UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

isp

TeSSLa – An Ecosystem For Runtime Verification

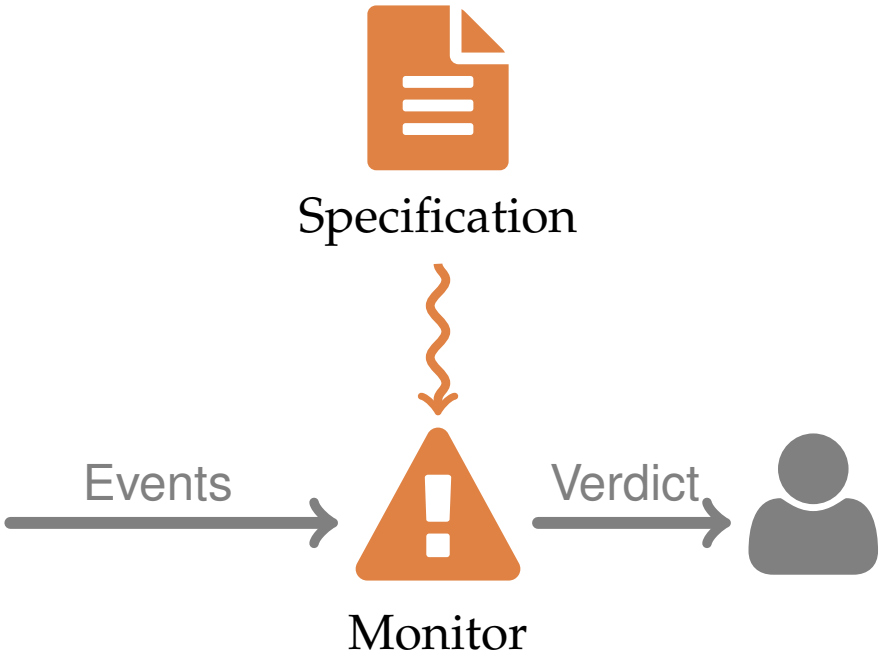
Hannes Kallwies[1] Martin Leucker[1] Malte Schmitz[1] Albert Schulz[2]
Daniel Thoma[1] Alexander Weiss[2]

[1] Institute for Software Engineering and Programming Languages,
University of Lübeck, Lübeck, Germany

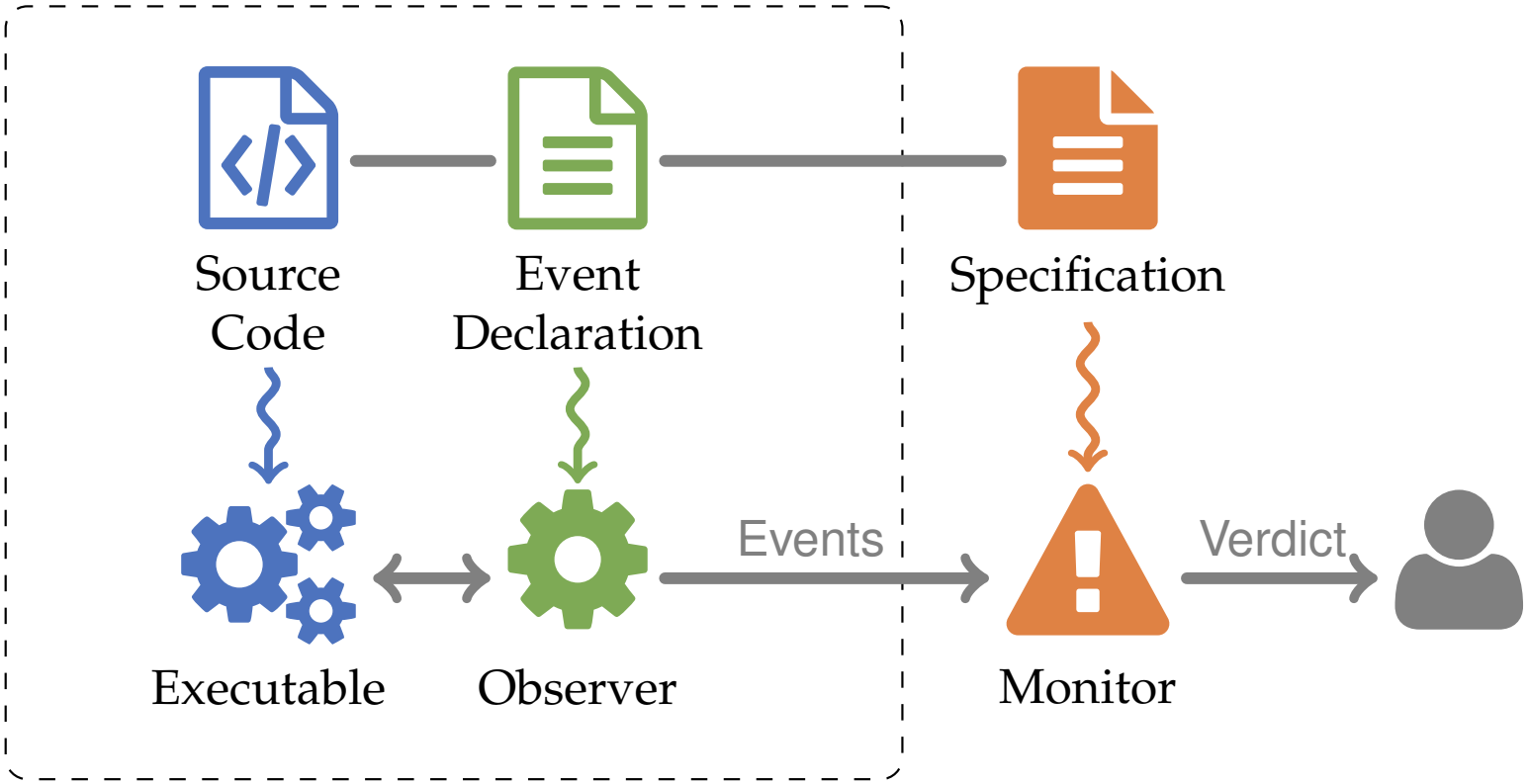
[2] Accemic Technologies GmbH, Kiefersfelden, Germany

