Stream Runtime Verification

Martin Leucker

Together with the whole TeSSLa Team
(Lukas Convent, Hannes Kallwies, Martin Sachenbacher, Malte Schmitz, Daniel Thoma, Volker Stolz, Cesar Sanchez, and many others)
Plan

- Stream Runtime Verification
- LOLA
- TeSSLa
  - Language
  - Eco-System
- Control
  - Cyber-Physical Systems
  - Controllers
  - TeSSLa/ROS bridge
Motivation
Streams
Streams

Concurrency/Distribution
Streams
Streams

Time? Synchrony/Ticks
Equational specifications, data, time, concurrency

LOLA

[D’Angelo et al.]

\[
\begin{align*}
 s_1 &= \text{true} \\
 s_2 &= t_3 \\
 s_3 &= t_1 \lor (t_3 \leq 1) \\
 s_4 &= ((t_3)^2 + 7) \mod 15 \\
 s_5 &= \text{ite}(s_3, s_4, s_4 + 1) \\
 s_6 &= \text{ite}(t_1, t_3 \leq s_4, \neg s_3) \\
 s_7 &= t_1[+1, \text{false}] \\
 s_8 &= t_1[-1, \text{true}] \\
 s_9 &= s_9[-1, 0] + (t_3 \mod 2) \\
 s_{10} &= t_2 \lor (t_1 \land s_{10}[1, \text{true}])
\end{align*}
\]
Example

\[
\begin{align*}
    s_1 &= \text{true} \\
    s_2 &= t_3 \\
    s_3 &= t_1 \lor (t_3 \leq 1) \\
    s_4 &= ((t_3)^2 + 7) \mod 15 \\
    s_5 &= \text{ite}(s_3, s_4, s_4 + 1) \\
    s_6 &= \text{ite}(t_1, t_3 \leq s_4, \neg s_3) \\
    s_7 &= t_1[+1, \text{false}] \\
    s_8 &= t_1[-1, \text{true}] \\
    s_9 &= s_9[-1, 0] + (t_3 \mod 2) \\
    s_{10} &= t_2 \lor (t_1 \land s_{10}[1, \text{true}])
\end{align*}
\]
Example

\[
\begin{align*}
  s_1 &= \text{true} \\
  s_2 &= t_3 \\
  s_3 &= t_1 \lor (t_3 \leq 1) \\
  s_4 &= ((t_3)^2 + 7) \mod 15 \\
  s_5 &= \text{ite}(s_3, s_4, s_4 + 1) \\
  s_6 &= \text{ite}(t_1, t_3 \leq s_4, \neg s_3) \\
  s_7 &= t_1[+1, \text{false}] \\
  s_8 &= t_1[-1, \text{true}] \\
  s_9 &= s_9[-1, 0] + (t_3 \mod 2) \\
  s_{10} &= t_2 \lor (t_1 \land s_{10}[1, \text{true}])
\end{align*}
\]
Example

\[
\begin{align*}
s_1 & = \text{true} \\
s_2 & = t_3 \\
s_3 & = t_1 \lor (t_3 \leq 1) \\
s_4 & = ( (t_3)^2 + 7 ) \mod 15 \\
s_5 & = \text{ite} ( s_3, s_4, s_4 + 1 ) \\
s_6 & = \text{ite} ( t_1, t_3 \leq s_4, \neg s_3 ) \\
s_7 & = t_1[+1, \text{false}] \\
s_8 & = t_1[-1, \text{true}] \\
s_9 & = s_9[-1,0] + ( t_3 \mod 2 ) \\
s_{10} & = t_2 \lor ( t_1 \land s_{10}[1, \text{true}] )
\end{align*}
\]
Example – Reaching the end of the Trace

\begin{align*}
    &s_1 = \text{true} \\
    &s_2 = t_3 \\
    &s_3 = t_1 \lor (t_3 \leq 1) \\
    &s_4 = ((t_3)^2 + 7) \mod 15 \\
    &s_5 = \text{ite}(s_3, s_4, s_4 + 1) \\
    &s_6 = \text{ite}(t_1, t_3 \leq s_4, \neg s_3) \\
    &s_7 = t_1[+1, \text{false}] \\
    &s_8 = t_1[-1, \text{true}] \\
    &s_9 = s_9[-1, 0] + (t_3 \mod 2) \\
    &s_{10} = t_2 \lor (t_1 \land s_{10}[1, \text{true}])
\end{align*}
Defining new Streams
Defining new Streams

