                              x <= 0 => 2]
```

Verification: Correctness of Tables

SMT Solvers can help find/fix non-trivial logic errors in programs.

```
fun isCapital(a_city: cit
```

Error: [MANUALLY CHECKED] Unreachable cell.

			a_country == Germany	a_city == Madrid
a_country == France	true	false		false
a_country == Germany	false	true		false
a_country == Spain	false	false		true

```
enum country { France, Germany, Spain }
enum city {
```

Error: [MANUALLY CHECKED] Overlapping Columns: These columns all match in the following case:
a_city = city:Paris

```
fun isCapital(a_city: city, a_country: country) =
```

	a_city == Paris	a_city == Paris	a_city == Madrid
a_country == France	true	false	false
a_country == Germany	false	true	false
a_country == Spain	false	false	true

```
fun isCapital(a_city: city, a_country: country) =
```

Error: [MANUALLY CHECKED] Overlapping Rows: These rows all match in the following case:
a_country = country:Germany

		= Berlin	a_city == Madrid
			false
a_country == Germany	false	true	false
a_country == Germany	false	false	true

Verification: Order/Shadowing

Earlier rows must be more specific than later rows to avoid shadowing.

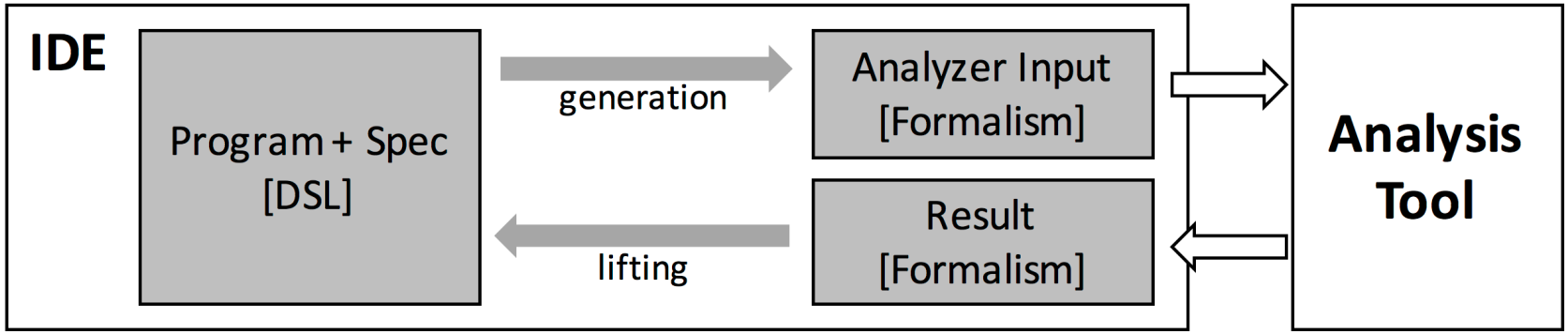
```
enum REGION { EU, ASIA, NA, ME }
enum COUNTRY { DE, FR, US, CA, JA }
type cur: number[0|∞]{2}
fun minutePrice(region: REGION, country: COUNTRY, rebated: boolean) =
```

	region	country	rebated	local: cur	longDis: cur
EU-rebated	*EU*		true	0.80	1.00
EU-non-rebated	*EU*		false	0.85	1.10
DE	*EU*	*DE , FR*	false	0.82	1.05
US	*NA*	*US*		0.70	0.75
CA	*NA*	*CA*		0.75	0.80
REST				1.00	1.20

Solver Integration

General Pattern:

SMT-LIB
THE SATISFIABILITY MODULO THEORIES LIBRARY



Specific Approach for Solver

Z3

kernelf

functional • embeddable • extensible

KernelF core features

Primitive Types

Numbers, Booleans, Strings, Enums, Records

Arithmetic Operations

+ - * /

Comparison Operations

> >= < <= == !=

Functions

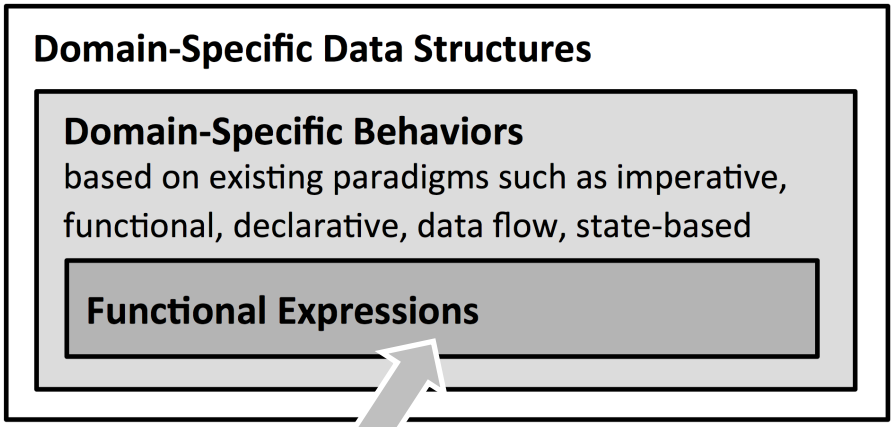
Definitions, Calls, Higher-Order

KernelF is a functional base language

KernelF Context: DSL Development

New Language

Analyze Domain to find Abstractions
Define suitable, new notations
Rely on existing behavioral paradigm
Reuse standard expression language
Interpret/Generate to one or more GPLs

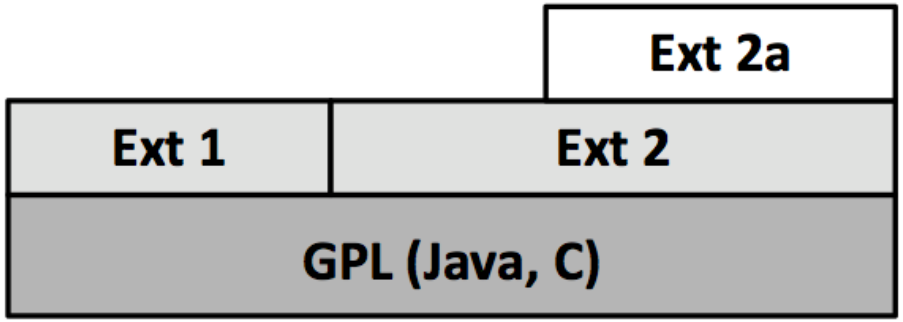


kernelf
functional • embeddable • extensible

KernelF Context: Three DSL Patterns

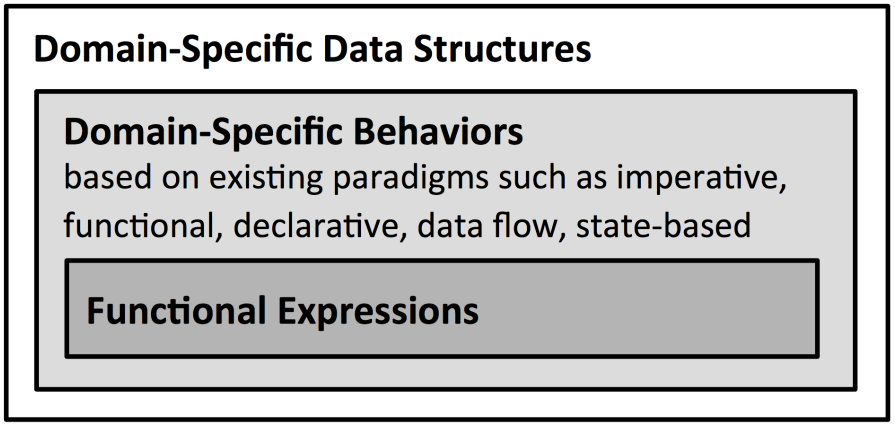
GPL Extension

Reuse GPL incl. Expressions and TS
Add/Embed DS-extensions
Compatible notational style
Reduce to GPL



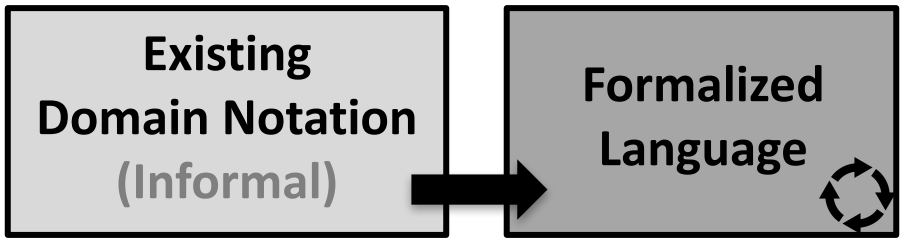
New Language

Analyze Domain to find Abstractions
Define suitable, new notations
Rely on existing behavioral paradigm
Reuse standard expression language
