# Efficient and Verified Non-Terminating Programs with Isabelle-LLVM

Rafael Castro G. Silva

rasi@di.ku.dk

Department of Computer Science
University of Copenhagen

02/11/2023

# Introduction

# Context

# Context

- Distributed Systems
  - Stream processing frameworks
    - Dataflow models
      - Time-Aware Computations

# Context

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

# Stream Processing

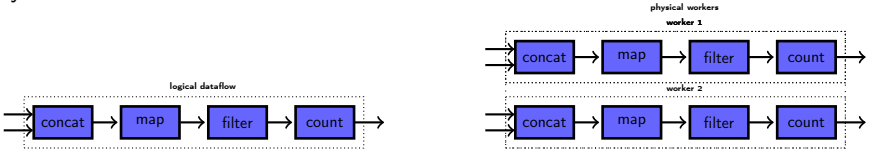
- Stream Processing: Abstraction for processing data when the input is not completely presented in the begging of the computation

# Stream Processing

- Stream Processing: Abstraction for processing data when the input is not completely presented in the begging of the computation
- Dataflow Model:
  - Directed graph of interconnected operators that perform event-wise transformations
  - E.g.: Apache Flink, Apache Samza, Apache Spark, Google Cloud Dataflow, and Timely Dataflow

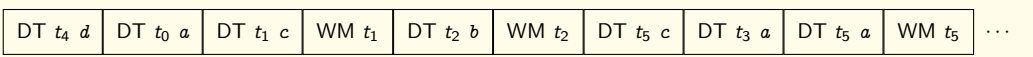
Cloud DataFlow Flink

- Highly Parallel



# Time-Aware Stream Processing

- Time-Aware Computations:
  - Timestamps: Metadata associating the data with some data collection
    - An unix timestamp
    - Version of the data
    - Logical grouping
  - Watermarks: Metadata indicating the completion of a data collection
    - e.g.: A watermark 5 says that there is no data associated with timestamp 5 or bellow arriving
    - Are increasingly monotonic (they don't go backwards in time)
  - e.g.:



# Preliminaries

# Isabelle/HOL

- Classical higher-order logic (HOL): Simple Typed Lambda Calculus + (Hilbert) axiom of choice + axiom of infinity + rank-1 polymorphism

# Isabelle/HOL

- Classical higher-order logic (HOL): Simple Typed Lambda Calculus + (Hilbert) axiom of choice + axiom of infinity + rank-1 polymorphism
- Isabelle: A generic proof assistant


- Isabelle/HOL: Isabelle's flavor of HOL

# Isabelle/HOL: (Co)datatypes

- Datatypes and Codatatypes

```
codatatype (lset: 'a) llist = lnull: LNil | LCons (lhd: 'a) (ltl: 'a llist)
  for map: lmap where ltl LNil = LNil
```

- Examples:
  - LNil
  - LCons *1* (LCons *2* (LCons *3* LNil))
  - LCons *0* (LCons *0* (LCons *0* (...)))
- Proofs by induction
- Proofs by coinduction

# State of this work

# What have I formalized so far? (part 1)

- Formalization stream processing (model)
  - Using Isabelle/HOL: (co)datatypes, (co)recursion, and (co)induction
  - Streams are lazy lists, and operators as a codatatype
  - Semantics: a `produce` function that runs an operator throughout a lazy lists
    - Mix of recursion and corecursion: inductive and coinductive principles
  - Sequential composition
    - Correctness!

# What have I formalized so far? (part 2)

- Time-Aware computations
  - Coinductive properties of streams: monotonicity and productivity
  - Building blocks operators:
    - Convenience operators: batching and incremental computations
      - Incremental computing: only update results that are affected by the new input
      - With verified properties: Soundness, Completeness, preservation of productivity, and preservation of monotonicity
  - Compositional reasoning
- Case studies with the building blocks:
  - Incremental histogram operator
  - Relational join

# Next Steps

# Efficient Stream Processing

- It is executable! But slow!
  - Code generator: functional languages (OCaml, Haskell, SML...)

# Efficient Stream Processing

- It is executable! But slow!
  - Code generator: functional languages (OCaml, Haskell, SML...)
- How do we make efficient and verified programs in Isabelle/HOL?

# Efficient Stream Processing

- It is executable! But slow!
  - Code generator: functional languages (OCaml, Haskell, SML...)
- How do we make efficient and verified programs in Isabelle/HOL?
- Isabelle-LLVM!
- This is a non-terminating program

# Isabelle-LLVM

# Refinement Framework and Isabelle-LLVM

- Evolution

# Isabelle-LLVM's Recursion Model

- Monotone: watermarks do not go back in time

# What the Heck is a CCPO?

- Connect to the Isabelle-LLVM refinement framework

# The First Steps

# Our CCPO Attempt

- 
- Exit argument
- Connect to the Isabelle-LLVM refinement framework

# Our Monad Attempt

- Feedback loop
- Exit argument
- Connect to the Isabelle-LLVM refinement framework

## Questions, comments and suggestions