Runtime Verification as Stream Transformation
Streams

Time? Synchrony/Ticks
Streams

Time triggered systems

Time? Synchrony/Ticks
Streams

Time triggered systems

Event-triggered

Time? Synchrony/Ticks
Tessla’s Streams

Time? Events
Tessla’s Streams

Time? Events
Tessla’s Streams

Time? Events
Tessla’s Streams

Time? Events
Tessla’s Streams

Time? Events
Tessla’s Streams

Time? Events
Tessla’s Streams

Time? Events
Tessla’s Streams

Time? Events
Streams of Programs - After Discretization

Values
e.g., of a program variable $x$

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>6</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
</table>

Program events
e.g., call to `my_func()`
Streams

Values

- e.g., of a program variable $x$

| 5 | 6 | 1 | 2 |

Program events

- e.g., call to `my_func()`
Defining new Streams

- Observations (Input streams)
- Derived streams (definable)

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>irq 4</td>
<td>998, 42, 2012, 1280, 10, 1404</td>
</tr>
<tr>
<td>(with value) f</td>
<td>17, 98, 0, 23</td>
</tr>
</tbody>
</table>

- \( x > 1023 \)
- \( \text{changeOf}(x) \)
- \( f \text{ inPast} \leq 10\text{ms} \)
Defining new Streams

Runtime Verification as Stream Transformation

Observations (Input streams)

Derived streams (definable)

Event (with value) f

\[ x > 1023 \]

\[ \text{changeOf}(x) \]

\[ f \text{ inPast} \leq 10\text{ms} \]

Time

Value

x

1023
Runtime Verification with Uncertainties
Lola Example

In \( ld \): Real \[3 \rightarrow 4 \rightarrow 5 \rightarrow 7 \]

Def \( acc := acc[-1|0] + ld[now] - ld[-3|0] \) \[3 \rightarrow 7 \rightarrow 12 \rightarrow 16 \]

Def \( ok := (acc[now] \leq 15) \) \( tt \rightarrow tt \rightarrow tt \rightarrow ff \)

Three basic LOLA stream expressions:

- Constant streams
- Offset operators \( s[o|c] \)
  \( \Rightarrow \) We restrict our self to the past fragment here (i.e. \( o \leq 0 \))
- Function applications
Using Abstract Domains

In 2019 Leucker et al. presented approach with intervals as abstract domain.

In $ld$: Real $\rightarrow [1,5] \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow$

Def $acc := acc[-1|0] + ld[now] - ld[-3|0] \rightarrow [1,5] \rightarrow [5,9] \rightarrow [10,14] \rightarrow [12,20] \rightarrow$

Def $ok := (acc[now] \leq 15) \rightarrow tt \rightarrow tt \rightarrow tt \rightarrow ? \rightarrow$

- Approach is sound, but not perfect.
- Handling of complex assumptions in general not possible.
Symbolic Evaluation

**Idea:** Use symbolic formulas for representation of unknown values and additional logical constraints (e.g. assumptions).

⇒ Use SMT solver for queries on possible values.

In $ld$: Real

$$ld^0 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow$$

Def $acc := acc[-1|0] + ld[now] - ld[-3|0]$

$$ld^0 \rightarrow ld^0+4 \rightarrow ld^0+9 \rightarrow 16 \rightarrow$$

Def $ok := (acc[now] \leq 15)$$

$$tt \rightarrow tt \rightarrow tt \rightarrow ff \rightarrow$$

Additional constraints: $\{1 \leq ld^0 \leq 5\}$

- Approach in principle perfect.
- Assumptions can be added as propositions to constraint set.
TeSSLa
TeSSLa

