# Verified Time-Aware Stream Processing

Rafael Castro G. Silva

rasi@di.ku.dk

Department of Computer Science
University of Copenhagen

02/11/2023

# What is this PhD/Status seminar about?

- Distributed Systems
    - Stream processing frameworks
        - Dataflow models
        - Time-Aware Computations
- Formal Methods
    - Verification using proof assistants
        - Isabelle proofs
        - Verified and executable code
- Formalization of Time-Aware Stream Processing

# Contents

# Introduction

# Dataflow Models

- Stream Processing
- Dataflow Model
- Time-Aware Computations
- Bugs in Stream Processing

# Preliminaries

# Isabelle/HOL

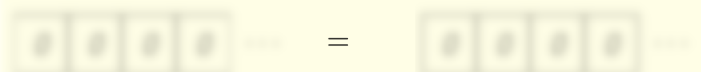
- HOL
- Isabelle/HOL

# Isabelle/HOL: (Co)datatypes

- Datatypes and Codatatypes

$$\textbf{codatatype}\ (\mathsf{lset}\colon {'a})\ \mathit{llist} = \mathsf{lnull}\colon \mathsf{LNil} \mid \mathsf{LCons}\ (\mathsf{lhd}\colon {'a})\ (\mathsf{ltl}\colon {'a}\ \mathit{llist})$$
$$\textbf{for}\ \mathsf{map}\colon \mathsf{lmap}\ \textbf{where}\ \mathsf{ltl}\ \mathsf{LNil} = \mathsf{LNil}$$

- Examples:
  - $\mathsf{LNil} :: \mathit{nat}\ \mathit{llist}$
  - $\mathsf{LCons}\ (\mathit{1} :: \mathit{nat})\ (\mathsf{LCons}\ \mathit{2}\ (\mathsf{LCons}\ \mathit{3}\ \mathsf{LNil}))$
  - $\mathsf{LCons}\ (\mathit{0} :: \mathit{nat})\ (\mathsf{LCons}\ \mathit{0}\ (\mathsf{LCons}\ \mathit{0}\ (\ldots)))$
- Induction principle assuming membership in the lazy list
- Coinductive principle for lazy list equality:
  - Show that there is a pair of goggles that makes them to look the same, which implies that:
    - The first lazy list if empty iff second is
    - They have the same head
    - Their tail looks the same

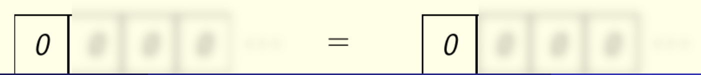


# Isabelle/HOL: (Co)datatypes

- Datatypes and Codatatypes

$$\textbf{codatatype}\ (\mathsf{lset}\colon {}'a)\ \mathit{llist} = \mathsf{lnull}\colon \mathsf{LNil} \mid \mathsf{LCons}\ (\mathsf{lhd}\colon {}'a)\ (\mathsf{ltl}\colon {}'a\ \mathit{llist})$$
$$\textbf{for}\ \mathsf{map}\colon \mathsf{lmap}\ \textbf{where}\ \mathsf{ltl}\ \mathsf{LNil} = \mathsf{LNil}$$

- Examples:
  - $\mathsf{LNil} :: \mathit{nat}\ \mathit{llist}$
  - $\mathsf{LCons}\ (1 :: \mathit{nat})\ (\mathsf{LCons}\ 2\ (\mathsf{LCons}\ 3\ \mathsf{LNil}))$
  - $\mathsf{LCons}\ (0 :: \mathit{nat})\ (\mathsf{LCons}\ 0\ (\mathsf{LCons}\ 0\ (\ldots)))$
- Induction principle assuming membership in the lazy list
- Coinductive principle for lazy list equality:
  - Show that there is a pair of goggles that makes them to look the same, which must imply that:
    - The first lazy list if empty iff second is
    - They have the same head
    - Their tail looks the same



# Isabelle/HOL: (Co)datatypes

- Datatypes and Codatatypes

$$\textbf{codatatype}\ (\mathsf{lset}\colon {'a})\ \mathit{llist} = \mathsf{lnull}\colon \mathsf{LNil} \mid \mathsf{LCons}\ (\mathsf{lhd}\colon {'a})\ (\mathsf{ltl}\colon {'a}\ \mathit{llist})$$
$$\textbf{for}\ \mathsf{map}\colon \mathsf{lmap}\ \textbf{where}\ \mathsf{ltl}\ \mathsf{LNil} = \mathsf{LNil}$$

- Examples:
  - $\mathsf{LNil} :: \mathit{nat}\ \mathit{llist}$
  - $\mathsf{LCons}\ (\mathit{1} :: \mathit{nat})\ (\mathsf{LCons}\ \mathit{2}\ (\mathsf{LCons}\ \mathit{3}\ \mathsf{LNil}))$
  - $\mathsf{LCons}\ (\mathit{0} :: \mathit{nat})\ (\mathsf{LCons}\ \mathit{0}\ (\mathsf{LCons}\ \mathit{0}\ (\ldots)))$
- Induction principle assuming membership in the lazy list
- Coinductive principle for lazy list equality:
  - Show that there is a pair of goggles that makes them to look the same, which must imply that:
    - The first lazy list if empty iff second is
    - They have the same head
    - Their tail looks the same

$$\boxed{0}\boxed{0}\boxed{0}\boxed{0}\ \cdots \quad = \quad \boxed{0}\boxed{0}\boxed{0}\boxed{0}\ \cdots$$

