UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

isp

TeSSLa: A Real-Time Specification Language for Runtime Verification of Non-synchronized Streams

S. Hungerecker M. Leucker T. Scheffel M. Schmitz D. Thoma

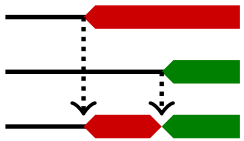
Institute for Software Engineering and Programming Languages,
University of Lübeck, Germany

Dagstuhl Seminar on Behavioural Specification 2017

Temporal Stream-based Specification Language

- Reasons over a set of **non-synchronized real-time streams**
- **Declarative** style: Specification rather than implementation
- **Modularity**: Allowing abstractions based on six core operators
- **Time** as first-class citizen
- Abstraction for both, **events** and **signals**
- **Recursion** to reason about the past
- Implementable with **limited memory**

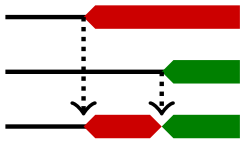
TeSSLa core operators



default, defaultFrom

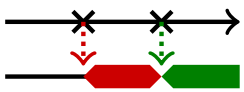
- Initialize streams
- Start of recursion

TeSSLa core operators



default, defaultFrom

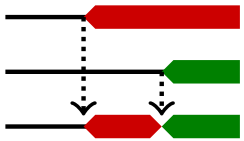
- Initialize streams
- Start of recursion



time

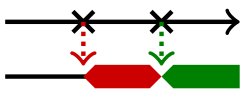
- Get timestamps of stream
- Replaces data values with timestamps
- Only way to read timestamps

TeSSLa core operators



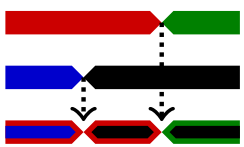
default, defaultFrom

- Initialize streams
- Start of recursion



time

- Get timestamps of stream
- Replaces data values with timestamps
- Only way to read timestamps



lift

- Lifts standard functions to streams
- Used to manipulate data, events, ...

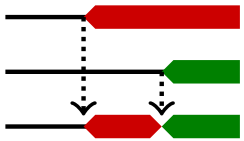
TeSSLa: Lifted Addition

Lift arbitrary functions to streams with signal semantics.

$$\mathsf{lift}_{\mathbb{D}_1,\ldots,\mathbb{D}_n,\mathbb{D}'} : (\mathbb{D}_1 \times \cdots \times \mathbb{D}_n \rightharpoonup \mathbb{D}') \to (\mathcal{S}_{\mathbb{D}_1} \times \cdots \times \mathcal{S}_{\mathbb{D}_n} \to \mathcal{S}_{\mathbb{D}'})$$

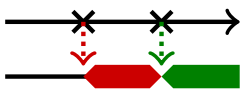
$$\mathsf{add} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z},$$
$$z = \mathsf{lift}(\mathsf{add})(x, y)$$

TeSSLa core operators



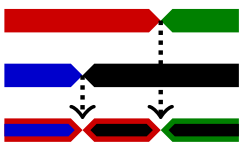
default, defaultFrom

- Initialize streams
- Start of recursion



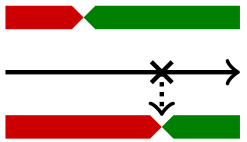
time

- Get timestamps of stream
- Replaces data values with timestamps
- Only way to read timestamps



lift

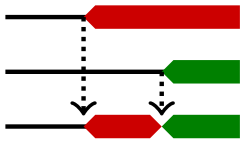
- Lifts standard functions to streams
- Used to manipulate data, events, ...



last

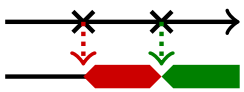
- Refers to previous value of a stream
- Recursion

TeSSLa core operators



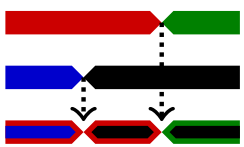
default, defaultFrom

- Initialize streams
- Start of recursion



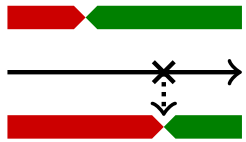
time

- Get timestamps of stream
- Replaces data values with timestamps
- Only way to read timestamps