- Temporal
- Stream-based
- Specification
- Language

- Specifying the (expected) behavior of a system’s execution
Language - Overview

Core Language

- Type System
- Macro System
- Module System
- Meta Data
Design Goals – Core Language

• Declarative style: Specification rather than implementation
• Modularity: Allowing abstractions based on few primitives
  (6 operators: unit, nil, lift, last, delay, time)
• Time as first-class citizen
• Abstractions for both events and signals
• Recursion to reason about past
• Implementable with limited memory
  (For a restricted fragment)
TeSSLa by example

\[
\begin{array}{c}
\begin{array}{c}
 a \\
 b \\
 c \\
\end{array}
\begin{array}{c}
 2 \\
 1 \\
 3 \\
\end{array}
\begin{array}{c}
 4 \\
 3 \\
 5 \\
\end{array}
\begin{array}{c}
 6 \\
 5 \\
 7 \\
\end{array}
\begin{array}{c}
 6 \\
 5 \\
 7 \\
\end{array}
\end{array}
\]

\texttt{def c := a + b}

\[
\begin{array}{c}
\begin{array}{c}
 x \\
 r \\
 c \\
\end{array}
\begin{array}{c}
 0 \\
 1 \\
 2 \\
\end{array}
\begin{array}{c}
 1 \\
 2 \\
 3 \\
\end{array}
\begin{array}{c}
 0 \\
 1 \\
 1 \\
\end{array}
\begin{array}{c}
 1 \\
 2 \\
\end{array}
\end{array}
\]

\texttt{def c := \texttt{eventCount}(x, \texttt{reset} = r)}
TeSSLa operators: Last

- Needed to define properties over sequences of events.
- Last allows to refer to the values of events on one stream that occurred strictly before the events on another stream.

Read last(x,y) as last of x when event on y
TeSSLa operators: Time

- Provides access to the timestamps of events
- Produces events carrying their timestamps as data value
- Hence all operators for data values can be applied to timestamps.
Create Events

\[
\text{in write: Events[\text{Unit}]}
\]

\[
\text{def timeout := const(5, write)}
\]

\[
\text{def error := delay(timeout, write)}
\]

\[
\text{out error}
\]
Data types in TeSSLa

- TeSSLa strongly typed, generic types
- TeSSLa agnostically wrt any time or data domain
- Different data structures can be used to represent time and data
- Monitoring in hardware: atomic data types, e.g. int or float
- Monitoring in software: complex data structures like lists, trees and maps
Macros in TeSSLa

- Few primitive operators
- Readable specifications via Macros
- TeSSLa Standard Library for common useful stuff
- Domain specific libraries for application areas/domains (anticipated)
  - Timex/Autosar library
  - PastLTL
  - Petri nets (under development)
Macros in TeSSLa: EventCount

```python
# Count the number of events on `values`.
def eventCount[A,B](values: Events[A]) := {
def count: Events[Int] := merge(
    # increment counter
    last(count, values) + 1
, 0)
count
}
```
Modules in TeSSLa

- Sets of Macros can be grouped to modules/libraries
- TeSSLa Standard Library for common useful stuff
- Domain specific libraries for application areas/domains (anticipated)
  - Timex/Autosar library
  - PastLTL
  - Petri nets (under development)
Meta Data / Annotations

- TeSSLa allows annotations similar like @interface in Java
- Several categories for annotations
  - Documentation
  - Correspondence to Source Code (C-Code)
  - Graphical presentation of streams / dashboard support
  - Directives for Example Generator
  - Directives for bridging to frameworks (ROS)
TeSSLa by Example

```python
# Inputs
@InstFunctionCall("read_brake_sensor")
in read_brake_sensor: Events[Unit]
@InstFunctionCall("activate_brakes")
in activate_brakes: Events[Unit]

# Trace Processing
def latency = measureLatency(read_brake_sensor, activate_brakes)
def error = latency > 4ms
def high = filter(latency, error) - 4ms
def is_critical = count(high) > 10
def critical = filter(high, is_critical)

# Output
@VisDots out high
@VisEvents out critical

# Macro
def measureLatency[A, B](a: Events[A], b: Events[B]) =
  time(b) - last(time(a), b)
```