22nd International Conference on Runtime Verification, September 2022

Runtime Verification Process



Runtime Verification Process



TeSSLa language features

TeSSLa language features

- **Stream-based** specification language with **functional part**

TeSSLa language features

- **Stream-based** specification language with **functional part**
- Based on six basic stream operators (**core operators**) (`unit`, `nil`, `lift`, `last`, `delay`, `time`)

TeSSLa language features

- **Stream-based** specification language with **functional part**
- Based on six basic stream operators (**core operators**)
  (`unit`, `nil`, `lift`, `last`, `delay`, `time`)
- **Macro system** for definition of user defined stream operators
  Lots of predefined macros in the stdlib

TeSSLa language features

- **Stream-based** specification language with **functional part**
- Based on six basic stream operators (**core operators**)
  (`unit`, `nil`, `lift`, `last`, `delay`, `time`)
- **Macro system** for definition of user defined stream operators
  Lots of predefined macros in the stdlib
- **Type system** with support of generic types

TeSSLa language features

- **Stream-based** specification language with **functional part**
- Based on six basic stream operators (**core operators**)
  (`unit`, `nil`, `lift`, `last`, `delay`, `time`)
- **Macro system** for definition of user defined stream operators
  Lots of predefined macros in the stdlib
- **Type system** with support of generic types
- **Annotation system** for steering of the RV tool chain

TeSSLa - Example Specification

```
# 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 - Example Specification

```
# 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

TeSSLa - Example Specification

```
# 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

TeSSLa - Example Specification

```
# 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

TeSSLa - Example Specification

```
# 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)
```