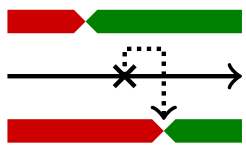
lift

- Lifts standard functions to streams
- Used to manipulate data, events, ...



last

- Refers to previous value of a stream
- Recursion



delayedLast

- Only way to create events
- Takes a stream and delays events by its current value
- Output events have the previous value of another given stream

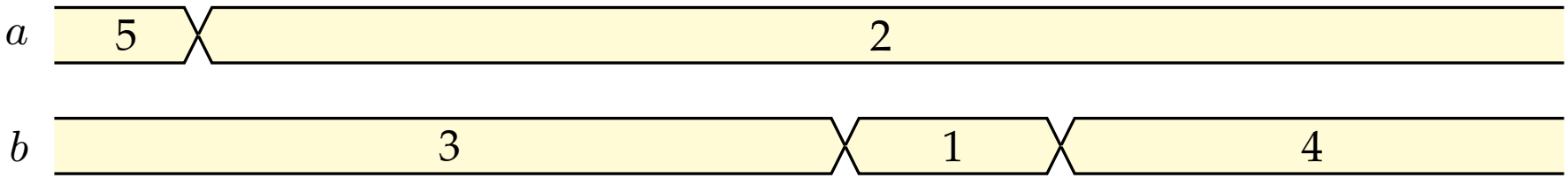
TeSSLa: easy example

Let $a \in \mathcal{S}_{\text{Int}}$ be an input stream, $x, y \in \mathcal{S}_{\text{Int}}$ be intermediate streams and $z \in \mathcal{S}_{\text{Int}}$ be the output stream.

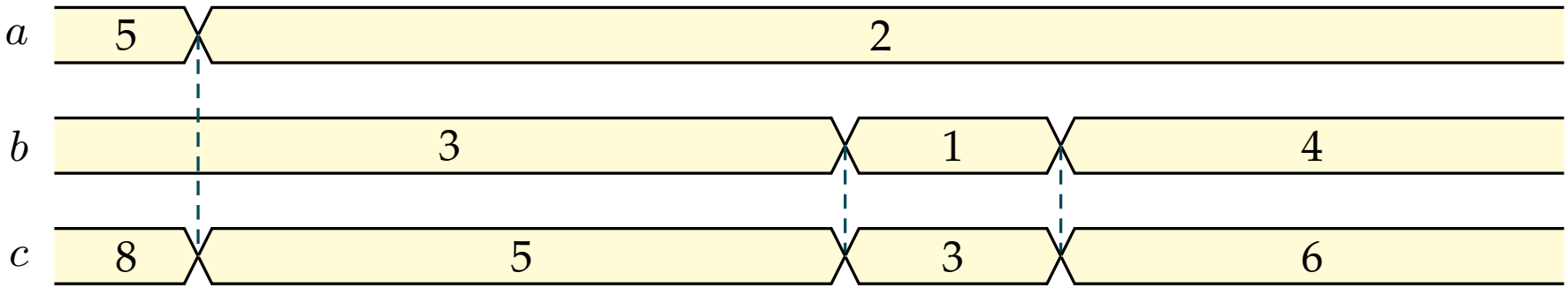
TeSSLa specifications can be seen as a system of equations:

```
x = last(z, a)
y = default(x, 0)
z = y + a
```