Interpret/Generate to one or more GPLs



Formalization

Use existing notation from domain
Clean up and formalize
Generate/Interpret
Often import existing „models“



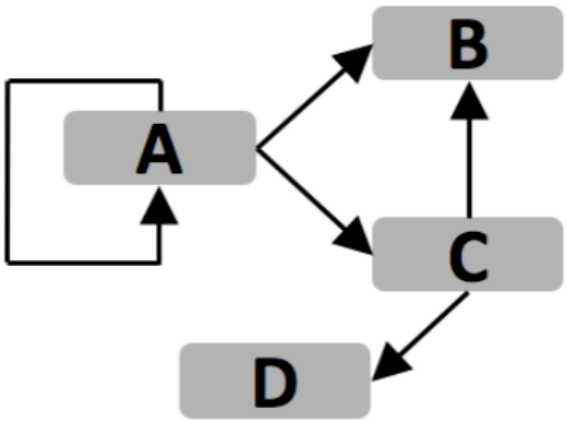
Safer Software through Better Abstractions and Static Analysis

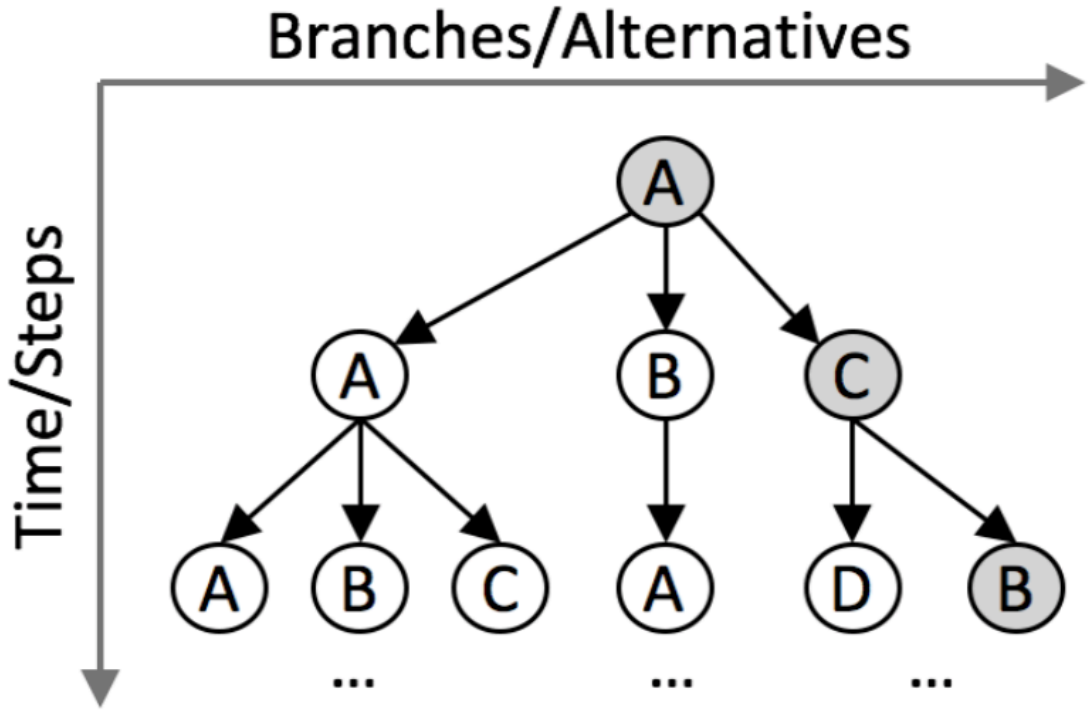
5

Verification: Model Checking

MV

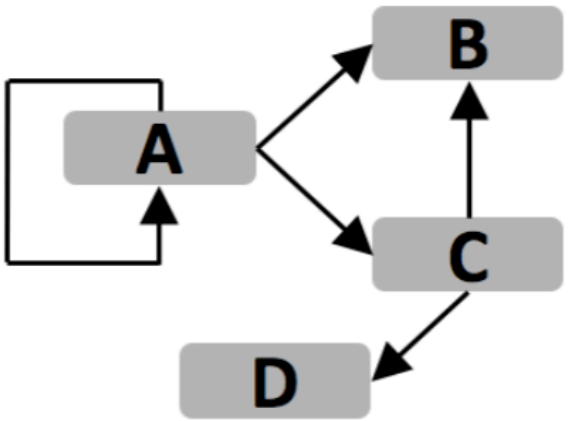
Basics about State Machines

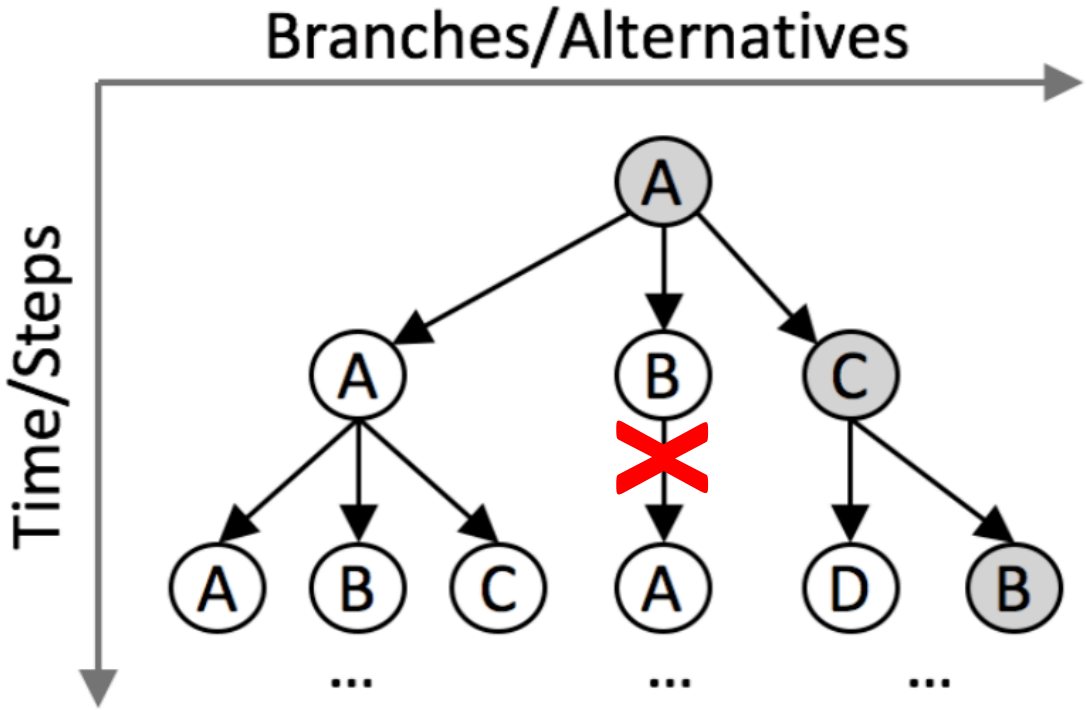




State machines express how a system's state changes over time, as it is stimulated externally, and how the system reacts to stimuli depending on the state it is in.

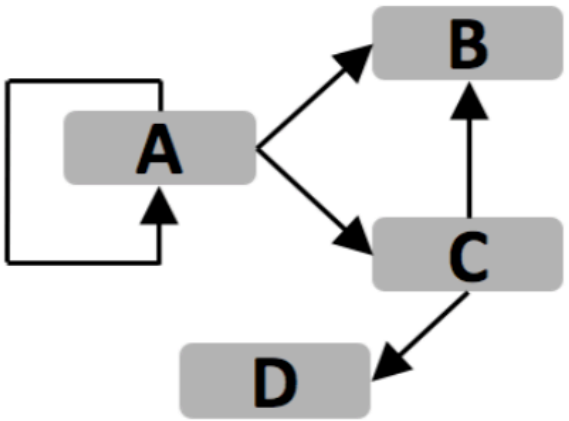
Basics about State Machines

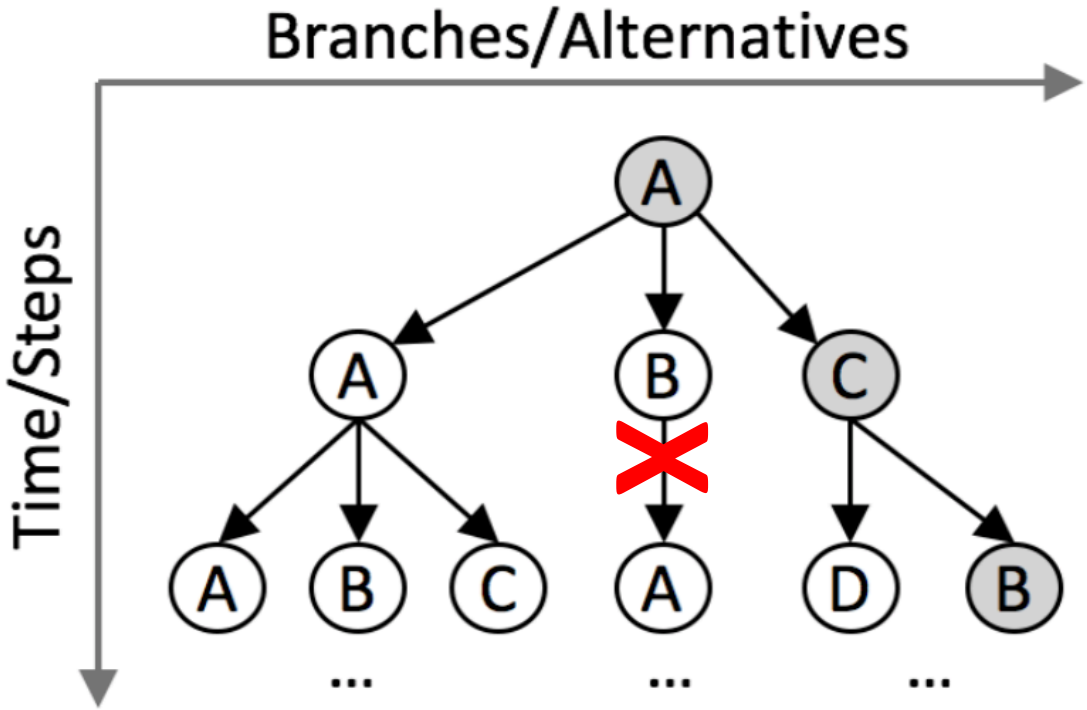




State machines express how a system's state changes over time, as it is stimulated externally, and how the system reacts to stimuli depending on the state it is in.

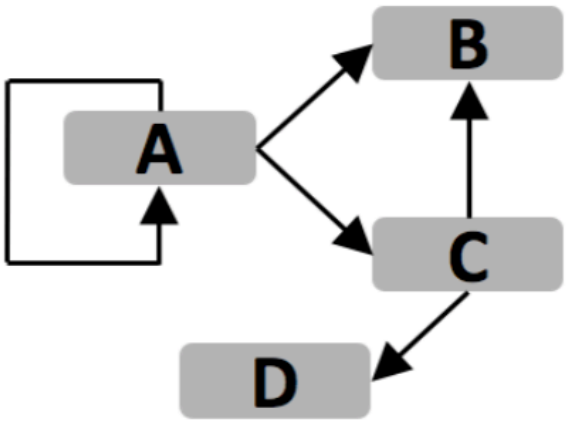
Basics about State Machines

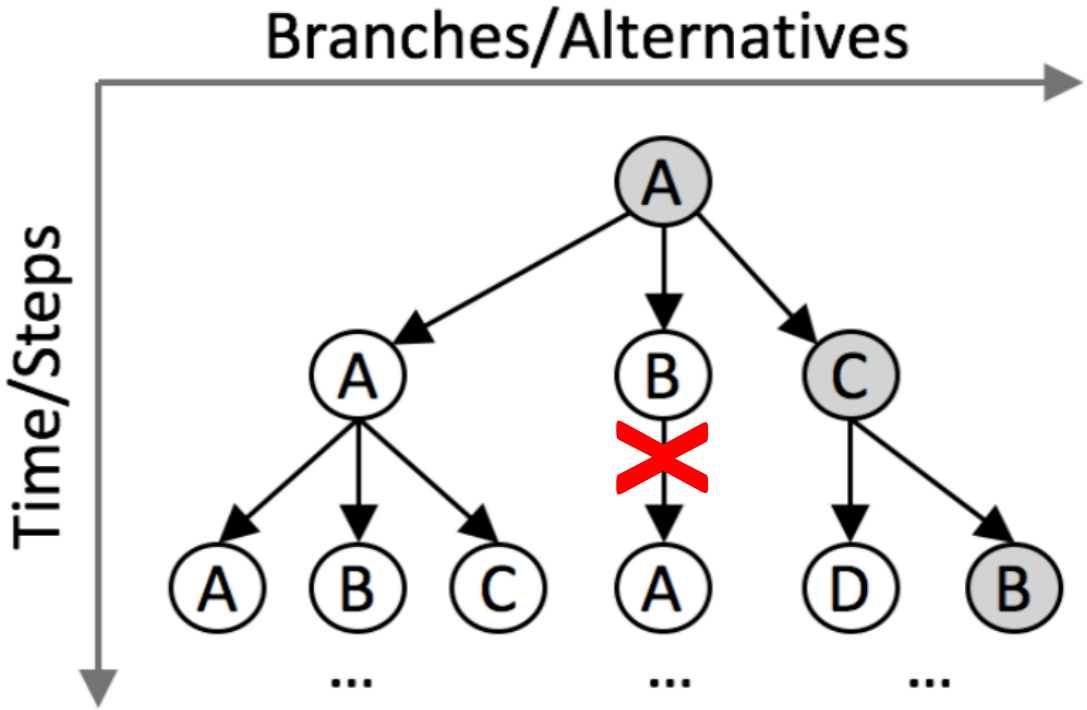




Every imperative program can be seen as a (very complicated) state machine.

Basics about State Machines

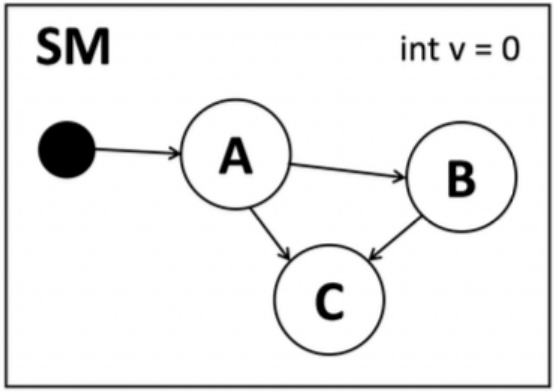




Every imperative program can be transformed into a (very complicated) state machine.

Temporal Logic (I)

Temporal logic can express time (= state change) as part of logic expressions.

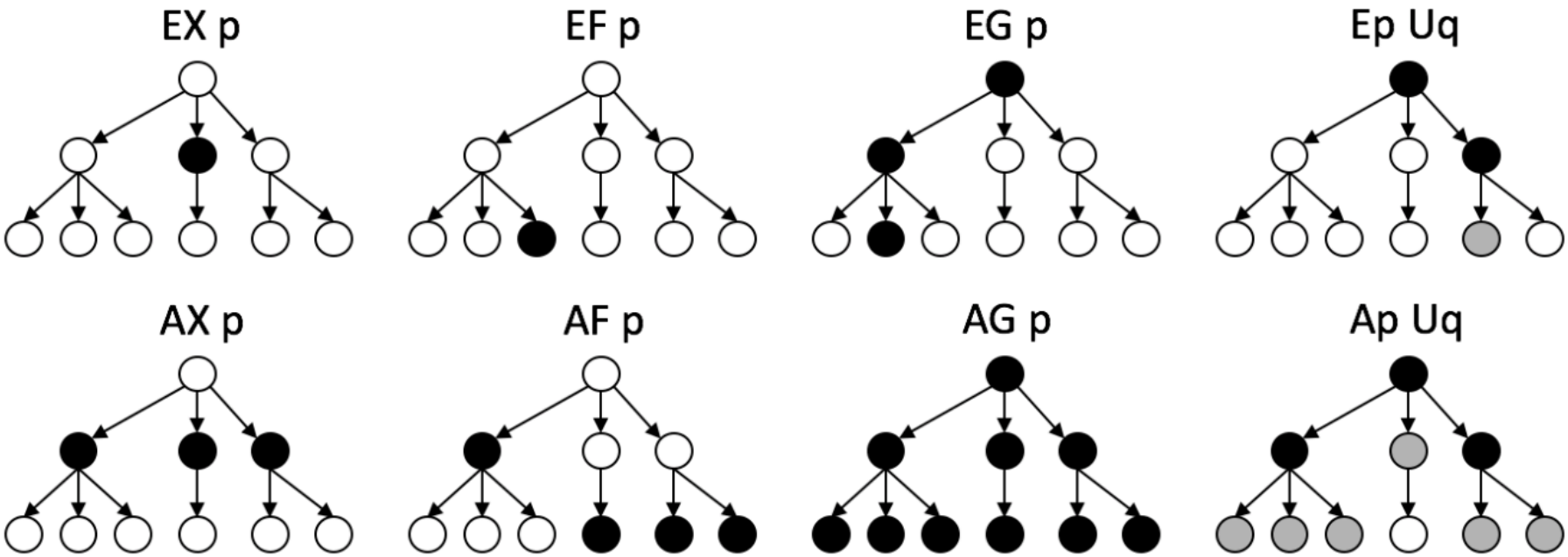


1. For all possible traces, after SM has been in state A, it will eventually move into state B.
2. For all possible traces, after SM has been in state A, it will move into state B or state C in the next step.
3. There exists an trace where, after SM has been in state A, it will eventually move into state B.
4. Before being in state B, SM has always been in A.
5. For all possible traces, SM will never reach state A after it was in state C.
6. Whenever SM is in state A, a variable v will never have a value greater than 10.

Temporal Logic (II)

Temporal Quantifiers are at the core of TL.

[Branch][Steps][Property]



1. AG((state == A) ⇒ AF(state == B)).
2. AG((state == A)
   ⇒ AF(state == B || state == C)).
3. EF(state == A ⇒ EF(state == B).
4. EF(state == B) ⇒ (state == A).
5. AG((state == C) ⇒ !EF(state == A)).
6. AG(state == A) ⇒ (v > 10).

What is Model Checking?

Express a system as a state machine.
Transform a system **into** a state machine.

Express allowed and disallowed behaviors (state evolutions over time) as temporal logic properties.

Use a Model Checker to verify that a property always holds (or show a counter example).

Assuming (!) a complete set of properties, this proves correctness of the system.

Model Checking C programs

Express properties as interface contract. Generate C, invoke CBMC to perform model check, lift results.

Components