Input decl. & annotations
Monitoring property
Output decl. & annotations
Macro definitions
TeSSLa by Example

# Inputs
@InstFunctionCall("read_brake_sensor")
in read_brake_sensor: Events[Unit]
@InstFunctionCall("activate_brakes")
in activate_brakes: Events[Unit]

# Trace Processing
def latency = measureLatency(read_brake_sensor, activate_brakes)

def error = latency > 4ms

def high = filter(latency, error) - 4ms

def is_critical = count(high) > 10

def critical = filter(high, is_critical)

# Output
@VisDots out high
@VisEvents out critical

# Macro
def measureLatency[A, B](a: Events[A], b: Events[B]) =
  time(b) - last(time(a), b)
TeSSLa by Example

# Inputs
@InstFunctionCall("read_brake_sensor")
in read_brake_sensor: Events[Unit]
@InstFunctionCall("activate_brakes")
in activate_brakes: Events[Unit]

# Trace Processing
def latency = measureLatency(read_brake_sensor, activate_brakes)
def error = latency > 4ms
def high = filter(latency, error) - 4ms
def is_critical = count(high) > 10
def critical = filter(high, is_critical)

# Output
@VisDots out high
@VisEvents out critical

# Macro
def measureLatency[A, B](a: Events[A], b: Events[B]) =
time(b) - last(time(a), b)
TeSSLa by Example

# Inputs
@InstFunctionCall("read_brake_sensor")
in read_brake_sensor: Events[Unit]
@InstFunctionCall("activate_brakes")
in activate_brakes: Events[Unit]

# Trace Processing
def latency = measureLatency(read_brake_sensor, activate_brakes)
def error = latency > 4ms
def high = filter(latency, error) - 4ms
def is_critical = count(high) > 10
def critical = filter(high, is_critical)

# Output
@VisDots out high
@VisEvents out critical

# Macro
def measureLatency[A, B] (a: Events[A], b: Events[B]) =
    time(b) - last(time(a), b)
TeSSLa compilers
Observation/Instrumentation

- Instrumenter for C code integrated in compiler
- Accemic’s CEDARtools for non-intrusive hardware monitoring
- Connection to other instrumentation tools via generic annotation system
Supporting Web IDE
Supporting Online Documentation

constIf

\[ \text{constIf}[T](\text{value}: T, \text{condition}: \text{Events}[\text{Bool}]): \text{Events}[T] \]

Produce an event with the given value every time that the condition is met.

Usage example:

```python
in  condition: \text{Events}[\text{Bool}]
def  \text{result} = \text{constIf}(42, \text{condition})
out \text{result}
```

Trace example:

<table>
<thead>
<tr>
<th>condition</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>42</td>
</tr>
<tr>
<td>1</td>
<td>42</td>
</tr>
<tr>
<td>0</td>
<td>42</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>42</td>
</tr>
<tr>
<td>1</td>
<td>42</td>
</tr>
</tbody>
</table>

Source

```python
def constIf[T](value: T, condition: Events[Bool]): Events[T] =
filter(const(value, condition), condition)
```

count

\[ \text{count}[T](x: \text{Events}[T]): \text{Events}[\text{Int}] \]

Count the number of events on \(x\). Provides for every input event an output event whose value is the number of events seen so far. See \text{resetcount} for a counting macro with an external reset.
TeSSLa Ecosystem

• User Libraries
  • Macro system allows definition of application-specific libraries
  • E.g. AUTOSAR Timex, Past LTL libraries...

• Tutorials
  • Extensive tutorials about the usage of the TeSSLa language and tools.

• Open-Source availability
  • Free availability of most parts of the tool chain.
  • Community-driven project.
TeSSLa for professional usage

- Clear definition of license
- Separation of
  - Language,
  - Compilers, and
  - Tools
- Language specification
  - TeSSLa and TeSSLa Core
- Reference Compiler (Interpreter)
Resources

• TeSSLa Website:
  https://www.tessla.io/

• TeSSLa Playground:
  https://play.tessla.io/

• TeSSLa Sourcecode:
  https://git.tessla.io/

• Contact:
  info@tessla.io
A Convenient Language for Specification and Verification of Your System
TeSSLa Installation and First-Steps
Installation – TeSSLa Bundle

- contains a compiler, interpreter and other useful tools for executing TeSSLa specifications
- written in Scala and available as a single JAR archive.
- The TeSSLa bundle is licensed under Apache 2.0 license. Run java -jar tessla.jar -h for information on the usage of the TeSSLa command line tool.