# Isabelle/HOL: Recursion and While Combinator

- Recursion

  **fun** lshift $:: {'a}\ list \Rightarrow {'a}\ llist \Rightarrow {'a}\ llist$ (*infixr* @@ *65*) **where**
  lshift [] $lxs = lxs$
  | lshift $(x \# xs)$ $lxs =$ LCons $x$ (lshift $xs$ $lxs$)

- While Combinator

  **definition** while_option $:: ({'a} \Rightarrow bool) \Rightarrow ({'a} \Rightarrow {'a}) \Rightarrow {'a} \Rightarrow {'a}\ option$ **where**
  while_option $b\ c\ s = \ldots$

- While rule for invariant reasoning:
  - There is something that holds before a step; that thing still holds after the step

# Isabelle/HOL: Corecursion and Friends

- Corecursion is like recursion, but instead of always eventually reducing an argument it always eventually produces something
- Corec:

$$\textbf{corec}\ \mathsf{lapp} :: {'a}\ \mathit{llist} \Rightarrow {'a}\ \mathit{llist} \Rightarrow {'a}\ \mathit{llist}\ \textbf{where}$$
$$\mathsf{lapp}\ \mathit{lxs}\ \mathit{lys} = \mathtt{case}\ \mathit{lxs}\ \mathtt{of}\ \mathsf{LNil} \Rightarrow \mathit{lys} \mid \mathsf{LCons}\ x\ \mathit{lxs'} \Rightarrow \mathsf{LCons}\ x\ (\mathsf{lapp}\ \mathit{lxs'}\ \mathit{lys})$$

- Friendly function
  - Preserves productivity: it may consume at most one constructor to produce one constructor.

$$\textbf{friend\_of\_corec}\ \mathsf{lshift}\ \textbf{where}$$
$$\mathit{xs}\ @@\ \mathit{lxs} = (\mathtt{case}\ \mathit{xs}\ \mathtt{of}$$
$$[] \Rightarrow (\mathtt{case}\ \mathit{lxs}\ \mathtt{of}\ \mathsf{LNil} \Rightarrow \mathsf{LNil} \mid \mathsf{LCons}\ x\ \mathit{lxs'} \Rightarrow \mathsf{LCons}\ x\ \mathit{lxs'})$$
$$\mid x\#\mathit{xs'} \Rightarrow \mathsf{LCons}\ x\ (\mathit{xs'}\ @@\ \mathit{lxs}))$$
$$\textbf{by}\ (\mathit{auto\ split\!: list.splits\ llist.splits})\ (\mathit{transfer\_prover})$$

$$\mathsf{lconcat}\ \mathit{lxs} = \mathtt{case}\ \mathit{lxs}\ \mathtt{of}\ \mathsf{LNil} \Rightarrow \mathsf{LNil} \mid \mathsf{LCons}\ \mathit{xs}\ \mathit{lxs'} \Rightarrow \mathsf{lshift}\ \mathit{xs}\ (\mathsf{lconcat}\ \mathit{lxs'})$$

# Isabelle/HOL: (Co)inductive Predicates

- Inductive predicate
  - Finite number of introduction rule applications

**inductive** in_llist :: $'a \Rightarrow 'a\ llist \Rightarrow bool$ **where**
  In_llist: in_llist $x$ (LCons $x$ $lxs$)
| Next_llist: in_llist $x$ $lxs$ $\Rightarrow$ in_llist $x$ (LCons $y$ $lxs$)

in_llist ($2$::$nat$) (LCons $1$ (LCons ($2$ ($\ldots$))))

- Coinductive predicate
  - Infinite number of introduction rule applications

**coinductive** lprefix :: $'a\ llist \Rightarrow 'a\ llist \Rightarrow bool$ **where**
  LNil_lprefix: lprefix LNil $lys$
| LCons_lprefix: lprefix $lxs$ $lys$ $\Rightarrow$ lprefix (LCons $x$ $lxs$) (LCons $x$ $lys$)

lprefix (LCons $1$ (LCons ($2$ ($\ldots$)))) (LCons $1$ (LCons ($2$ ($\ldots$))))

- Coinduction principle

# Lazy Lists Processors

# Operators

- Operator as a codatatype
  - Taking $'i$ as the input type, and $'o$ as the output type:

$$\textbf{codatatype } ('o, 'i)\ op = \mathsf{Logic}\ (\mathsf{apply}\colon ('i \Rightarrow ('o, 'i)\ op \times 'o\ list))$$