```
section store {
  exported cs interface TrackpointStore1 {
    void store(Trackpoint* tp)
      pre(0) isEmpty()
      pre(1) tp != NULL
      post(2) !isEmpty()
    Trackpoint* get()
      pre(0) !isEmpty()
    Trackpoint* take()
      pre(0) !isEmpty()
      post(1) result != NULL
      post(2) isEmpty()
    query boolean isEmpty()
  }
```

Components

```
[checked]
exported component InMemoryStorage extends nothing {
  provides TrackpointStore1 store

  Trackpoint* storedTP = NULL;

  void store_store(Trackpoint* tp) <= op store.store {
    // here is a regular statement comment.
      @arg(tp) is a reference to an argument.
    return;
  } runnable store_store

  boolean store_isEmpty() <= op store.isEmpty {
    return storedTP == NULL;
  } runnable store_isEmpty
} component InMemoryStorage
```

CBMC Lifted Results

Idx	Property	Status	Trace Size	Time
	Components Contracts (7)	FAIL (1)		11.2s
001	pre(0) isEmpty()	SUCCESS		1.92s
002	pre(1) tp != null	SUCCESS		1.89s
003	post(2) !isEmpty()	FAIL	40	1.96s
004	pre(0) !isEmpty()	SUCCESS		1.58s
005	pre(0) !isEmpty()	SUCCESS		1.61s
006	post(1) result != null	SUCCESS		1.39s
007	post(2) isEmpty()	SUCCESS		0.85s

Filter: Regex

Mark Results in Editor Project cex.

Idx	Raw	Thread	Kind	Value
1	6	0	call	Components_verification
2	8	0	call	UnitTestMessages____testing_runn...
3	9	0	return	UnitTestMessages____testing_runn...
4	11	0	call	Components_verificationInstances_...
5	17	0	call	Components_InMemoryStorage_init
6	19	0	return	Components_InMemoryStorage_init
7	20	0	return	Components_verificationInstances_...
8	22	0	call	Components_Interpolator_processo...
9	28	0	call	Components_InMemoryStorage_sto...
10	30	0	return	Components_InMemoryStorage_sto...
11	37	0	call	Components_InMemoryStorage_sto...
12	39	0	call	Components_InMemoryStorage_sto...
13	41	0	return	Components_InMemoryStorage_sto...
14	45	0	call	Components_InMemoryStorage_sto...
15	47	0	return	Components_InMemoryStorage_sto...
16	49	0	FAIL	Error Label Reached

Filter: Regex

Call/Return Locations Last 100 Re-run

Verification: Heartbleed Bug I

Heartbleed

From Wikipedia, the free encyclopedia

For other uses, see Heartbleed (disambiguation).

Heartbleed is a security bug in the OpenSSL cryptography library, which is a widely used implementation of the Transport Layer Security (TLS) protocol. It was introduced into the software in 2012 and publicly disclosed in April 2014. Heartbleed may be exploited regardless of whether the party is using a vulnerable OpenSSL instance for TLS as a server or a client. It results from improper input validation (due to a missing bounds check) in the implementation of the TLS heartbeat extension,[3] thus the bug's name derives from *heartbeat*.[4] The vulnerability is classified as a buffer over-read,[5] a situation where more data can be read than should be allowed.[6]

Heartbleed is registered in the Common Vulnerabilities and Exposures database as CVE-2014-0160.[5] The federal Canadian Cyber Incident Response Centre issued a security bulletin advising system administrators about the bug.[7] A fixed version of OpenSSL was released on April 7, 2014, on the same day Heartbleed was publicly disclosed.

As of May 20, 2014, 1.5% of the 800,000 most popular TLS-enabled websites were still vulnerable to Heartbleed.[8]

TLS implementations other than OpenSSL, such as GnuTLS, Mozilla's Network Security Services, and the Windows platform implementation of TLS, were not affected because the defect existed in the OpenSSL's implementation of TLS rather than in the protocol itself.[9]

Logo representing Heartbleed. Security company Codenomicon gave Heartbleed both a name and a logo, contributing to public awareness of the issue.[1][2]

Verification: Heartbleed Bug II

What they wanted to express:

```
struct {
  uint16 payload_length;
  unsigned char payload[payload_length];
} HeartbeatMessage;
```

The above code is invalid in C

cannot „dynamically“ initialize length.

Instead „manual“ synchronization between the two values.

Bug: sizeof(payload) != payload_length

Led to Buffer Over-read.

Verification: Heartbleed Bug III

Find Problem via formal Verification

```
HeartbeatMessage prepareUntrustedMessage() {
  HeartbeatMessage msg;
  assign nondet msg;
  return msg;
}
```

HeartBleed

```
//[ Parsing the message ]
size_t length = pM->payload_length;
uint8* p = pM->payload;

//[ Just some memory to read into, a
void* dest = malloc(length);
//[ Here a problem happens, because
memcpy(dest, p, length);
```

Verification (CBMC)

Idx	Property	Status	Trac...	Time
40	dereference fail...	FAIL	59	0.44s
41	dereference fail...	SUCCESS		0.35s
42	arithmetic overfl...	SUCCESS		0.29s

Idx	Raw	Thre...	Kind	Value
17	0	0	m.payload_le...	25
18	0	0	m.payload_le...	25
19	0	0	m.payload[0]	0
20	46	0	**return**	prepareUnkn...

Verification: Heartbleed Bug IV

Even better: First-class message concept that handles low level message stuff.

```
// a field representing a times
uint8[6] f_time = {0x00A,  // f
                   UNIT_TIME24,
                   3, // 3 payl
                   10, 20, 00 /
                  };

// a field representing a measured value
uint8[4] f_value = {0x04D,  // field type identifier
                    UNIT_QDOT, // unit used: mass flow
                    1, // 1 payload byte follows
                    &dataField // addr of variable
                   };

// a message that uses the two fields
uint8[5] message = {0xAEE, // message type idenfier
                    ID, // unique running message ID
                    2, // two fields folllowing
                    f_time, // embed the time field
                    f_value// embed the value field
                   };
```

```
message CurrentMeasuredValue:42 {
  int32     timestamp; // time of measurement
  uint16/A/ value;     // measured value in Amps
  uint16    accuracy;  // accuracy in 1/100 %
}
message ... { ... }
...
```

Verification: Pacemaker Challenge I

Report from Dagstuhl Seminar 14062

The Pacemaker Challenge: Developing Certifiable Medical Devices

Edited by

Dominique Méry[1], Bernhard Schätz[2], and Alan Wassyng

1 LORIA – Nancy, FR
2 fortiss GmbH – München, DE
3 McMaster University – Hamilton, CA, wassyng@mcmaster.ca

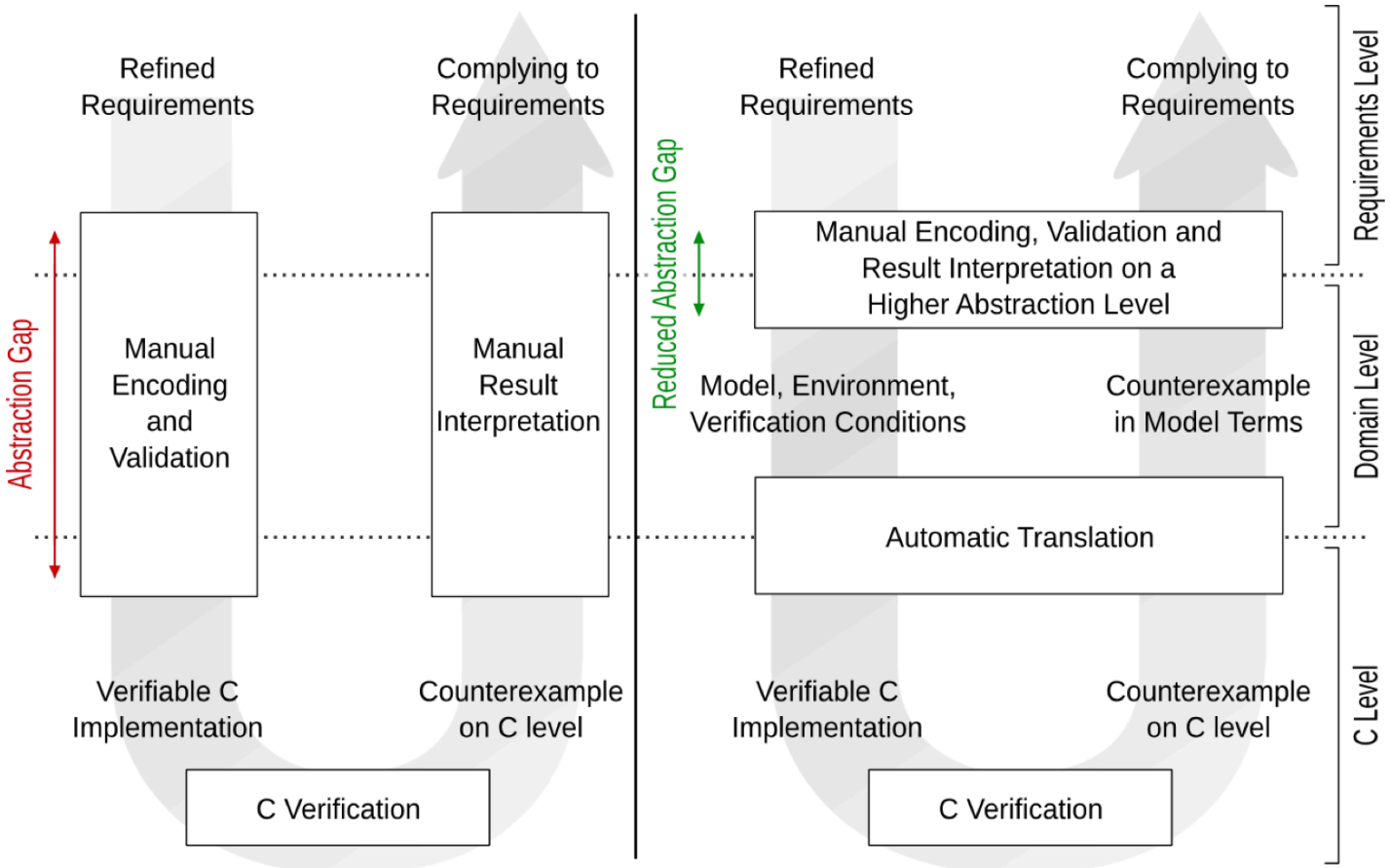
Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 14
maker Challenge: Developing Certifiable Medical Devices". The aim of the semi
together leading researchers and industrial partners of this field; the seminary e
participants from 8 countries: Canada, Denmark, France, The Unites States, G
Kingdom, Brazil. Through a series of presentations, discussions, and working
the seminar attempted to get a general view of the field of medical devices and ce
through the pacemaker challenge. The seminar brought together, on the one hand, researchers
from the different notations and various tools. The main outcome of the seminar is the exchange
of information between different groups and the project of a book.

Seminar February 2–7, 2014 – http://www.dagstuhl.de/14062
1998 ACM Subject Classification D.2 Software Engineering, K.4 Computers and Society
Keywords and phrases Embedded systems, Real-time systems, Medical devices, Model-driven development, Software certification, Validation & verification, Formal methods
Digital Object Identifier 10.4230/DagRep.4.2.17

Verification: Pacemaker Challenge II

Better: extend C with a DSL to express safety properties as well as proof strategies



Verification: Pacemaker Challenge II

Left: system is modeled as a state machine in C

Right: the representation in low-level C