TeSSLa by Example

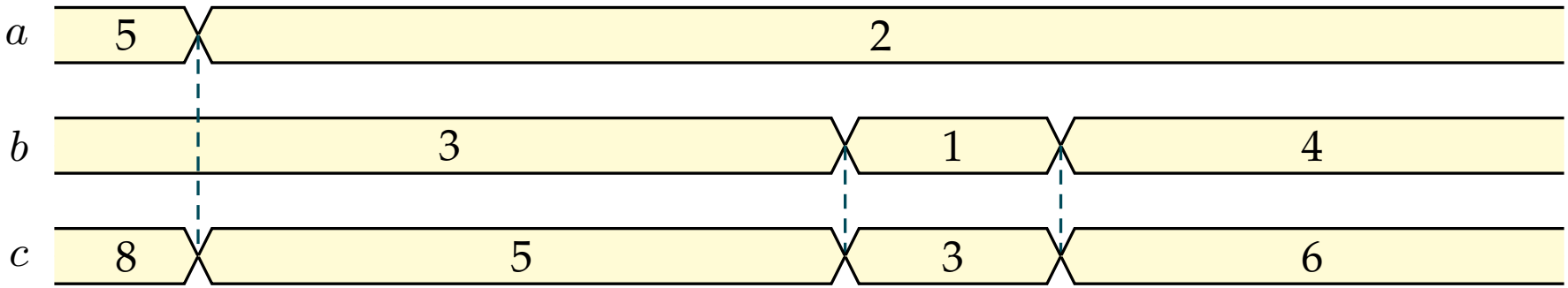


TeSSLa by Example

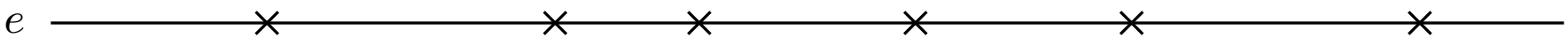



```
def c := a + b
```

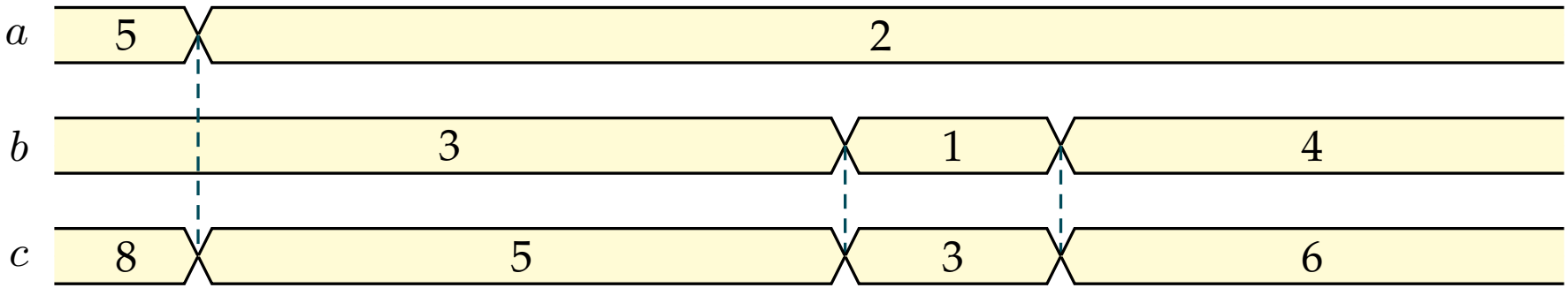
TeSSLa by Example



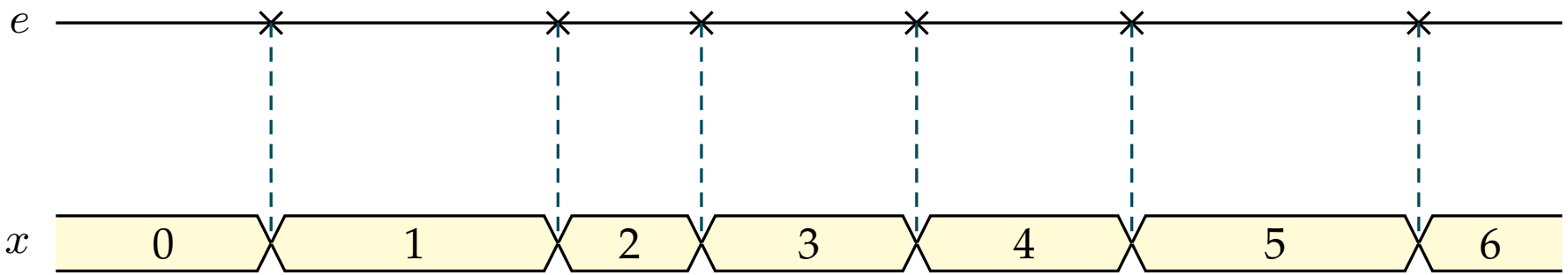

```
def c := a + b
```



TeSSLa by Example

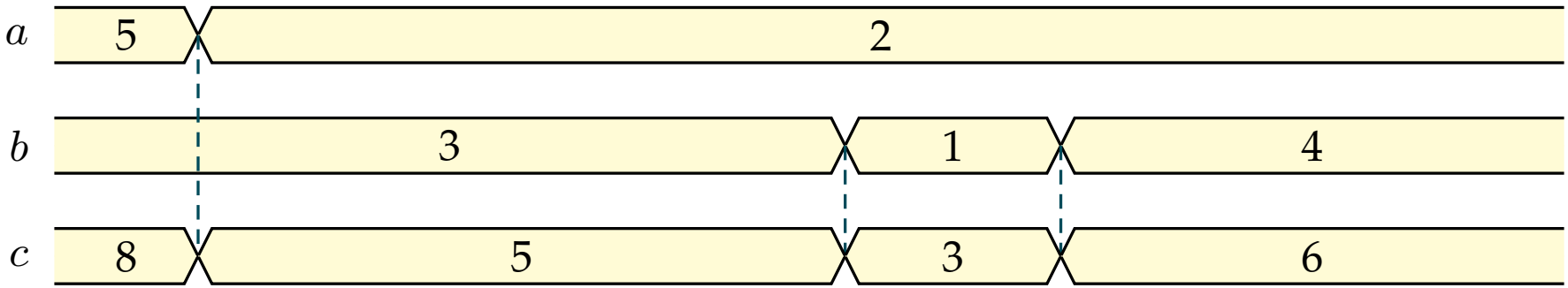



```
def c := a + b
```

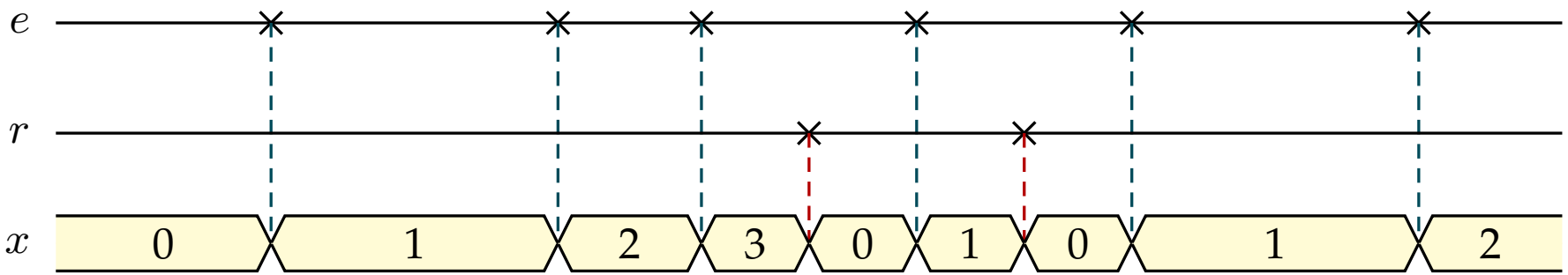



```
def x := eventCount(e)
```

TeSSLa by Example




```
def c := a + b
```




```
def x := eventCount(e, reset = r)
```

TeSSLa Standard Library

Counting events until reset

```
def eventCount(values, reset) :=
  default(
    if default(time(reset) > time(values), false)
    then 0
    else if default(time(reset) == time(values), false)
    then 1
    else last(eventCount, values) + 1
  , 0)
```

Outlook

In the future, the following features would be desirable for TeSSLa:

- High level data structures like maps and queues
- Possibility to handle gaps and uncertainties in streams
- ...

This will result in

$$\underset{\text{Bounded maps / queues, ...}}{\text{TeSSLa}_{\text{Hardware}}} \subset \underset{\text{Effectively monitorable}}{\text{TeSSLa}_{\text{Software}}} \subseteq \text{TeSSLa}$$