Logging Library

- For instrumenting C-Code

https://www.tessla.io/logging.zip
TeSSLa libraries

Further libraries

https://www.tessla.io/usrLibs/overview/
A simple specification

• specification.tessla

```python
in x: Events[Int]
in y: Events[Int]

def diff = sum(x) - sum(y)

liftable
def abs(x: Int) = if x < 0 then -x else x
def tooBig = abs(diff) >= 10

out diff
out tooBig
```
Input trace

- trace.input

10: x = 2
17: x = 1
19: y = 4
37: x = 7
45: x = 6
78: y = 9
98: x = 2
In the playground

https://play.tessla.io
Playground
Running

- java -jar tessla.jar interpreter
  specification.tessla trace.input

0: tooBig = false
0: diff = 0
10: tooBig = false
10: diff = 2
17: tooBig = false
17: diff = 3
19: tooBig = false
19: diff = -1

37: tooBig = false
37: diff = 6
45: tooBig = true
45: diff = 12
78: tooBig = false
78: diff = 3
98: tooBig = false
98: diff = 5
TeSSLa Scala/Rust Compiler

- **Scala compiler**
  - allows compilation to Scala code or a JAR file executable on the Java JVM.
    
    ```
    java -jar tessla.jar compile-scala -j monitor.jar specification.tessla
    ```
  - creates an executable Jar-File **monitor.jar** which receives inputs and produces outputs via stdio in the same format as the interpreter

- **Rust compiler**
  
  ```
  java -jar tessla.jar compile-rust -b monitor specification.tessla
  ```
  - creates an executable **monitor** which receives inputs and produces outputs via stdio in the same format as the interpreter
Instrumenting C-Code

• Instrument the C source code using the observation annotations defined in the TeSSLa specification:

  java -jar tessla.jar instrumenter spec.tessla main.c
  /usr/lib/gcc/x86_64-linux-gnu/9/include/

• Instrumentation is done on the LLVM level and specific setup for your machine is needed
For convenience

• As long as it works

docker run -v $(pwd)/wd -w /wd --rm registry.isp.uni-luebeck.de/tessla/tessla-docker:2.0.0 rv spec.tessla main.c
TeSSLa Language in Detail
Let’s work through the tutorial

https://www.tessla.io/tutorial/
RV with TeSSLa
main.c

void foo() {
    int x = 42;
}

int main() {
    for (int i = 0; i < 5; i++) {
        foo();
    }
    return 0;
}

spec.tessla

@InstFunctionCall("foo")
in foo: Events[Unit]
out foo

def num := count(foo)
out num
Explore

- Instrument the C source
  
  ```bash
  java -jar tessla.jar instrumenter spec.tessla main.c
  ```

- Compile the instrumented C code
  
  ```bash
  gcc main.c.instrumented.c -llogging -pthread -ldl -o main
  ```

- Execute the compiled program, creating the file trace.log
  
  ```bash
  ./main
  ```

- Monitor the trace
  
  ```bash
  java -jar tessla.jar interpreter --base-time 1ns spec.tessla trace.log
  ```