```
statemachine VVI initial = Start
  in config(int lri, int vrp)
  in s
  in t
  out p => doPace()
  int c = 0
  int LRI = 0
  int VRP = 0

  state Start {
    on config -> Start {
      LRI = lri;
      VRP = vrp; }
    on t -> Pace
  }

  state Wait {
    on s [c <= VRP] -> Wait
    on s [c > VRP] -> Wait {
      c = 0; }
    on t [c < LRI] -> Wait { ++c; }
    on t [c == LRI] -> Pace
  }

  state Pace {
    on entry {
      send p;
      c = 0; }
    on t -> Wait {++c; }
  }
```

⇨

```
typedef enum  states {
  Start,  Wait,  Pace
} VVI_States;

VVI_States state;
int c;
int LRI;
int VRP;

typedef enum events {
  config,  s,  t
} Events_VVI;
```

```
void execute(Events_VVI e,
                    int** args)
{ switch (state) {
    case Start: {
      switch (e) {
        case config: {
          LRI = *args[0];
          VRP = *args[1];
          state = Start;
          return; }
        case t: {
          state = Pace;
          doPace();
          c = 0;
          return; }
```

```
case Wait: {
  switch (e) {
    case t: {
      if ( c < LRI ) { ++c;
        return ; }
      if ( c == LRI ) {
        state = Pace;
        doPace();
        c = 0;
        return; }
    }
    case s: {
      if ( c <= VRP ) {
        state = Wait;
      }      // ...
```

Verification: Pacemaker Challenge III

Safety Properties (and their tx to C)

```
after smIsInState(Wait)
  until smIsInState(Pace)
  exists c == LRI
```

⇨

```
static int q = 0;
static int e = 1;
if (state == Pace) {
  q = 0;
  assert(e);
  e = 1;
}
```

```
if ( q ) {
 e = (c == LRI) || e;
}
if (state == Wait
      && !q)
{ q = 1; e = 0; }
```

A

```
nondet_choice {
  choice {
    smtrigger(s);
  }
  choice {
    // nothing
  }
}
```

⇨

```
int ch = nondet_int();
assume(0 <= ch
            && ch < 2);
if (ch == 0) {
  execute(s, 0);
}
if (ch == 1) {
  // nothing
}
```

B

```
smStateSubset Initial:
  smInState(Pace)
    && VRP < LRI;

smNonDetInit(Initial);
```

⇨

```
state = (VVI_States)
            nondet_int();
assume(0<= state
      && state <= 2);
c = nondet_int();
LRI = nondet_int();
VRP = nondet_int();
assume(state == Pace
            && VRP < LRI);
```

C

Verification: Pacemaker Challenge IV

Inductive Proof

```
induction on t
  from Initial
  until MAX_LRI {
    nondetermenistically smtrigger(s);
    after      smIsInState(Wait)
      until smIsInState(Pace)
        exists c == LRI }
```

⇨

```
smNonDetInit(Initial);
for (int step = 0; step < MAX_LRI; ++step) {
  nondetermenistically smtrigger(s);
  after smIsInState(Wait)
    until smIsInState(Pace)
      exists c == LRI;
  smtrigger(t);
  after step > 1 until step == MAX_LRI
    exists smIsInStateSubset(Initial); }
```

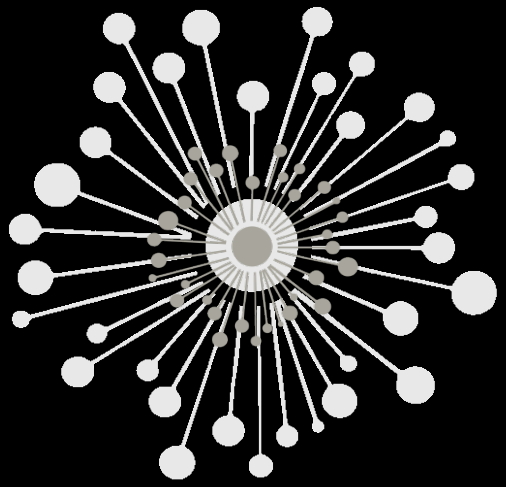
Generic tools don‘t find Specific problems

Guide the tools

Translate to their input language

Safer Software through Better Abstractions and Static Analysis

6



Verification: Test Generation

BK

Test Vector Generation

kernelf
functional • embeddable • extensible

For everything that can be seen as taking a list of arguments, those can be synthesized.

```
test case TestFunctions [success] {
  vectors function add  -> producer: random 25
          results: true
```

	a	a2	b	c	s	res	status
0: valid	3	-3.06	false	BLUE	""	3	ok
1: valid	2	2.74	false	BLUE	"M/Yh-0I/ac"	2	ok
2: valid	1	0.22	false	BLUE	""	1.22	ok
3: invalid input	4	-0.45	true	BLUE	"7l:6h7sg!afLmULGU8wtI00H9"		not executed
4: invalid input	1	1.38	false	RED	"Mtoa7J66uuWTye2f2-fLhD§hj8C2K"		not executed
5: invalid input	1	-2.63	false	BLUE	"n66r7E (f0J$aQMj§"		not executed

```
vectors function plus -> producer: eqclass
        results: true
```

	a	b	c	res	status
0: valid	0	-10	GREEN	-10	ok
1: valid	0	-10	BLUE	-10	ok
2: valid	0	10	GREEN	10	ok
3: valid	0	10	BLUE	10	ok
4: valid	0	-1	GREEN	-1	ok

	a	b	status
0: valid	7	2	no expected value given; actual was 14
1: valid	1	8	no expected value given; actual was 8
2: valid	5	2	[POST] res == a * b
3: valid	7	8	no expected value given; actual was 56
4: valid	5	6	[POST] res == a * b
5: valid	8	0	[PRE] b > 0
6: valid	5	8	[POST] res == a * b

Mutation Testing

kernelf
functional • embeddable • extensible

For a set of tests that all succeed, if after a change to the program they still do, this is a problem.

```
fun doodle(a: number[1|5]) = if true then a else a * 1

fun add(a: number[1|5]) = alt [ a > 3      => a + 1
                                otherwise => doodle(a) ]

test case TestFunctions [incomplete] {
  vectors function add                        -> producer: random 30
          results: true                          { 30 entries }
          mutator: # of mutations 20
                   keep all:      false
                     -> ParensExpression
                     -> NumberLiteral
                     -> LogicalNotExpression
                     -> ParensExpression
                     -> ParensExpression
}
```

```
fun add(a: number[1|5]) = alt [ ((a) + 1) > 3 => a + 1
                                a
                                otherwise     => doodle(a) ]

fun doodle(a: number[1|5]) = if true then a else a * 2
                                                     1

fun doodle(a: number[1|5]) = if true then a else ((a * 1) + 1)
                                                 a * 1

fun doodle(a: number[1|5]) = if !(true) then a else a * 1
                                true
```

Coverage Analysis

```
assessment: TestCoverage
query:      test coverage in current model
              problems only: ☐
              languages: << ... >>
              ignore:    << ... >>
sorted: ☒    must be ok: ☐    hide ok ones: ☐
last updated: Dec 16, 2016 (16 months ago) by Wladimir
```

test.in.expr.os.ml

data	Covered.
hasAThree	Partial.
many	Covered.
dontGiveMeAnInt	Covered.
lambda3Arg	Partial.
lambda3Bind1	Covered.
lambda3Bind2	Covered.
invalidPair	Missing.

Are all parts of the test code executed in the test?

Coverage Analysis

```
assessment: StructuralCoverage
query:      structural test coverage {...}
sorted: ☒    must be ok: ☐    hide ok ones: ☐
last updated: Feb 28, 2017 (9 months ago) by marku
```

```
org.iets3.core.expr.base
▌ TupleValue                    Covered. N=2, V
▌ SomeValExpr                   Covered. N=11,
▌ LogicalImpliesExpression      Covered. N=4, V
▌ ErrorLiteral                  Covered. N=27,
▌ ErrorExpression               Covered. N=9, V=24 H=1..3
▌ RangeTarget                   Covered. N=13, V=104 H=4..7
▌ TupleAccessExpr               Covered. N=4, V=10 H=2..3
▌ SomeExpression                Covered. N=21, V=252 H=4..9
▌ NoneExpression                Covered. N=21, V=58 H=1..6
▌ CastExpression                Covered. N=3, V=12 H=3..6
▌ AsSetOp                       Covered. N=5, V=23 H=3..8
▌ SimpleSortOp                  Covered. N=5, V=22 H=3..5
▌ ForeachOp                     Missing.
▌ OldMemberRef                  Covered. N=1, V=32 H=8..8
▌ RecordMember                  Partial. Missing: [type_old, type_old]
▌ RecordLiteral                 Partial. Missing: [type_old]
```

Is every language construct used in the tests, with all relations, and in which degrees of complexity?

Coverage Analysis

Are all parts of the interpreter (incl. all branches) executed in the tests?

```
assessment: InterpreterCoverage
query:      interpreter test coverage {...}
sorted: ☒    must be ok: ☐    hide ok ones: ☐
last updated: Feb 28, 2017 (9 months ago) by marku
```

org.iets3.core.expr.base

TupleValue	Covered.
SomeValExpr	Covered.
LogicalImpliesExpression	Covered.
ErrorExpression	Covered.
RangeTarget	Covered.
TupleAccessExpr	Covered.
SomeExpression	Covered.
ParensExpression	Covered.
TryExpression	Partial. Missing: [success]
IfExpression	Covered.

