TeSSLa – An Introduction

Hannes Kallwies Martin Leucker Malte Schmitz Daniel Thoma

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

Formal Methods Porto, October 2019

TeSSLa

- **Declarative** style: Specification rather than implementation
- Abstractions for both **events** and **signals**
- Useful for description of **Cyber Physical Systems**
- **Modularity**: Allowing abstractions based on **few primitives**
- **Time** as first-class citizen
- **Recursion** to reason about past
- Implementable with **limited memory**

TeSSLa

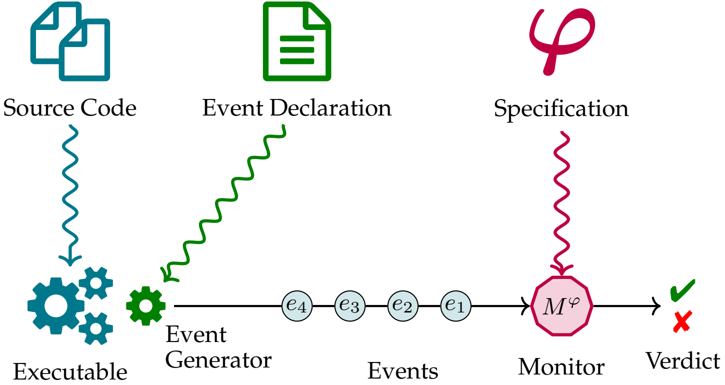
TeSSLa is a general purpose Stream-based Specification language:

Every monotonous, continuous and future-independent stream transformation function f can be specified in TeSSLa

Possible fields of application:

- Online Monitoring
- Logfile Analysis
- Event pattern generation
- Analysis of the specification
- ...

Runtime Verification with TeSSLa

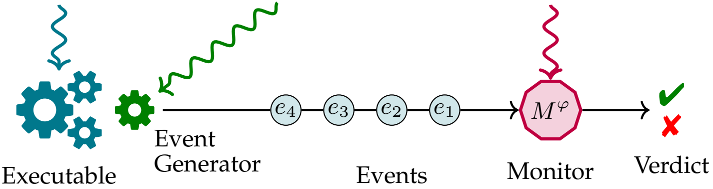


Runtime Verification with TeSSLa

```
int main() {
  while (1) {
    lock();
    critical();
    unlock();
  }
}
```

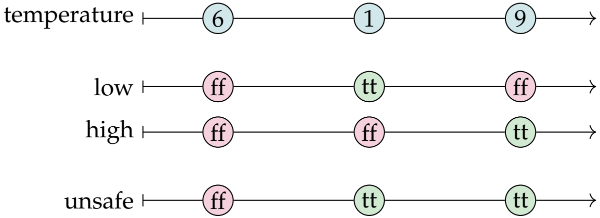
```
@InstFunctionCall("lock")
    in lock: Events[Unit]
@InstFunctionCall("unlock")
    in unlock: Events[Unit]
@InstFunctionCall("critical")
    in crit: Events[Unit]

    out on(crit, count(lock) -
                count(unlock) == 1)
    as verdict
```



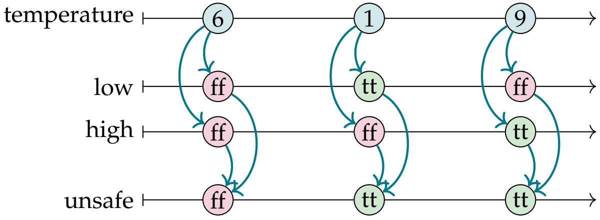
SRV: Combining streams

Correctness property: Temperature is between 2 and 8.



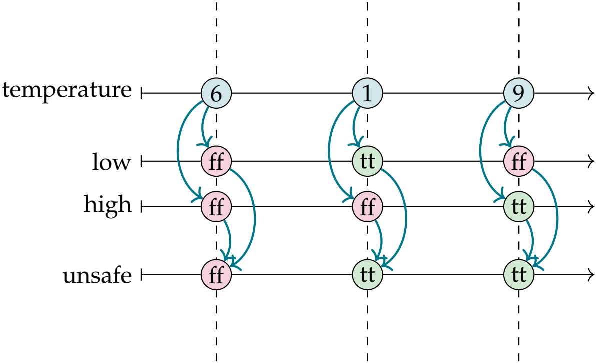
SRV: Combining streams

Correctness property: Temperature is between 2 and 8.