- Alternatively
  
  ```bash
  docker run -v $(pwd):/wd -w /wd --rm registry.isp.uni-luebeck.de/tessla/tessla-docker:2.0.0 rv spec.tessla main.c
  ```
Measuring a Function's Runtime

```c
#include <stdlib.h>
#include <unistd.h>

void compute() {
    int duration = 40000;
    duration += (rand() % 10) * 1000;
    usleep(duration);
}

int main() {
    for (int i = 0; i < 10; i++) {
        compute();
    }
}
```

```chef
@InstFunctionCall("compute")
in  call:  Events[Unit]

@InstFunctionReturn("compute")
in  ret:  Events[Unit]

def duration := runtime(call, ret)
out duration

out maximum(duration) as max
out average(duration) as avg
```
Checking Correctness of Values

```c
#include <stdio.h>
#include <unistd.h>

int add(int a, int b) {
    return a + b;
}

int main() {
    printf("%i\n", add(2,3));
    printf("%i\n", add(17,4));
    printf("%i\n", add(200000000,100000000));
}

@InstFunctionCallArg("add", 0)
in a: Events[Int]
@InstFunctionCallArg("add", 1)
in b: Events[Int]
@InstFunctionReturnValue("add")
in r: Events[Int]

def should = last(a + b, r)
def ok = r == should

out a
out b
out r
out should
out ok
```
Multiple Threads

```c
#include <pthread.h>

void foo() {}

void *task () {
    foo();
    foo();
    foo();
    return NULL;
}

int main ()
{
    pthread_t t1, t2;
    pthread_create(&t1, NULL, &task, NULL);
    pthread_create(&t2, NULL, &task, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    return 0;
}
```

```c
@InstFunctionCall("foo")
in foo: Events[Unit]

@ThreadId
in tid: Events[Int]

out foo
out tid
```
including <pthread.h>
#include <unistd.h>

int shared_memory[4] = {0};
pthread_mutex_t locks[4] = {
    PTHREAD_MUTEX_INITIALIZER, PTHREAD_MUTEX_INITIALIZER,
    PTHREAD_MUTEX_INITIALIZER, PTHREAD_MUTEX_INITIALIZER,
};

void use(int index) {
    shared_memory[index]++;
}

void *task1 () {
    for (int i = 0; i < 4; i++) {
        pthread_mutex_lock(&locks[i]);
        use(i);
        pthread_mutex_unlock(&locks[i]);
    }
    return NULL;
}

void *task2 () {
    for (int i = 3; i >= 0; i--) {
        pthread_mutex_lock(&locks[i]);
        use(i);
        pthread_mutex_unlock(&locks[i]);
    }
    return NULL;
}

int main () {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, &task1, NULL);
    pthread_create(&t2, NULL, &task2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
Checking Correct Locking (2)

```haskell
@InstFunctionCallArg("pthread_mutex_lock", 0)
in lock: Events[Int]

@InstFunctionCallArg("pthread_mutex_unlock", 0)
in release: Events[Int]

@InstFunctionCallArg("use", 0)
in access: Events[Int]

@ThreadId
in tid: Events[Int]

def locksOfThread = {
def oldMap = last(map, access)
def oldLocks = Map.getOrElse(oldMap, tid, Set.empty[Int])
def map: Events[Map[Int, Set[Int]]] = merge3(
on(lock, Map.add(oldMap, tid, Set.add(oldLocks, lock))),
on(release, Map.add(oldMap, tid, Set.remove(oldLocks, release))),
Map.empty[Int, Set[Int]])
map
}

def locksForResource = {
def old = last(map, access)
def currentLocks = Map.get(locksOfThread, tid)
def map: Events[Map[Int, Set[Int]]] = merge(
on(access, Map.add(old, access, Set.intersection(
currentLocks,
   Map.getOrElse(old, access, currentLocks)))),
   Map.empty[Int, Set[Int]])
map
}

def error = unitIf(Set.size(Map.get(locksForResource, access)) == 0)
out error
```
Cyber-Physical Systems
Cyber-Physical System

- Communicating hybrid systems
- Communicating embedded systems interacting with the physical world

- Discrete Math, Events, Propositions
- Continuous Math, Signals
Damped Harmonic Oscillator

\[ m \cdot y'' = -D \cdot y - d \cdot y' \]
Solving of ODE – Numerical Approximations