```
total 99, new 1, ok 0
value ranges
  decimal.max      = 340.00
  decimal.min      = -80.01
  decimal.zero     = true
  integer.zero     = true
  integer.max      = 99383
  decimal.minusOne = true
  integer.one      = true
  integer.minusOne = true
  decimal.one      = true
  integer.min      = -99383
coverage 99 %
```

Safer Software through Better Abstractions and Static Analysis

7

Validation: Review

BK

Better Abstraction, Better Review

This is better than a switch-case:

```
statemachine TrafficLights initial = red {
  in event timer(int64 t) <no binding>
  in event buttonPressed() <no binding>
  var boolean button = false
  var int64 tEnter = 0
  state red {
    on buttonPressed [ ] -> red { button = true; }
    on timer [t - tEnter > 1000] -> green { tEnter = t; }
  } state red
  state green {
    on timer [t - tEnter > 500] -> red { tEnter = t; }
  } state green
}
```

```
statemachine TrafficLights initial = red {
```

		Events	
		timer(int64 t)	buttonPressed()
States	red	[t - tEnter > 1000] -> green { tEnter = t; }	[ ] -> red { button = true; }
	green	[t - tEnter > 500] -> red { tEnter = t; }	

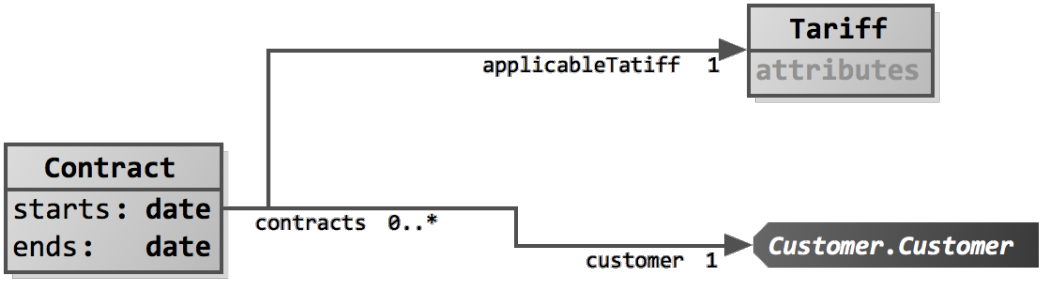
```
}
```

Better Notation, Better Review

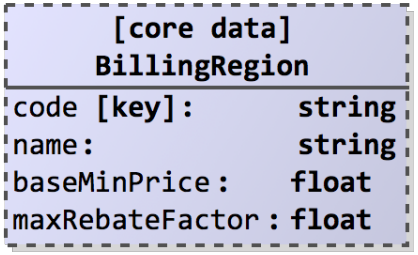
```
int32 averageIntArray(int32[] arr, int32 size) {
```

$$\text{return } \frac{\sum_{i=0}^{size} arr[i]}{size};$$

```
} averageIntArray (function)
```



Code	Name	Base Min Price	Max Rebate
BW	Baden Württemberg	0.20	0.8
BY	Bayern	0.20	0.8
BE	Berlin	0.15	0.7
BB	Brandenburg	0.10	0.7
HB	Bremen	0.20	0.7
HH	Hamburg	0.15	0.7
HE	Hessen	0.15	0.7
MV	Mecklenburg-Vorpommern	0.10	0.7
NI	Niedersachsen	0.15	0.7
NW	Nordrhein-Westfalen	0.15	0.7
RP	Rheinland-Pfalz	0.15	0.7
SL	Saarland	0.15	0.7
SN	Sachsen	0.10	0.7
ST	Sachsen-Anhalt	0.10	0.7
SH	Schleswig-Holstein	0.15	0.7
TH	Thüringen	0.10	0.7



Code Review and Security Audits

Track review state in the code

```
[Review: ready   Reviewed 6 days ago by zaur ]
exported component Judge2 extends nothing {
  int16 points = 0;
  void judger_reset() <= op judger.reset {
    points = 0;
} }
```

Assess review state over the system:

```
Assessment: ReviewOfComponentsStuff
query:      code review summary for chunk Components
```

reviewed	instancesJudging [InstanceConfiguration]
ready	Judge2 [AtomicComponent]
raw	testJudging [TestCase]
not reviewed yet	ContractMessages [MessageDefinitionTable]

Safer Software through Better Abstractions and Static Analysis

8

Validation: Simulation

BK

Debugging

```
fun circumference(r: number[0|∞]) = π * 2 * r

fun hypotenuseLength(a: number, b: number) = √(a² + b²)

fun cathetusLength(a: number, c: number) = √(c ⇒ 6² + a ⇒ 3²)
                                            ⇒ 36   ⇒ 9
                                            ⇒ 45.0
                                            ⇒ 6.7082039324993694151544332271441817283630371093 75
                                            ⇒ 6.7082039324993694151544332271441817283630371093 75

test case testPythagoras [fail] {
  assert hypotenuseLength(3, 4) equals 5 [1 ms]
  assert cathetusLength(3, 6)   equals 4 actual: 6.7082039324993694151544332271441817283630371093 75
}

test case testCircumference [fail] {
  assert circumference(-5) equals 14 actual: <constraint failed>
}
```

Simulation

Debugger for AnnidatedSM1

State machine AnnidatedSM1

<<<< <<< << < 0 days 04:00:00 > >> >>> >>>>

Inputs

Submit inputs

Interact

foo

bar

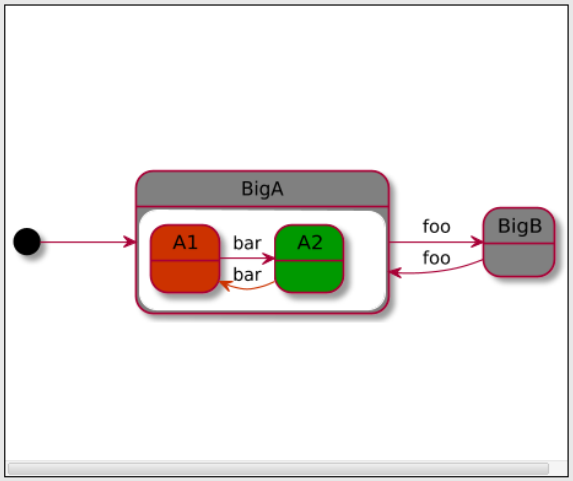
zum

Main

Time in state: 0 days 00:00:00

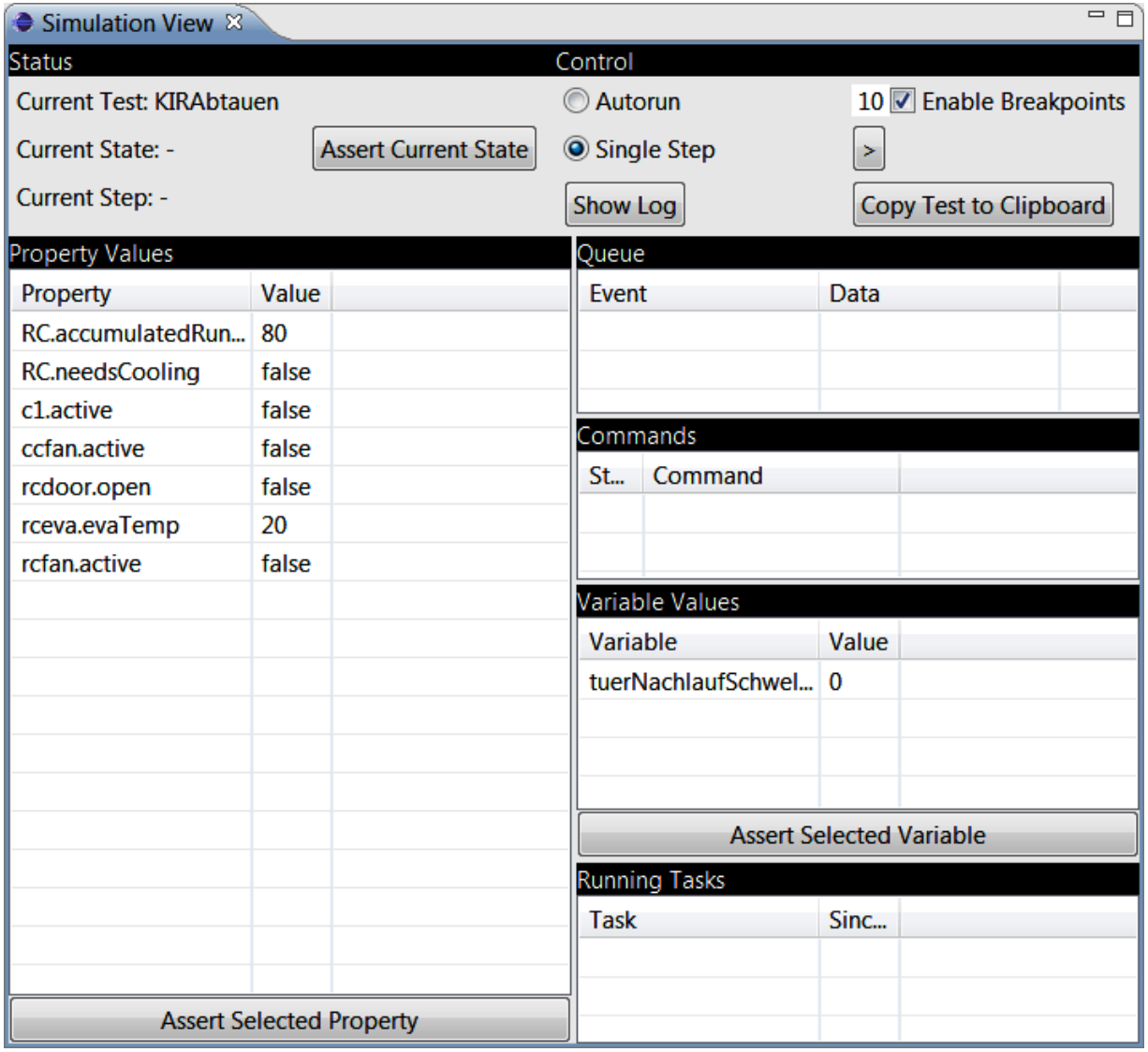
Current state: BigA.A2

Current Reachable Precedent Visited

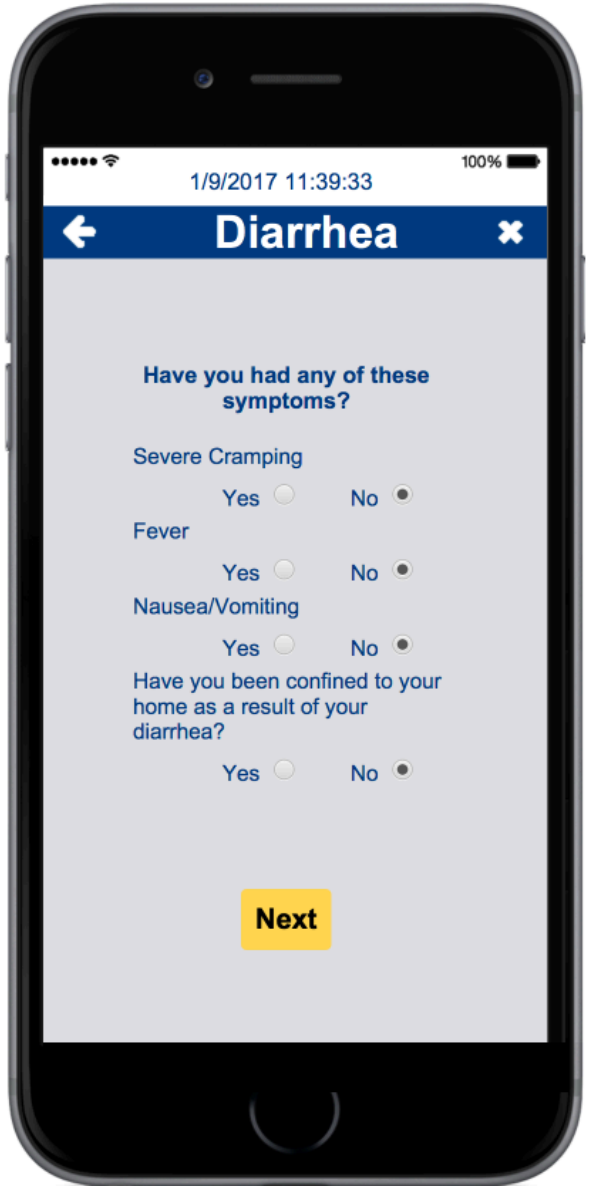


Time	Component	State	Type	Log
0 days 00:00:00	main	BigA.A1	COMMUNICATION	sending message "m"
0 days 00:00:00	main	BigA.A1	COMMUNICATION	s = in A1
0 days 00:00:00	-	-	CONTROL	Ticks 3600
0 days 01:00:00	-	-	CONTROL	Ticks 3600
0 days 02:00:00	-	-	CONTROL	Ticks 3600
0 days 03:00:00	-	-	CONTROL	Ticks 3600
0 days 04:00:00	main	BigA.A1	COMMUNICATION	Processing event bar
0 days 04:00:00	main	BigA.A1	STATE	following transition on bar to A2
0 days 04:00:00	main	BigA.A2	STATE	Entering state BigA.A2
0 days 04:00:00	main	BigA.A2	COMMUNICATION	sending message "m"
0 days 04:00:00	main	BigA.A2	COMMUNICATION	s = in A2

Simulation


Simulation View
Status
Control
Current Test: KIRAbtauen
Current State: -
Current Step: -
Assert Current State
Autorun
Single Step
10
Enable Breakpoints
>
Show Log
Copy Test to Clipboard
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
Assert Selected Property
Queue
Event
Data
Commands
St...
Command
Variable Values
Variable
Value
tuerNachlaufSchwel...
0
Assert Selected Variable
Running Tasks
Task
Sinc...

Simulation


100%
1/9/2017 11:39:33
Diarrhea
Have you had any of these symptoms?
Severe Cramping
Yes
No
Fever
Yes
No
Nausea/Vomiting
Yes
No
Have you been confined to your home as a result of your diarrhea?
Yes
No
Next

Impact Analysis

Propagation Explorer

SZ.6
Integrity Customizatio...
RAP: Very low

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	PUBL	
PRFC	RMAC	HRS	STRD	PUBL	

SZ.30
Authenticity WebsiteData
RAP: Very low

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	PUBL	
PRFC	RMAC	HRS	STRD	PUBL	

SZ.42
Integrity Webservice
RAP: Very low

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	PUBL	
PRFC	RMAC	HRS	STRD	PUBL	

B.3
Manipulation/Tampering...
RAP: Very low

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	PUBL	
PRFC	RMAC	HRS	STRD	PUBL	
Independent: Very low					
PRFC	RMAC	HRS	STRD	PUBL	
PRFC	RMAC	HRS	STRD	PUBL	

B.14
(Identity) Spoofing of...
RAP: Beyond high

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	RSTR	Beyond high
PRFC	RMAC	HRS	STRD	RSTR	Beyond high
Independent: Very low					
PRFC	RMAC	HRS	STRD	RSTR	
PRFC	RMAC	HRS	STRD	RSTR	

B.15
Man-in-the-Middle Atta...
RAP: Beyond high

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	RSTR	Beyond high
PRFC	RMAC	HRS	STRD	RSTR	Beyond high
Independent: Very low					
PRFC	RMAC	HRS	STRD	RSTR	
PRFC	RMAC	HRS	STRD	RSTR	

B.16
(Resource Location) Sp...
RAP: Beyond high

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	RSTR	Beyond high
PRFC	RMAC	HRS	STRD	RSTR	Beyond high
Independent: Very low					
PRFC	RMAC	HRS	STRD	RSTR	
PRFC	RMAC	HRS	STRD	RSTR	

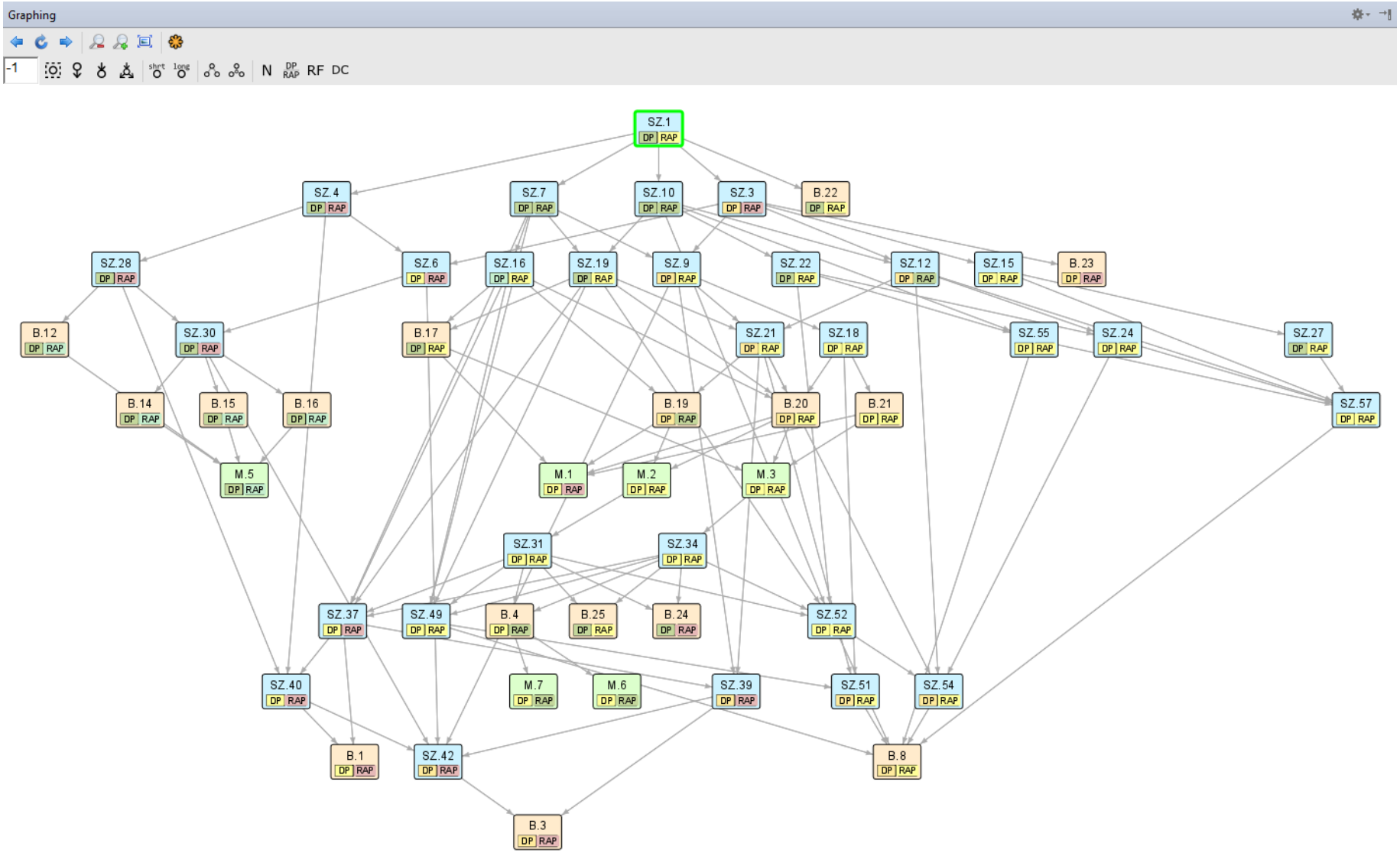
SZ.42
Integrity Webservice
RAP: Very low

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	PUBL	
PRFC	RMAC	HRS	STRD	PUBL	

B.3
Manipulation/Tampering...
RAP: Very low

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	PUBL	
PRFC	RMAC	HRS	STRD	PUBL	
Independent: Very low					
PRFC	RMAC	HRS	STRD	PUBL	
PRFC	RMAC	HRS	STRD	PUBL	

Understand Relationships



Safer Software through Better Abstractions and Static Analysis

9

Validation: Tracing

BK

mbeddr Requirements Support

Requirements

6.1 You should land as short as possible
ShortLandingRoll /functional: tags

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Suspendisse potenti. Etiam risus ante, bibendum ut mattis
vel condimentum velit. Quisque venenatis faucibus tellus
Phasellus rhoncus quam eu dui dictum sollicitudi

6.2 Once you land successfully, you get another
FullStop /functional: tags

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Suspendisse potenti. Etiam risus ante, bibendum ut mattis
purposes, this one references **@req**(InFlightPoints)

Rules