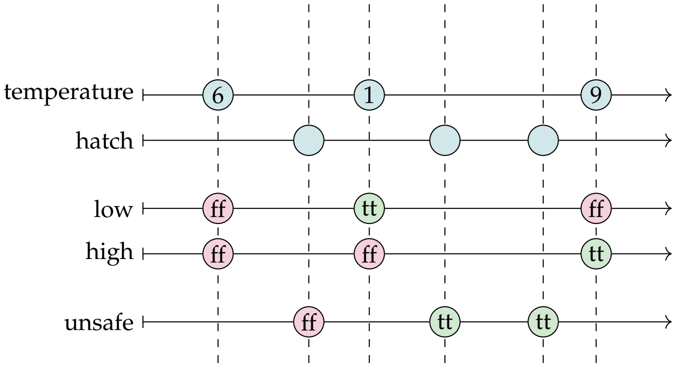
SRV: Synchronous streams

Correctness property: Temperature is between 2 and 8.



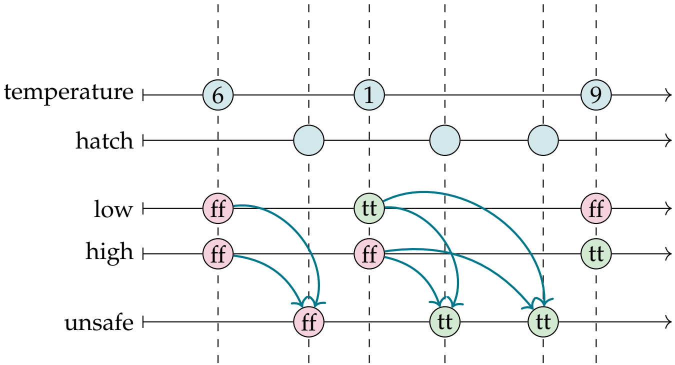
SRV: Synchronous streams

Correctness property: Temperature is between 2 and 8, when hatch is opened.



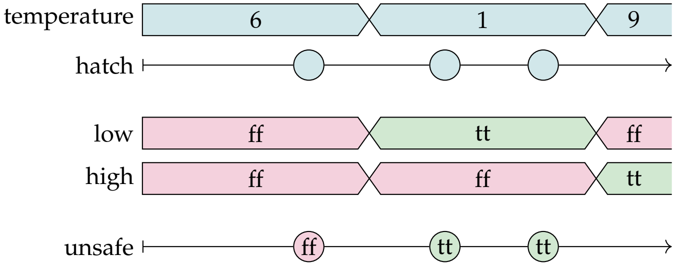
SRV: Synchronous streams

Correctness property: Temperature is between 2 and 8, when hatch is opened.



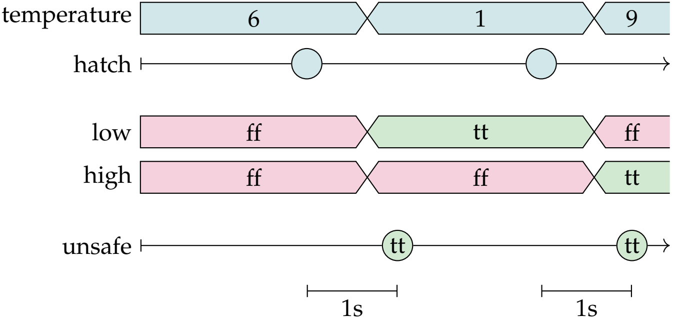
SRV: Signal semantics

Correctness property: Temperature is between 2 and 8, when hatch is opened.



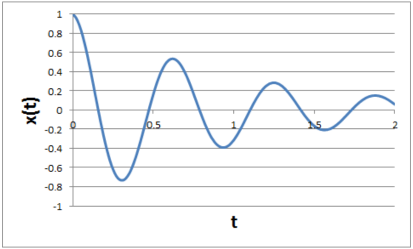
SRV: Asynchronous streams

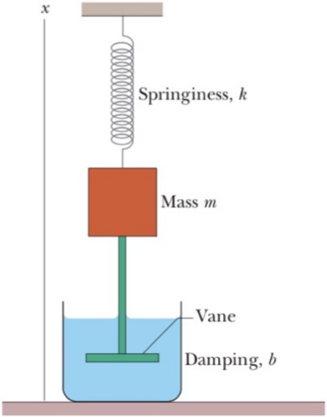
Correctness property: Temperature is between 2 and 8, one second after hatch is opened.