  - Infinite trees: applying the selector apply "walks" a branch of the tree
- Produce function: applies the logic (co)recursively throughout a lazy list
  - The recursive part:

$$
\begin{aligned}
&\textbf{definition}\ \mathsf{produce}_1'\ op\ lxs = \mathsf{while\_option}\\
&\quad (\lambda(op, lxs).\ \neg\ \mathsf{lnull}\ lxs \wedge \mathsf{snd}\ (\mathsf{apply}\ op\ (\mathsf{lhd}\ lxs)) = [])\\
&\quad (\lambda(op, lxs).\ (\mathsf{fst}\ (\mathsf{apply}\ op\ (\mathsf{lhd}\ lxs)), \mathsf{ltl}\ lxs))\ (op, lxs)\\
&\textbf{definition}\ \mathsf{produce}_1\ op\ lxs =\\
&\quad (\mathtt{case}\ \mathsf{produce}_1'\ op\ lxs\ \mathtt{of}\ \mathsf{None} \Rightarrow \mathsf{None}\\
&\quad \mid \mathsf{Some}\ (op', lxs') \Rightarrow \mathtt{if}\ \mathsf{lnull}\ lxs'\ \mathtt{then}\ \mathsf{None}\ \mathtt{else}\\
&\quad\quad \mathtt{let}\ (op'', out) = \mathsf{apply}\ op'\ (\mathsf{lhd}\ lxs')\ \mathtt{in}\ \mathsf{Some}\ (op'', \mathsf{hd}\ out, \mathsf{tl}\ out, \mathsf{ltl}\ lxs'))
\end{aligned}
$$

# Operators: Example

- Example: counts elements satisfying $P$:

**corec** count_op **where** count_op $P$ $n$ =
  Logic ($\lambda e.$ `if` $P$ $e$ `then` (count_op $P$ ($n$ + $1$), [$n$+$1$]) `else` (count_op $P$ $n$, []))

| | | | | | | |
|---|---|---|---|---|---|---|
| $stream_2$ | = | $0::nat$ | $3$ | $3$ | $6$ | $24$ |

| | | | | | | |
|---|---|---|---|---|---|---|
| produce (count_op $is\_even$ $0$) | $stream_2$ | = | $1$ | $2$ | $3$ |

Figure: Difference between incr_hist_op′ and incr_hist_op

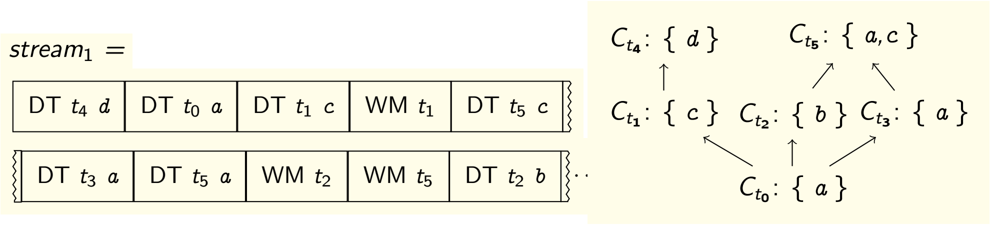
# Sequential Composition

- Composition
- Skip n

# Time-Aware Operators

# Monotone and Productive Time-Aware Streams

- Monotone
- Productive




(a) Prefix of $stream_1$

(b) Corresponding set of collections

Figure: An example stream and its collections (ordered by their time-stamps)

# Building Blocks: Batch Operator

# Batch Operator: Soundness

# Batch Operator: Completeness

- Uses soundness of batch_op
- Proof by induction over n

$$\begin{aligned}
&\text{mono_prod } \mathit{lxs}\ W \longrightarrow (\exists i\ d.\ \text{enat } i < \text{llength } \mathit{lxs} \wedge \text{lnth } \mathit{lxs}\ i = \text{DT } t\ d \wedge n = \text{Suc } i) \vee \\
&n = 0 \wedge t \in \text{set_t } \mathit{buf} \longrightarrow (\forall t' \in \text{set_t } \mathit{buf}.\ \text{lfinite } \mathit{lxs} \vee \exists wm \geq t'.\ \text{WM } wm \in \text{lset } \mathit{lxs}) \longrightarrow \\
&\exists wm\ \mathit{batch}.\ \text{DT } wm\ \mathit{batch} \in \text{lset } (\text{produce } (\text{batch_op } \mathit{buf})\ \mathit{lxs}) \wedge t \in \text{set_t } \mathit{batch} \vee \\
&(\forall k \in \{n\ ..< \text{the_enat } (\text{llength } \mathit{lxs})\}.\ \neg (\exists t' \geq t.\ \text{lnth } \mathit{lxs}\ k = \text{WM } t')) \wedge \text{lfinite } \mathit{lxs}
\end{aligned} \tag{1}$$

# Batch Operator: Monotone

# Batch Operator: Productive

# Building Blocks: Incremental Operator

# Batch Operator: Soundness

# Batch Operator: Completeness

# Batch Operator: Monotone

# Batch Operator: Productive

# Compositional Reasoning

# Case Study

# Histogram

# Histogram: Soundness

# Histogram: Completeness

# Histogram: Monotone

# Histogram: Productive

# Efficient Histogram

- Foo

# Join

# Join: Soundness

# Join: Completeness

# Join: Monotone

# Next Steps

# Next Steps

Questions, comments and suggestions