```
calculation PointForATrackpoint: This rule computes the points a
                                 It does so by taking into accou
                                 passed as arguments.
  parameters: [ int16 alt: current altitude of the trackpoint ] =
              [ int16 speed: current speed of the trackpoint ]
```

result = (*BASEPOINTS* * 1) *

	alt > 2000	alt > 1000
speed > 180	30	15
speed > 130	10	20

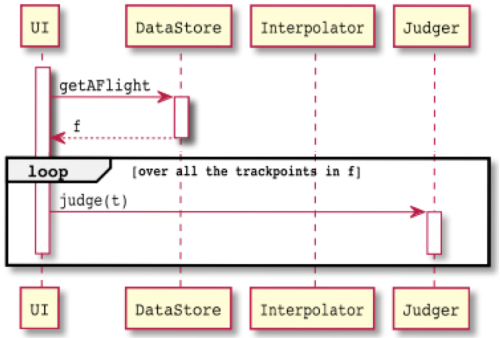
```
  tests: PointForATrackpoint(500, 100) == 0
         PointForATrackpoint(500, 1200) == 0
         PointForATrackpoint(1100, 165) == 200
         PointForATrackpoint(2100, 140) == 100
         PointForATrackpoint(2100, 200) == 300
```

Tracing

```
[#constant TAKEOFF = 100;] -> implements PointsForTakeoff
[#constant HIGH_SPEED = 10;] -> implements FasterThan100
[#constant VERY_HIGH_SPEED = 20;] -> implements FasterThan200
[#constant LANDING = 100;] -> implements FullStop

[checked]
exported statemachine FlightAnalyzer initial = beforeFli
  state beforeFlight {
    entry { points = 0; }
    on next [tp->alt > 0 m] -> airborne
    [exit { points += TAKEOFF; }] -> implements PointsForTakeoff
  } state beforeFlight
```

Visualisations



Requirements Tracing

Trace from **every** program element to requirements (internal or external)

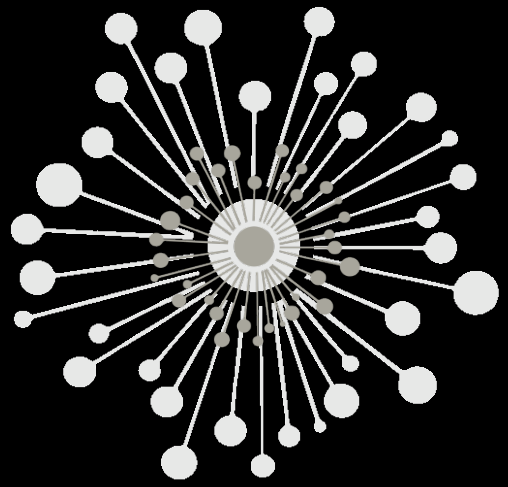
Analyze and find untraced program fragments.

Different kinds of traces to express semantics.