Spring example

$$m \cdot y'' = -D \cdot y - d \cdot y'$$





Spring example

$$m \cdot y'' = -D \cdot y - d \cdot y' \iff y'' = \frac{-D}{m} \cdot y - \frac{d}{m} \cdot y'$$

Spring pendulum in TeSSLa

```
in sensor: Events [Float]

def m: Float = 0.2  # kg
def D: Float = 2.6  # N/m
def d: Float = 0.15 # kg/s

def y''(t: Float, y: Float, y': Float): Float =
    -D / m * y - d / m * y'

def y_0 = 0.2    # m
def y'_0 = 0.0   # m/s

def approx: Events [(Float, Float)] =
    rk4(y'', y_0, y'_0)

def approxY: Events [Float] = approx._1
def alarm = |sensor-approxY| > e
```

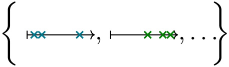
TeSSLa in comparison

Set of traces

$$\left\{a_1a_2a_3\ldots, b_1b_2b_3\ldots, \ldots\right\}$$

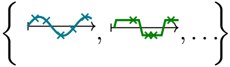
LTL

Set of streams



MTL

Set of signals



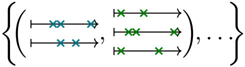
STL

Function from traces to traces

$$\left\{\left(\begin{matrix}a_1a_2a_3\cdots\\ b_1b_2b_3\ldots\\ c_1c_2c_3\ldots\end{matrix}, \begin{matrix}x_1x_2x_3\cdots\\ y_1y_2y_3\cdots\end{matrix}\right), \ldots\right\}$$

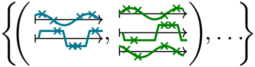
LOLA

Function from streams to streams



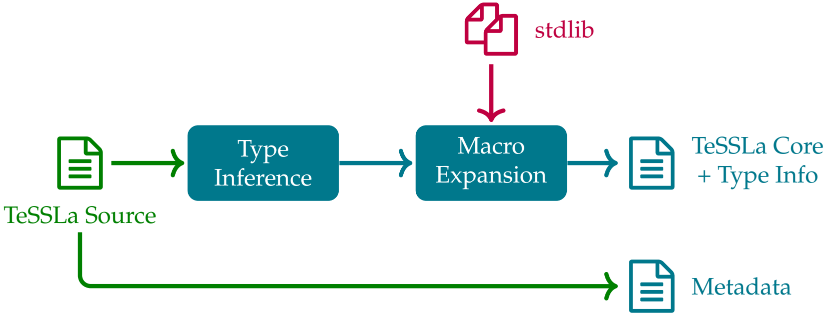
TeSSLa

Function from signals to signals



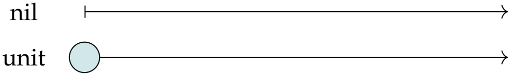
TeSSLa + Diffeq.

TeSSLa Language



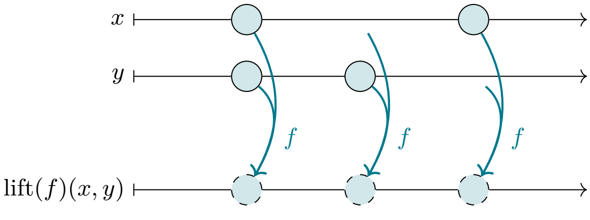
TeSSLa Core: Nil and Unit

- *nil* is the empty stream
- *unit* produces exactly one unit-event with timestamp zero



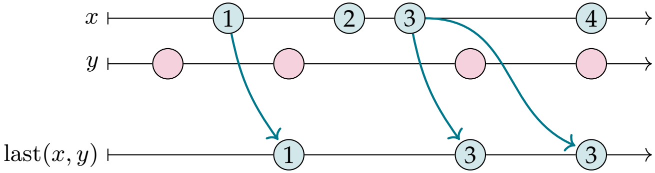
TeSSLa Core: Lift

- *Lift* applies a function to the current events on a certain number of streams
- e.g. adds two numerical event values