• Euler’s method
Solving of ODE – A Variety of Methods

By Svchbderivative work: tobi (talk) - RK Verfahren, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=32717385
ODEs in TeSSLa
Damped Harmonic Oscillator

\[ m \cdot y'' = -D \cdot y - d \cdot y' \]
The Spring Example

1 in sensor: Events[Float]
2
3 def m: Float = 0.2      # kg
4 def D: Float = 2.6      # N/m
5 def d: Float = 0.15     # kg/s
6 def y''(t: Float, y: Float, y': Float): Float =
7     -D / m * y - d / m * y'
8 def y_0 = 0.2        # m
9 def y'_0 = 0.0       # m/s
10
11 def approx:     Events[(Float, Float)] = rk4(y'', y_0, y'_0)
12 def approxY:    Events[Float] = approx._1
13 def approxY':   Events[Float] = approx._2
14 def alarm = |sensor - approxY| > ε
Plot of the Damped Spring
Control
Runtime Verification
Runtime Verification

- Partial Verification
Runtime Verification

- Partial Verification
- Testing Temporal Assertions
Runtime Verification

- Partial Verification
- Testing Temporal Assertions
- Test Cases as Input Sequences checked by Monitors
Runtime Verification

- Partial Verification
- Testing Temporal Assertions
- Test Cases as Input Sequences checked by Monitors
- Debugging
Runtime Verification

- Partial Verification
- Testing Temporal Assertions
- Test Cases as Input Sequences checked by Monitors
- Debugging
- Control?
Control from an RV Point of View
Control from an RV Point of View

- Monitor Output as Feedback/Intervention to System
Control from an RV Point of View

- Monitor Output as Feedback/Intervention to System
Control from an RV Point of View

- Monitor Output as Feedback/ Intervention to System
- Monitor has to give more specific Output
Control from an RV Point of View

- Monitor Output as Feedback/Intervention to System
- Monitor has to give more specific Output
- Here: Monitor actually computes control values
Self-Healing System (FDIR with RV)
Self-Healing System (FDIR with RV)
Control

Reference → + Measured error → Controller → System input → System → System output

Measured output → Sensor

By Orzetto - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=5000019
Control

Closed-Loop Controller - Feedback

Reference \(\rightarrow\) Measured error \(\rightarrow\) Controller \(\rightarrow\) System input \(\rightarrow\) System \(\rightarrow\) System output

Reference \(\rightarrow\) Measured output \(\rightarrow\) Sensor

By Orzetto - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=5000019
Control

Closed-Loop Controller - Feedback

By Orzetto - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=5000019
Control

Closed-Loop Controller - Feedback

Reference → Controller → System → Sensor → Drag Up → Monitor

+ Measured error

System input

System output

Measured output

Logging

By Orzetto - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=5000019
PID-Controller

Controller Combinations

**P**
Proportional controller to reduce the transient period.
*Changes the magnitude only.*

**I**
Integral controller to reduce the time invariant error
*Lags the output phase.*

**D**
Derivative controller to minimize the transient errors like overshoot, oscillatory response.
*Leads the output phase.*

**PI**
Reduces rise time and steady state errors
*Changes the magnitude as well as lags the output.*

**PD**
Reduces rise time and transient errors such as overshoot, oscillations in output.
*Changes both the magnitude as well as adds a leading phase to the output.*

**PID**
General case of a controller. Can be used to control the magnitude and lead/ lag phase problems.
*Changes the magnitude and can add positive or negative phase to the output as per the requirements.*

Code of Controller in TeSSLa

• See tessla.io
Controlling Robots
TeSSLa/ROS Bridge

include "TesslaROSBridge.tessla"

@RosSubscription("/reduced_scan_to_tessla", "int64", "10")
in scan: Events[Int]

# Stop if there are short rays detected

def stop = scan < 20

@RosPublisher("/result_from_tessla_to_ros", "bool", "10")
out stop
Example
Example
Example
Conclusions
Conclusions

- Stream-based Runtime Verification makes sense
- TeSSLa one approach in this setting
- Supports handling of data

- Monitoring CPS makes sense
- Controlling using RV techniques makes sense
- Separation of concerns
Future Work

• Controller module in TeSSLa?
• More concrete examples?
• Gain more experiences?
• Programming (safety aspects) of robots?
• Better use Modellica and FMUs?
• Add continuous functions symbolically to perform algebraic simplifications?