```
[#constant LANDING = 100;]-> implements FullStop
exported statemachine FlightAnalyzer initial = beforeFlight {
  state beforeFlight {
    on next [tp->alt > 0 m] -> airborne
    [exit { points += TAKEOFF; }]-> implements PointsForTakeoff
  } state beforeFlight
```

Safer Software through Better Abstractions and Static Analysis

10

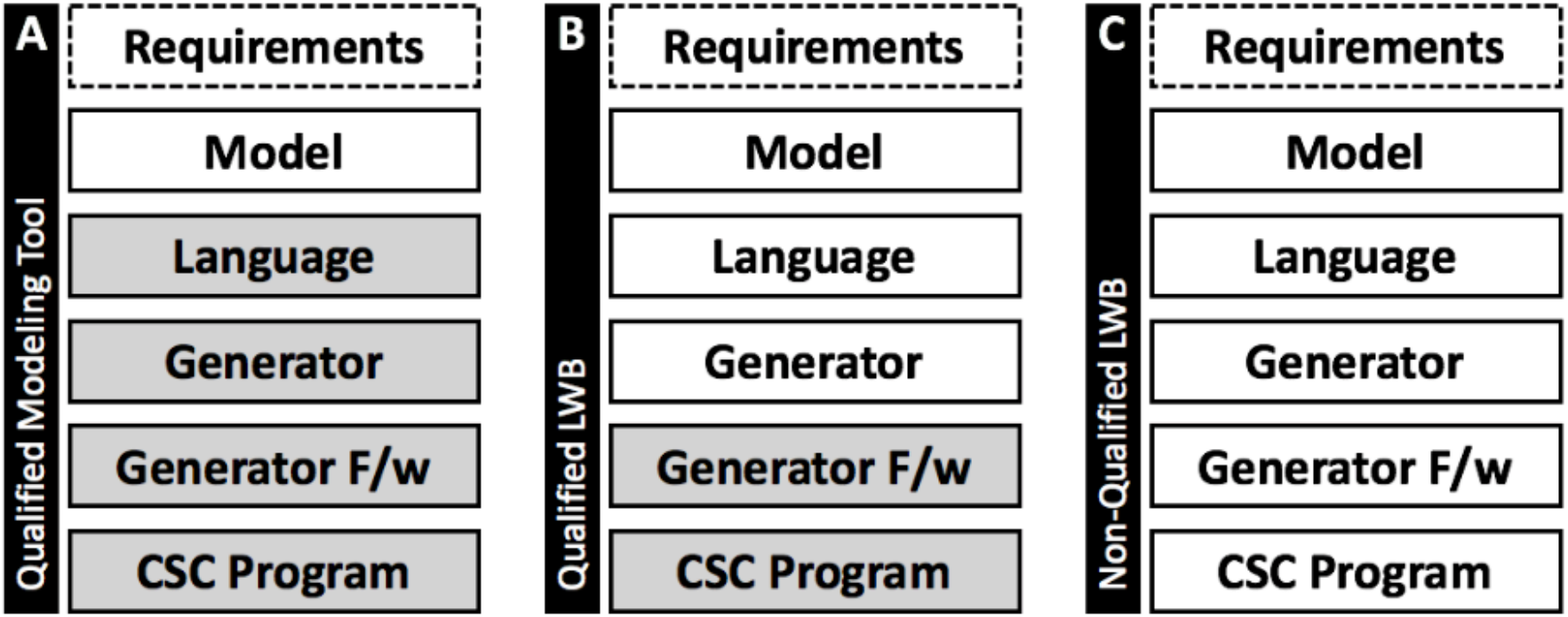


Assuring the Transformation

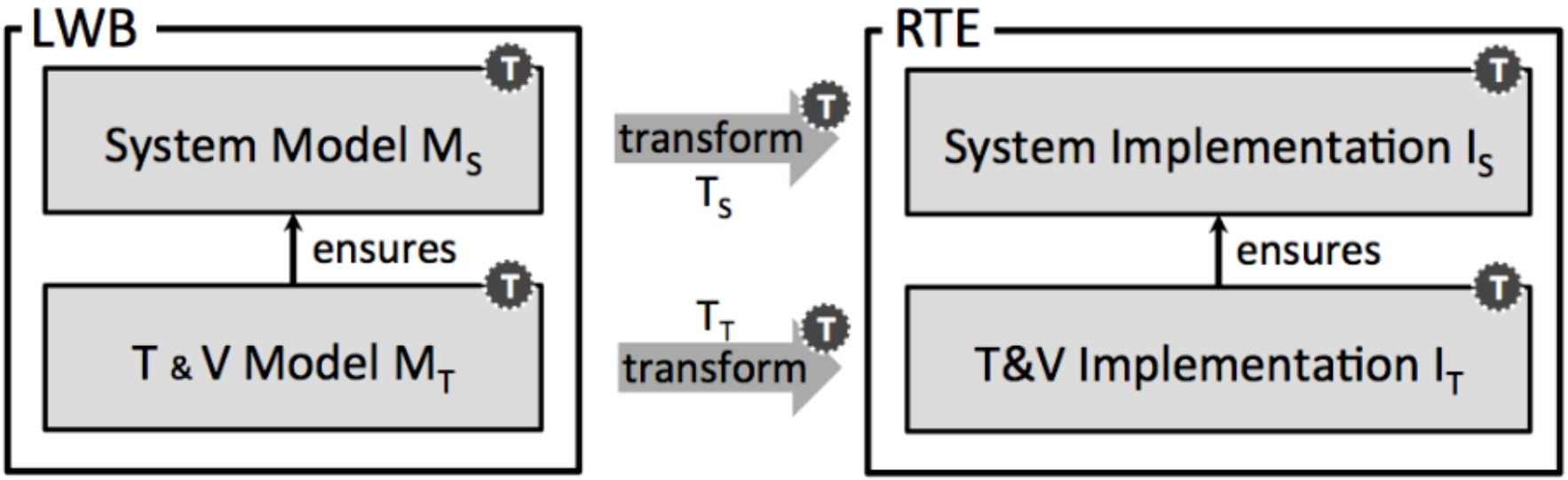
MV

What good is all the abstraction if we cannot trust the translation to the implementation?

A question of tool qualification.

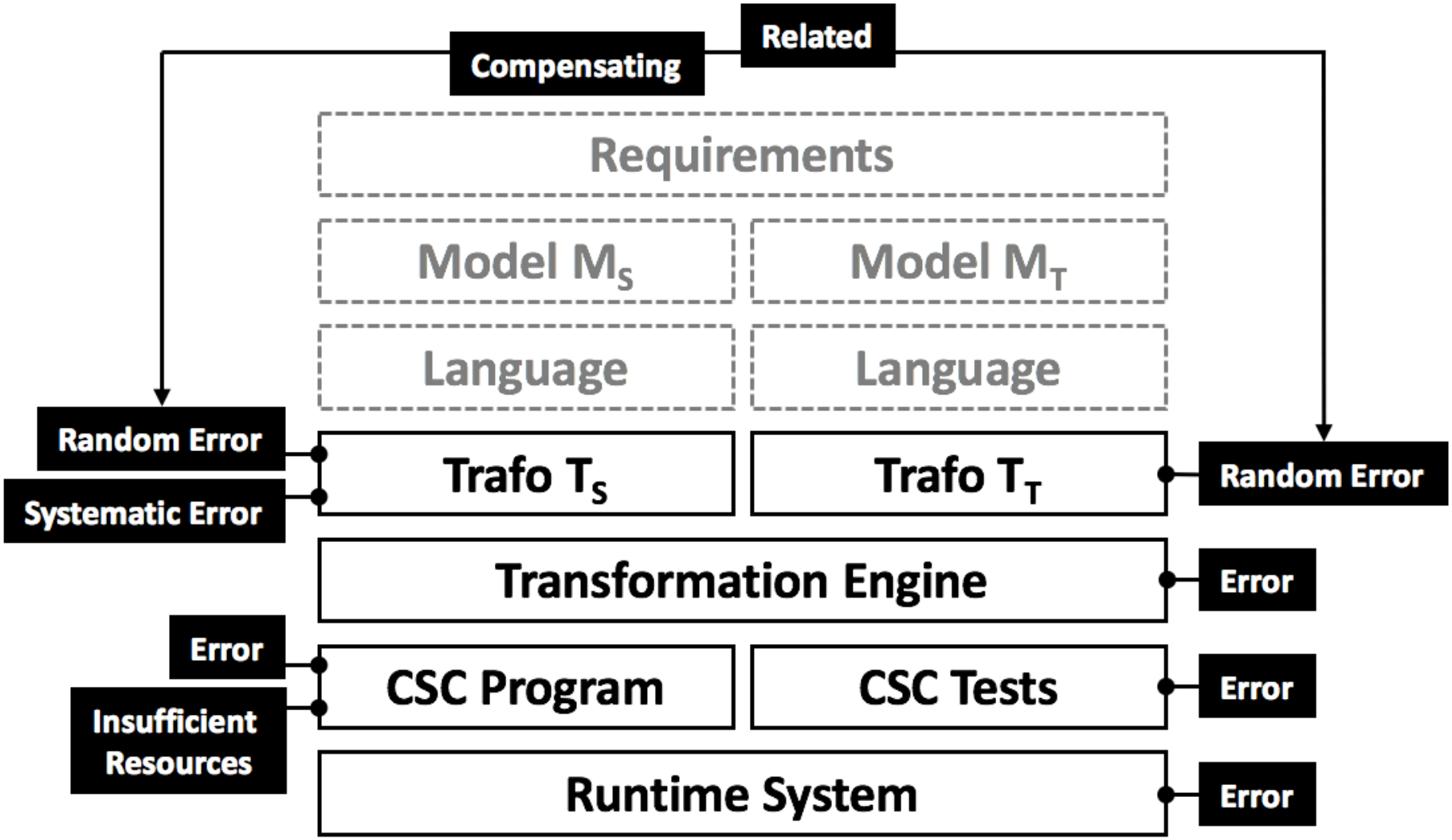


Baseline Tool Architecture

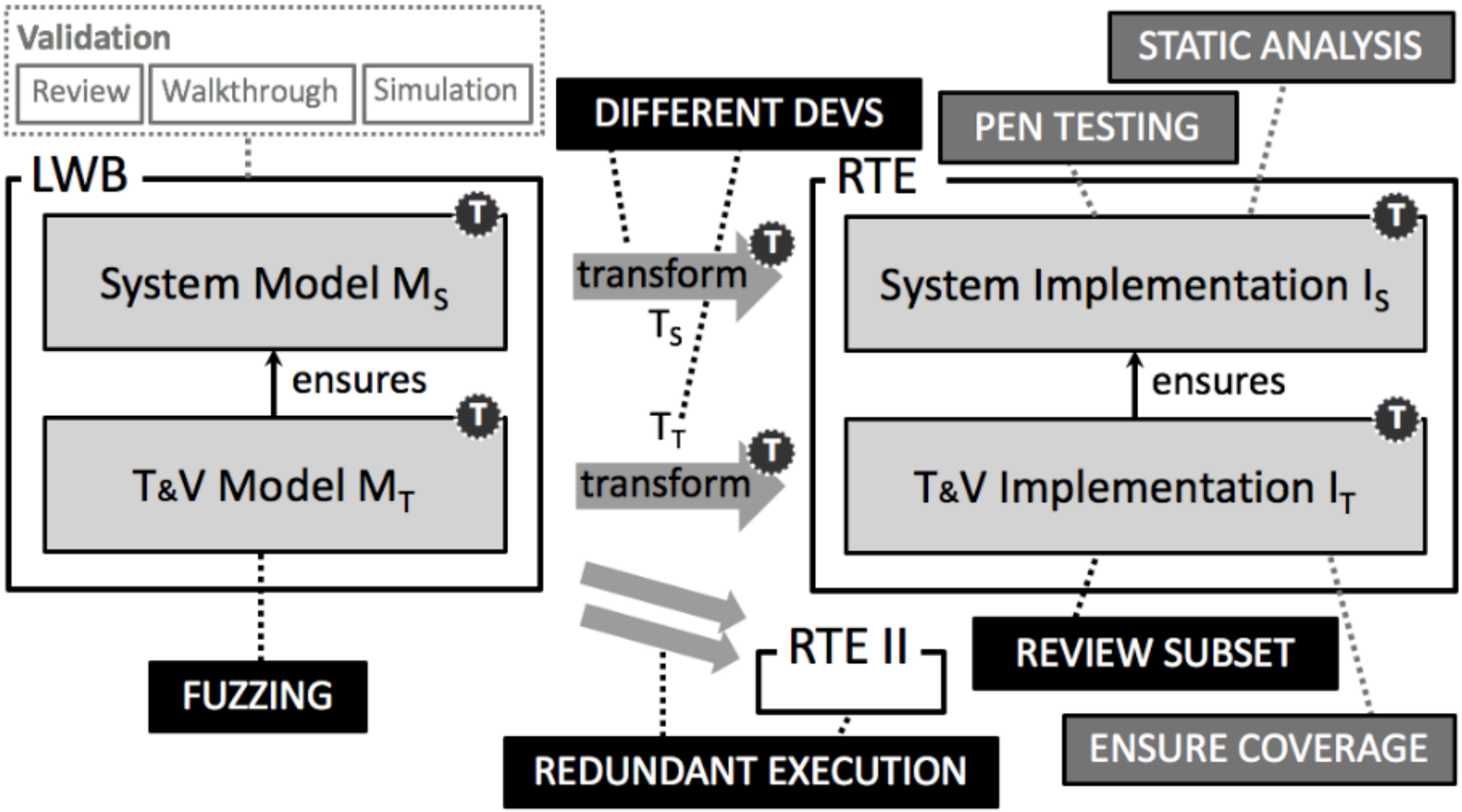


+ Risk Analysis
+ Mitigations

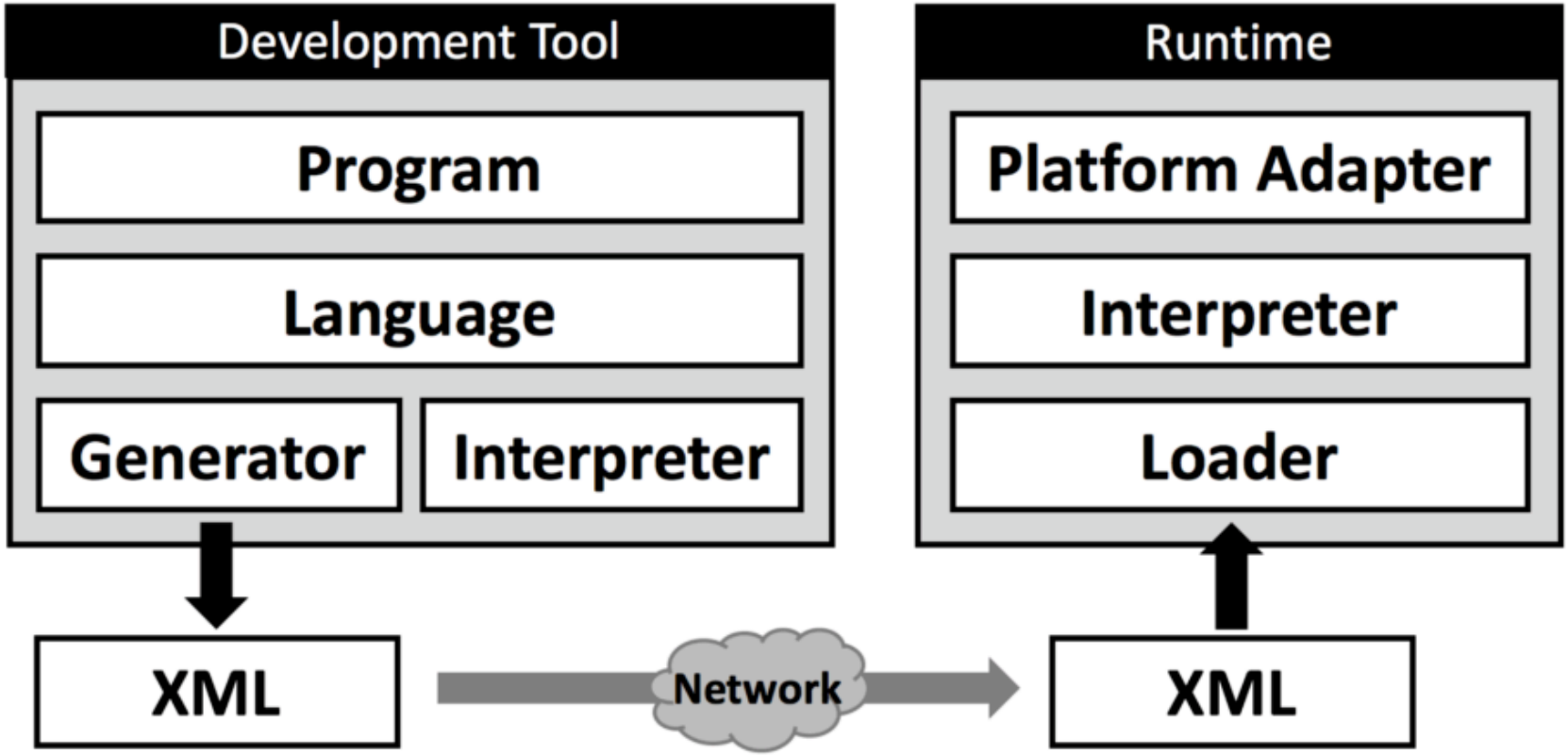
Risk Analysis



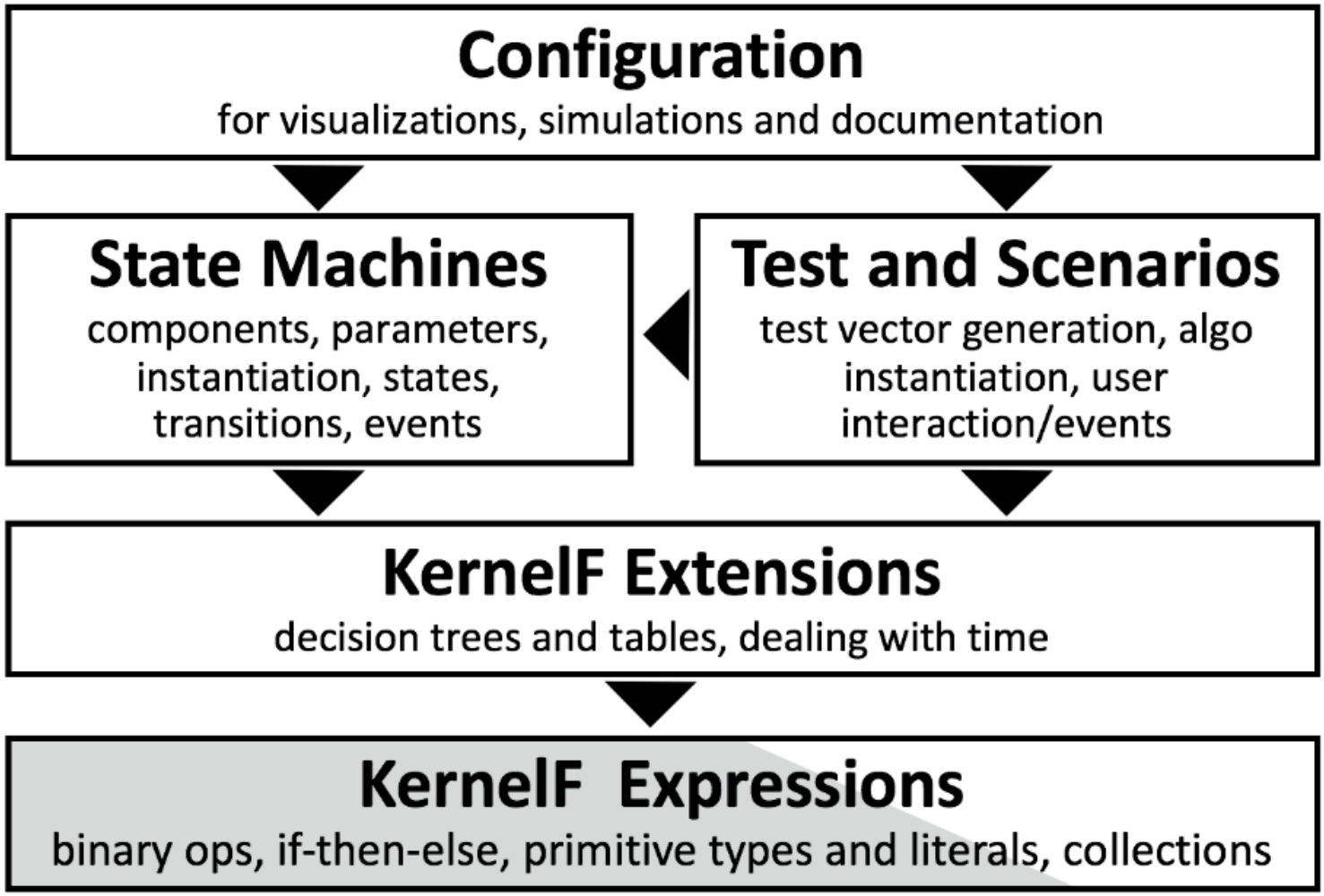
Safe Tool Architecture



Used in Medical System



Used in Medical System



Safer Software through Better Abstractions and Static Analysis

11

Wrap Up

BK

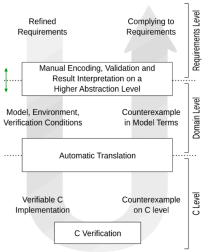
Wrap Up

Languages
Analysis Tools

Domain-Specificity, Ability to detect interesting problems, (Development Effort)

Analyze/Understand (domain-specific)

Extend Language (domain-specific)

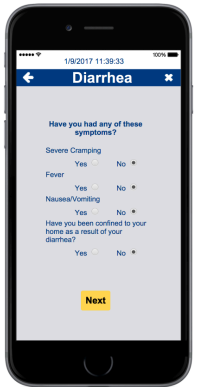


Extend Language (compensate crap)

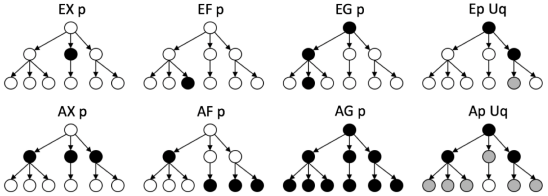
		Events	
		timer(int64 t)	buttonPressed()
States	red	[t - tEnter > 1000] -> green { tEnter = t; }	[ ] -> red { button = true; }
	green	[t - tEnter > 500] -> red { tEnter = t; }	

Advanced Analyses