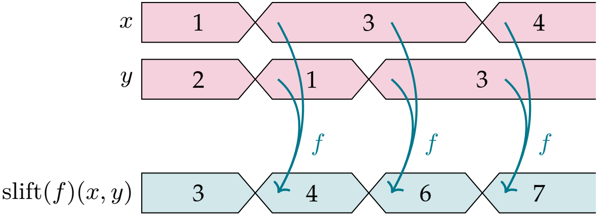
TeSSLa Core: Last

- *Last* allows to access the values of events on one stream that occurred strictly before the events on another stream
- Important for accessing streams with signal semantics



Signal-Lift (stdlib)

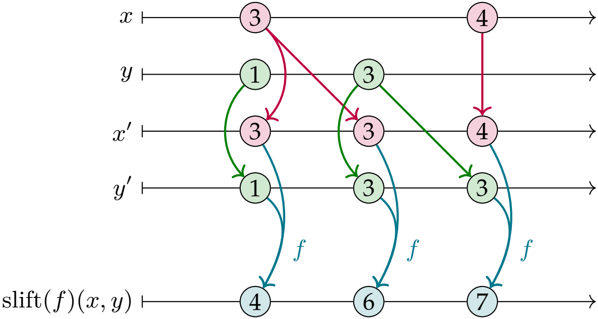
- *Signal lift* allows to lift operations on arbitrary data types to streams
- Useful for streams with signal semantics



Signal-Lift (stdlib)

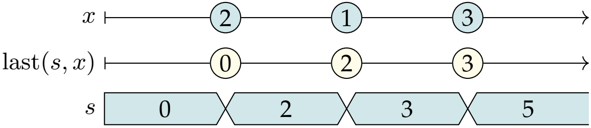
$$\begin{aligned}\mathrm{slift}(f)(x,y) &= \mathrm{lift}(f')(x',y')\\ x' &= \mathrm{merge}(x, \mathrm{last}(x,y))\\ y' &= \mathrm{merge}(y, \mathrm{last}(y,x))\end{aligned}$$

$$f'(a,b) = \begin{cases} f(a,b) & \text{if } a \neq \bot \wedge b \neq \bot \\ \bot & \text{else}\end{cases}$$



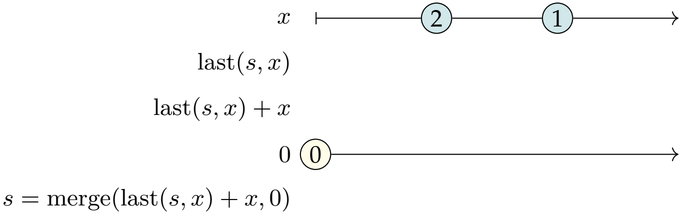
Recursive Equations in TeSSLa

- The *last* operator allows to write *recursive equations*
- The *merge* operation allows to *initialize* recursive equations with an initial event from an other stream
- Express *aggregation* operations like the *sum* over all values of a stream
- Evaluation algorithm iterates progressing event streams until fixed-point is reached

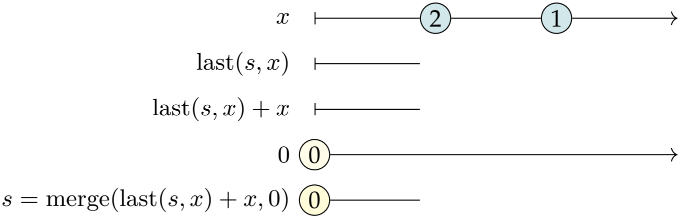


```
def s := merge(last(s, x) + x, 0)
```

Recursive Equations in TeSSLa: How It Works

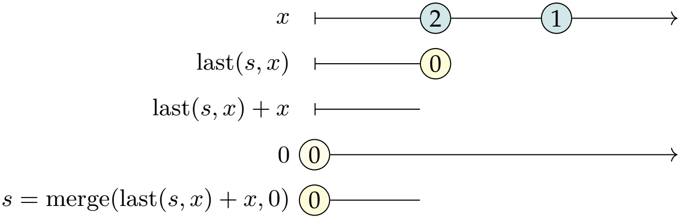

x
2
1
$\mathrm{last}(s, x)$
$\mathrm{last}(s, x) + x$
0
0
$s = \mathrm{merge}(\mathrm{last}(s, x) + x, 0)$

