Nondeterministic Asynchronous Dataflow in Isabelle/HOL

Rafael Castro G. Silva, Laouen Fernet and Dmitriy Traytel

rasi@di.ku.dk

Department of Computer Science
University of Copenhagen

14/09/2024

Introduction

Time-Aware Stream Processing

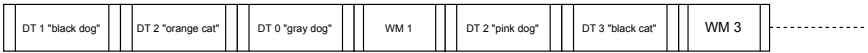
- What is Time-Aware Stream Processing?

Time-Aware Stream Processing

- What is Time-Aware Stream Processing?
  - Stream Processing: programs that compute (possibly) unbounded sequences of data (streams)

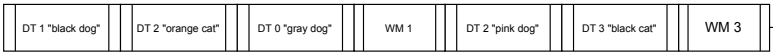
Time-Aware Stream Processing

- What is Time-Aware Stream Processing?
  - Stream Processing: programs that compute (possibly) unbounded sequences of data (streams)
  - Time-Aware: Data has timestamp metadata; timestamps are bounded by watermarks
    - Timestamp: A partially-ordered value associated with the data (e.g., unix-time, int, pairs of ints, etc...)
    - Watermark: A value of the same type of the timestamp. Represents data-completeness.



Time-Aware Stream Processing

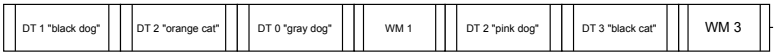
- What is Time-Aware Stream Processing?
  - Stream Processing: programs that compute (possibly) unbounded sequences of data (streams)
  - Time-Aware: Data has timestamp metadata; timestamps are bounded by watermarks
    - Timestamp: A partially-ordered value associated with the data (e.g., unix-time, int, pairs of ints, etc...)
    - Watermark: A value of the same type of the timestamp. Represents data-completeness.



- Asynchronous Dataflow Programming: Directed graph of interconnected operators that perform event-wise transformations
- E.g.: Apache Flink, Apache Samza, Apache Spark, Google Cloud Dataflow, and Timely Dataflow

Time-Aware Stream Processing

- What is Time-Aware Stream Processing?
  - Stream Processing: programs that compute (possibly) unbounded sequences of data (streams)
  - Time-Aware: Data has timestamp metadata; timestamps are bounded by watermarks
    - Timestamp: A partially-ordered value associated with the data (e.g., unix-time, int, pairs of ints, etc...)
    - Watermark: A value of the same type of the timestamp. Represents data-completeness.



- Asynchronous Dataflow Programming: Directed graph of interconnected operators that perform event-wise transformations
- E.g.: Apache Flink, Apache Samza, Apache Spark, Google Cloud Dataflow, and Timely Dataflow

- Why?
  - Highly Parallel
  - Low latency (output as soon as possible)
  - Incremental computing (re-uses previous computations)

Formalization in Isabelle/HOL

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: Codatatypes

- Codatatypes

Isabelle/HOL: Corecursion

- Recursion: always eventually reduces an argument
- Corecursion: always eventually produces something

Isabelle/HOL: Corecursion

- Recursion: always eventually reduces an argument
- Corecursion: always eventually produces something
- Corec:

Isabelle/HOL: (Co)inductive Predicates

- Inductive predicate
  - Finite number of introduction rule applications

Isabelle/HOL: (Co)inductive Predicates

- Inductive predicate
  - Finite number of introduction rule applications
- Coinductive predicate
  - Infinite number of introduction rule applications

Isabelle/HOL: (Co)inductive Predicates

- Inductive predicate
  - Finite number of introduction rule applications
- Coinductive predicate
  - Infinite number of introduction rule applications
- Coinduction principle

Conclusion

Conclusion

- Isabelle/HOL has a good tool set to formalize and reason about stream processing:
  - Codatatypes, coinductive predicates, corecursion with friends, reasoning up to friends (congruence),
    - Coinduction up to congruence principle is automatically derived for codatatypes (but not for coinductive principles)
- Next step: Feedback loop

Questions, comments and suggestions