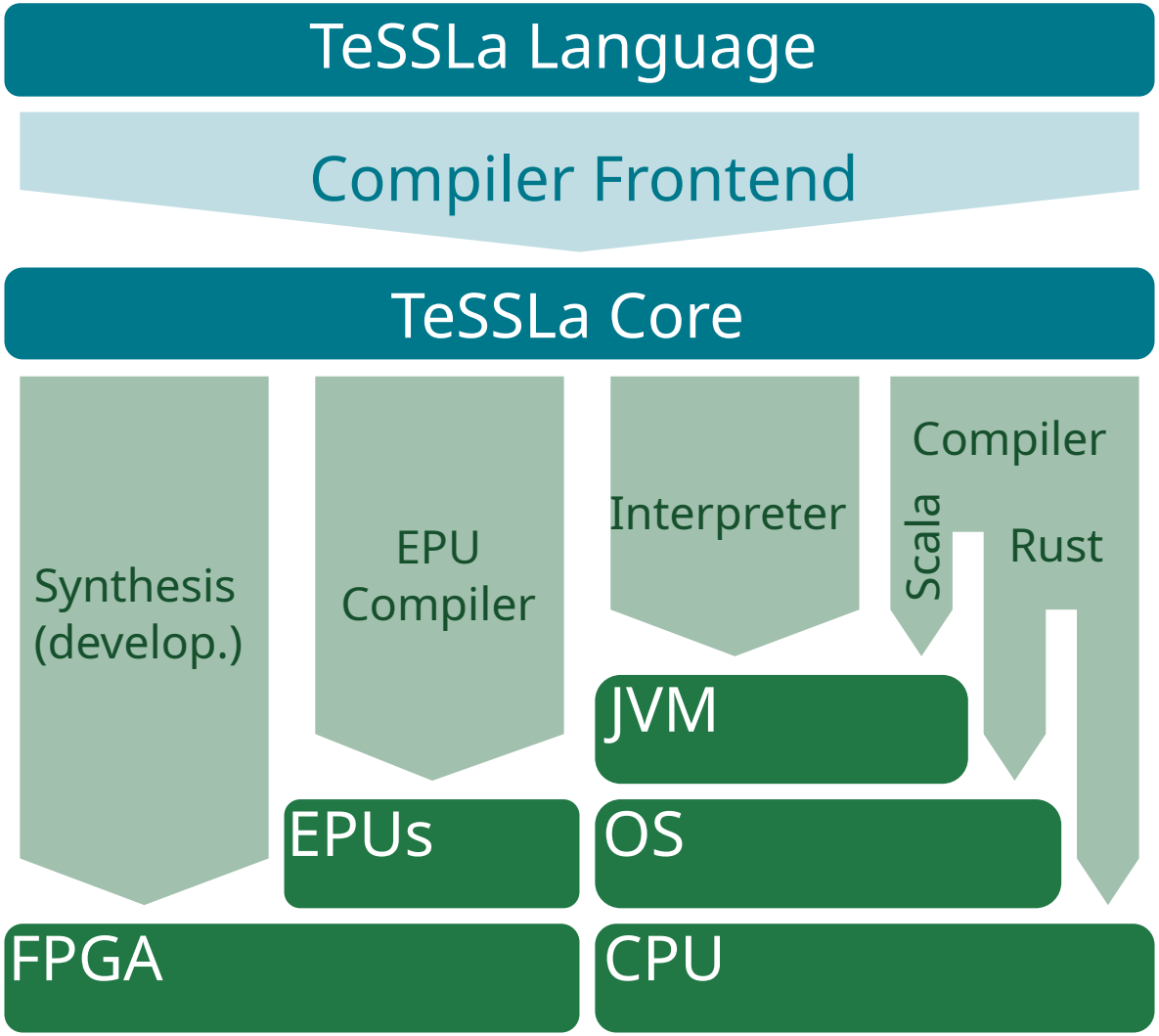
Input decl. & annotations

Monitoring property

Output decl. & annotations

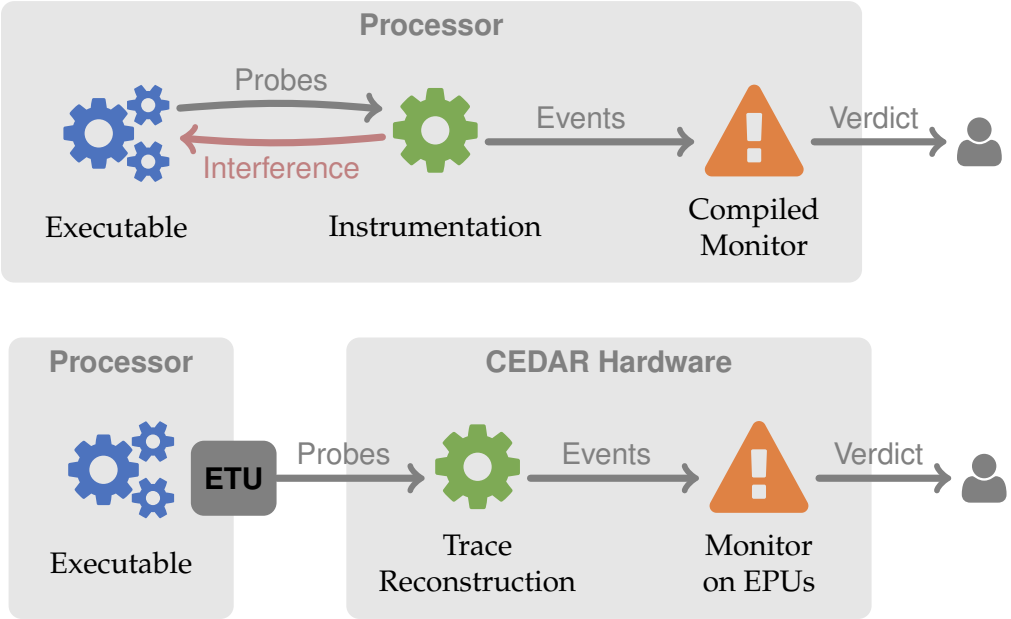
Macro definitions

TeSSLa Backends

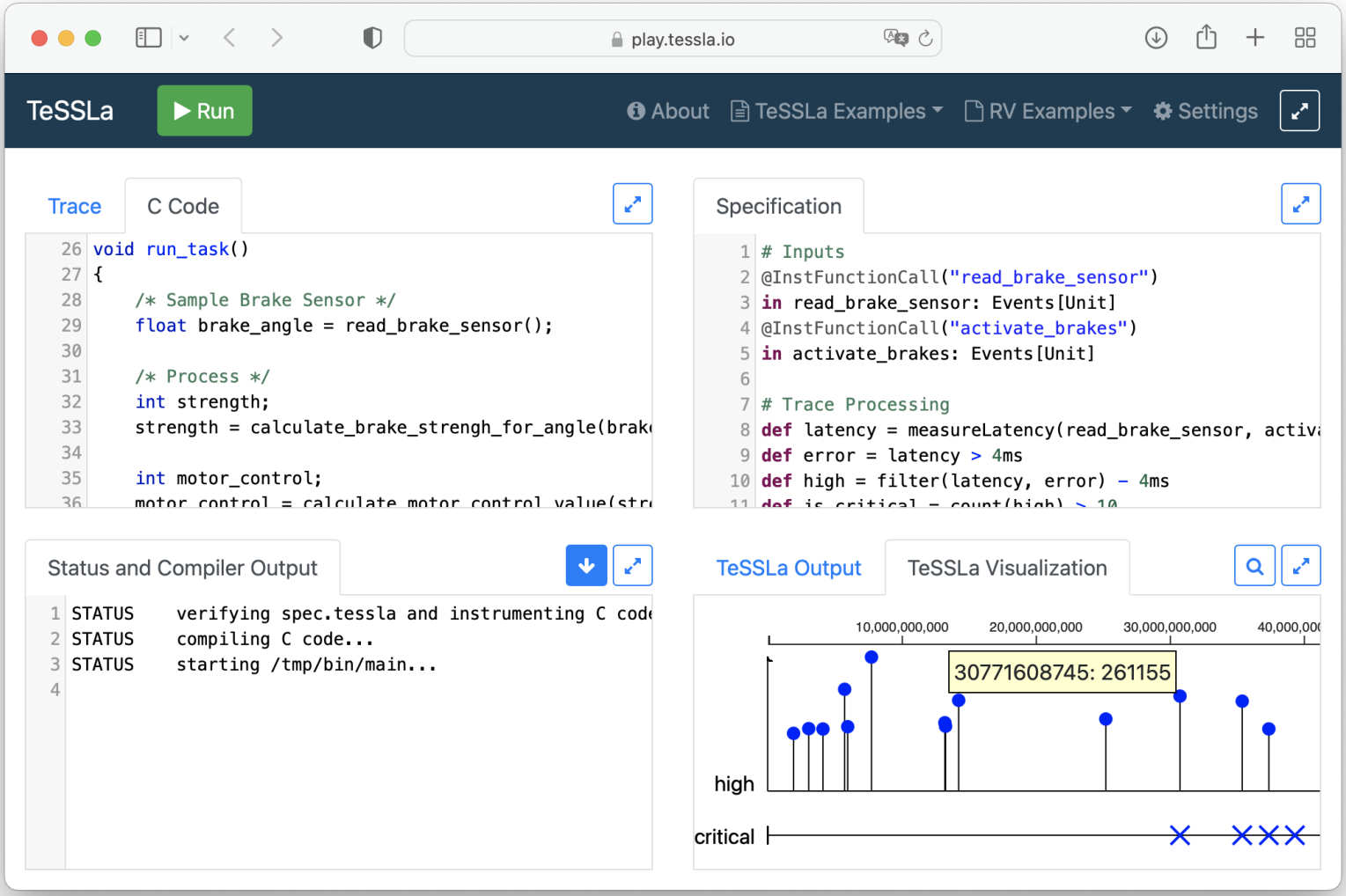


Observation / Instrumentation

- 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

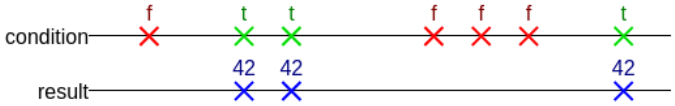
`constIf[T](value: T, condition: Events[Bool]): Events[T]`

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

Usage example:

```
in condition: Events[Bool]
def result = constIf(42, condition)
out result
```

Trace example:



Source

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

count

`count[T](x: Events[T]): Events[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 `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...

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.

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.

Future plans

The development of TeSSLa is still in progress...

Future plans

The development of TeSSLa is still in progress...

- Improvement of the TeSSLa language and compilers

Future plans

The development of TeSSLa is still in progress...

- Improvement of the TeSSLa language and compilers
- Extension of the tool chain: Further backends, integration with other tools

Future plans

The development of TeSSLa is still in progress...

- Improvement of the TeSSLa language and compilers
- Extension of the tool chain: Further backends, integration with other tools
- Development of further libraries for specific RV applications

Find out more

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

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

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

Contact:
info@tessla.io