Recursive Equations in TeSSLa: How It Works

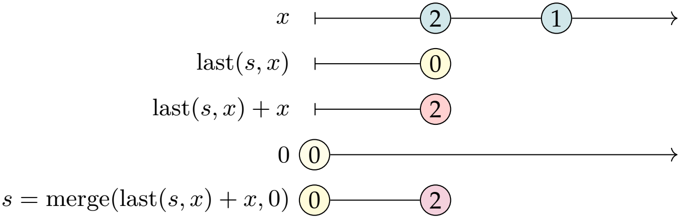


x
2
1
$\mathrm{last}(s, x)$
$\mathrm{last}(s, x) + x$
0
0
$s = \mathrm{merge}(\mathrm{last}(s, x) + x, 0)$
0

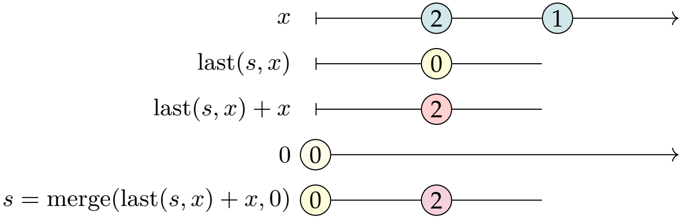
Recursive Equations in TeSSLa: How It Works


x
2
1
$\mathrm{last}(s, x)$
0
$\mathrm{last}(s, x) + x$
0
0
$s = \mathrm{merge}(\mathrm{last}(s, x) + x, 0)$
0

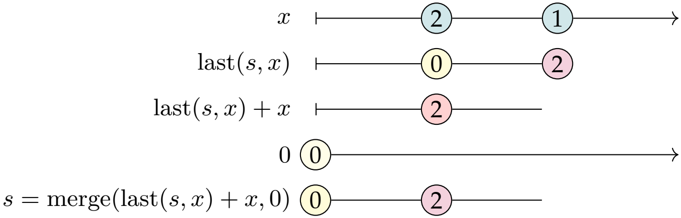
Recursive Equations in TeSSLa: How It Works


x
2
1
$\mathrm{last}(s, x)$
0
$\mathrm{last}(s, x) + x$
2
0
0
$s = \mathrm{merge}(\mathrm{last}(s, x) + x, 0)$
0
2

Recursive Equations in TeSSLa: How It Works

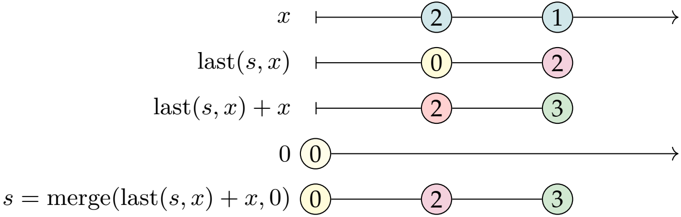

x
2
1
$\mathrm{last}(s, x)$
0
$\mathrm{last}(s, x) + x$
2
0
0
$s = \mathrm{merge}(\mathrm{last}(s, x) + x, 0)$
0
2

Recursive Equations in TeSSLa: How It Works

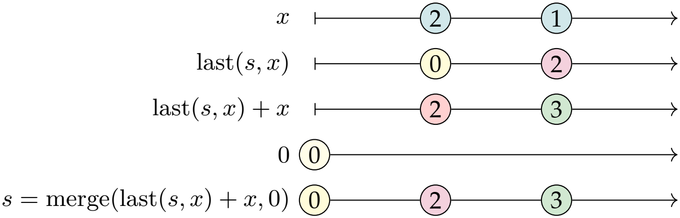


x
2
1
$\mathrm{last}(s, x)$
0
2
$\mathrm{last}(s, x) + x$
2
0
0
$s = \mathrm{merge}(\mathrm{last}(s, x) + x, 0)$
0
2