```
trysequentially {
  validateStep1(data, ...);
  validateStep2(data, ...);
  validateStep3(data, ...);
} on fail (errorcode) {
  handleFailedValidation(data, errorcode, ...);
}
```



High Q Testing



Lint & Co

Use a reasonable language

	a	b	status
0: valid	7	2	no expected value given; actual was 14
1: valid	1	8	no expected value given; actual was 8
2: valid	5	2	[POST] res == a * b
3: valid	7	8	no expected value given; actual was 56
4: valid	5	6	[POST] res == a * b
5: valid	8	0	[PRE] b > 0
6: valid	5	8	[POST] res == a * b

```
#constant LANDING = 100; -> implements FullStop
exported statemachine FlightAnalyzer initial = beforeFlight {
  state beforeFlight {
    on next [tp->alt > 0 m] -> airborne
      exit { points += TAKEOFF; } -> implements PointsForTakeoff
  } state beforeFlight
```

ethereum

- Verify the VM
- Clean up Solidity (eg., reentrancy)
- Integrate Solver/Model Checker into Solidity for `require/assert` clauses
- Higher-level language constructs that express state machines and declarative „contract“ patterns (eg., decisions)
- DSLs for particular app domains

Reading Material

An Overview of Program Analysis using Formal Methods

With a Particular Focus on their Relevance for DSLs

Markus Voelter
with
Tamás Szabó and Björn Engelmann

An addendum to the book *DSL Engineering*

Static program analysis refers to determining properties of programs without executing it, relying on a range of formal methods. While these methods have been around for a long time, over the last couple of years, some of these methods started to scale to solve problems of interesting size.
We have used advanced type systems, abstract interpretation, SMT solving and model checking to answer relevant questions about programs written with various DSLs. In this booklet we introduce the methods, illustrate what we have done with them, and describe how we have integrated the analysis method and existing tools with languages and IDEs.

Note that this booklet documents the authors' experience. This is not a scientific paper. There is no contribution. The aim is to explain and illustrate.

Version 1.0 | July 18, 2017

http://voelter.de/data/books/introToFormalMethodsAndDSLs-1.0.pdf

Automated Testing of DSL Implementations

Experiences from Building mbeddr

Daniel Ratiu · Markus Voelter · Domenik Pavletic

Received: date / Accepted: date

Abstract Domain specific languages promise to improve productivity and quality of software development by providing problem-adequate abstractions to developers. Projectional language workbenches, in turn, allow the definition of modular and extensible domain specific languages, generators and development environments. While recent advances in language engineering have enabled the definition of DSLs and tooling in a modular and cost-effective way, the quality assurance of their implementation is still challenging. In this paper we discuss our work on testing different aspects of the implementation of domain specific languages and associated tools, and present several approaches to increase the automation of language testing. We illustrate these approaches with the Jetbrains MPS language workbench and our experience with testing mbeddr, a set of domain specific languages and tools on top of C tailored to embedded software development. Based on the experience gained from the mbeddr project, we extract generic lessons for practitioners as well as challenges which need more research.

Keywords domain specific languages, testing, quality assurance, automation

http://voelter.de/data/pub/automatedTestingOfDSLs.pdf

Reading Material

Using Language Workbenches and Domain-Specific Languages for Safety-Critical Software Development

Markus Voelter
independent / itemis AG
voelter@acm.org

Bernd Kolb, Klaus Birken
itemis AG
{kolb|birken}@itemis.de

Federico Tomassetti
independent
federico@tomassetti.me

Patrick Alff, Laurent Wiart
Voluntis
{Patrick.Alff|Laurent.Wiart}@voluntis.com

Andreas Wortmann
OHB System AG
andreas.wortmann@ohb.de

Arne Nordmann
Bosch Corporate Research
arne.nordmann@de.bosch.com

Language workbenches allow developers to create, integrate and efficiently use domain-specific languages, typically by generating programming language code from models expressed with domain-specific languages. This leads to increased productivity and higher quality. However, in safety-/mission-critical environments, such generated code may not be considered trustworthy, because of the lack of trust in how the code was obtained. This makes it harder to use language workbenches in such an environment. In this paper, we demonstrate an approach to use such tools in critical environments. We argue that models created with domain-specific languages are easier to validate, and that the additional risk resulting from the transformation to code can be mitigated by an suitably designed transformation and verification architecture. We validate the approach with an industrial case study from the medical domain. We also discuss the degree to which the approach is appropriate for critical software in space, automotive and robotics systems.

Available online soon – contact me!