Recursive Equations in TeSSLa: How It Works

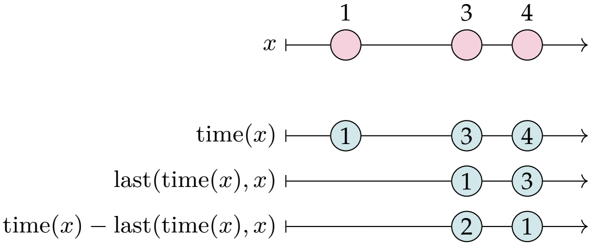


Recursive Equations in TeSSLa: How It Works


x
2
1
$\mathrm{last}(s, x)$
0
2
$\mathrm{last}(s, x) + x$
2
3
0
0
$s = \mathrm{merge}(\mathrm{last}(s, x) + x, 0)$
0
2
3

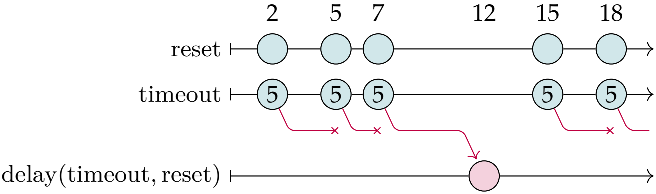
TeSSLa Core: Time

- *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.



TeSSLa Core: Delay

- *Delay* creates a new event some time after a reset event
- Possibility to create output events at timestamps without input events



TeSSLa Language: Typesystem

- Built-in basic types can be extended by user-defined types
- Supports externally defined nominal types
- Record types
- Generics

Supported basic types:

- Unit
- Int
- Float
- Boolean
- String

Supported complex datastructures:

- Lists
- Sets
- Maps

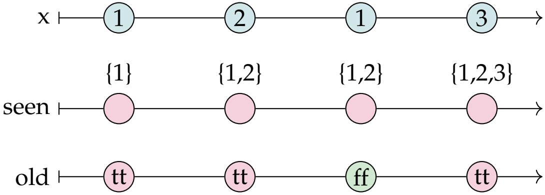
Complex datastructures

Complex datastructures

```
in x: Events[Int]

def seen: Events[Set[Int]] :=
    merge(Set_add(last(seen, x), x), Set_empty[Int])

out Set_contains(last(seen, x), x) as old
```



Macro-System

- Possibility to extend minimal language core (nil, unit, time, delay, lift, last) by arbitrary functions

Makro Definition Fold

```
def fold[T,R](stream: Events[T], init: R,
              f: (Events[R],Events[T]) => Events[R]) = result
where
{
    def result: Events[R] = default(f(last(result, stream),
                                           stream), init)
}
```

Usage of Fold

```
def y = fold(x, 0, (x: Events[Int], _: Events[Int]) => x+1)
```

Standard-Library

Defines a high number of macros to make the usage of TeSSLa comfortable

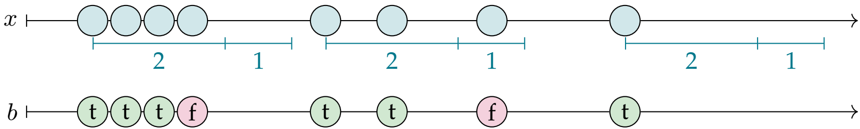
- Basic operations: `Merge, Signal Lift, Const, Filter, ...`
- Aggregation functions: `Minimum, Maximum, Fold, Reduce, Filter, ...`
- Common datastrucutre functions: `Set_contains, Map_getOrElse, ...`
- Application specific functions: `Burst-Pattern recognition, Event-Chain recognition, ...`

Standard-Library

Burst Pattern

```
in x: Events[Int]

out bursts(x, burstLength = 2, waitingPeriod = 1,
           burstAmount = 3) as b
```



Meta Data/Annotations

Possibility to pass Event declaration to connected tools:

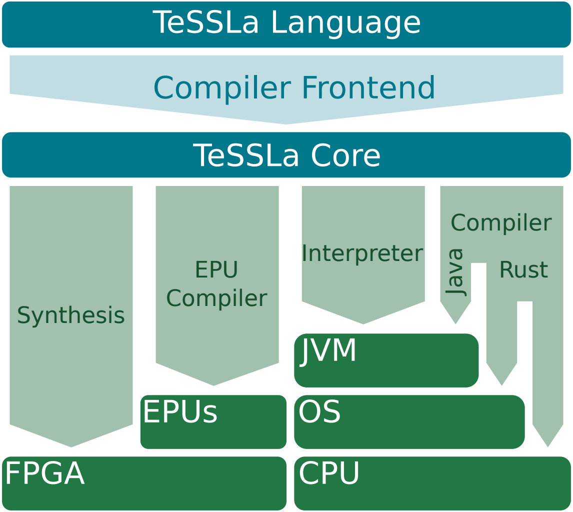
- `@InstFunctionReturn("func_name")`
- `@InstFunctionCall("func_name")`
- `@InstFunctionCallArg("func_name", par_pos)`

- `@LocalWrite(var_name)`
- `@GlobalRead(var_name)`

- `@ThreadId`

- ...

Evaluation Approaches for TeSSLa

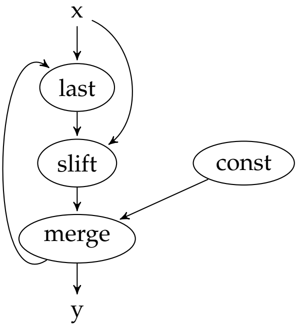


TeSSLa Evaluation

```
in x: Events[Int]
def y = merge(last(y,x) + x, 0)
out y
```

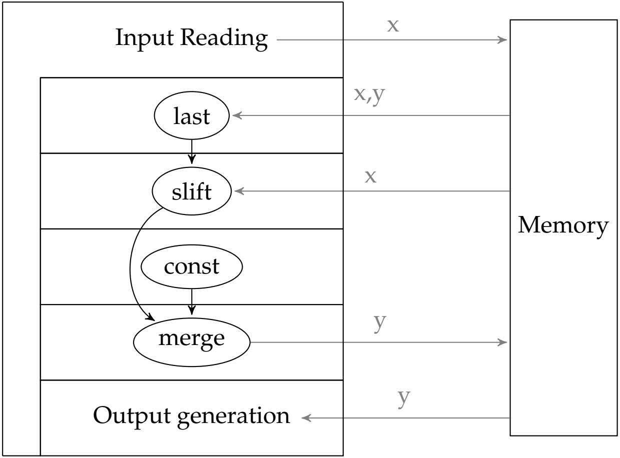
TeSSLa Interpreter

```
in x: Events[Int]
def y = merge(last(y,x) + x, 0)
out y
```



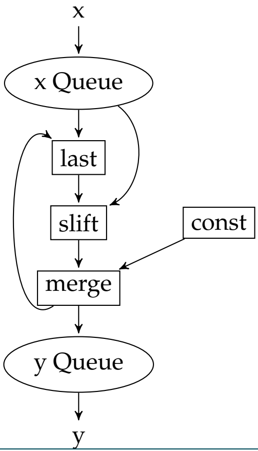
TeSSLa Compiler

```
in x: Events[Int]
def y = merge(last(y,x) + x, 0)
out y
```



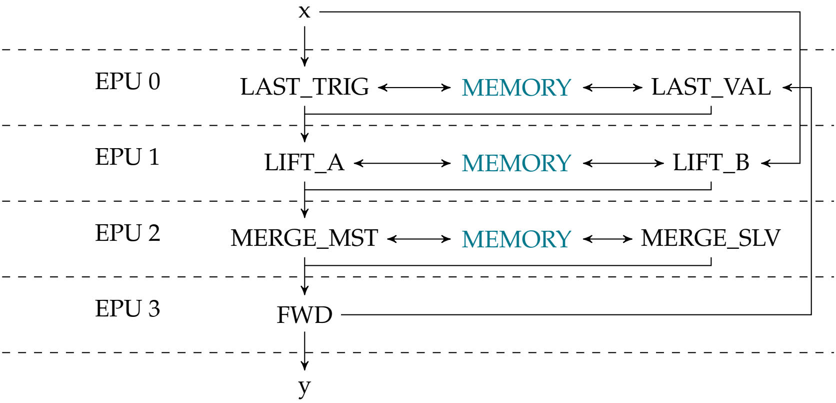
FPGA Synthesis

```
in x: Events[Int]
def y = merge(last(y,x) + x, 0)
out y
```



EPU Configuration

```
in x: Events[Int]
def y = merge(last(y,x) + x, 0)
out y
```



x
EPU 0
LAST_TRIG
MEMORY
LAST_VAL
EPU 1
LIFT_A
MEMORY
LIFT_B
EPU 2
MERGE_MST
MEMORY
MERGE_SLV
EPU 3
FWD
y