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

14/05/2025

Motivation

Motivation

Context:

- Stream Processing: programs that compute (possibly) unbounded sequences of data (streams)
- A common problem in the industry
- Frameworks:
  Apache Flink, Apache Samza, Apache Spark, Google Cloud Dataflow, and Timely Dataflow
- Why use frameworks?
  - Highly Parallel
  - Low latency (output as soon as possible)
  - Incremental computing (re-uses previous computations)

Motivation

Context:

- Stream Processing: programs that compute (possibly) unbounded sequences of data (streams)
- A common problem in the industry
- Frameworks:
  Apache Flink, Apache Samza, Apache Spark, Google Cloud Dataflow, and Timely Dataflow

Cloud DataFlow Flink

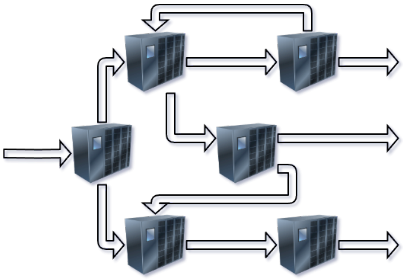
- Why use frameworks?
  - Highly Parallel
  - Low latency (output as soon as possible)
  - Incremental computing (re-uses previous computations)

Our goal:

Mechanically Verify Timely Dataflow algorithms

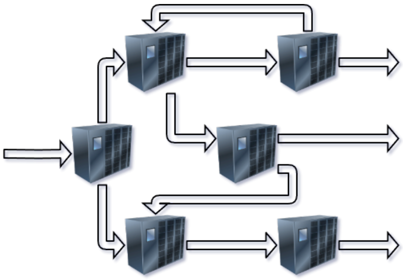
A Good Foundation

A Good Foundation



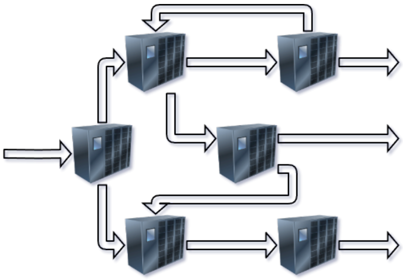
- Nondeterministic Asynchronous Dataflow

A Good Foundation



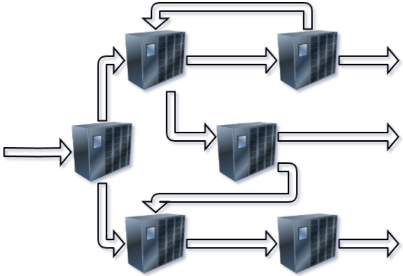
- Nondeterministic Asynchronous Dataflow
  - Dataflow: Directed graph of interconnected operators

A Good Foundation



- Nondeterministic Asynchronous Dataflow
  - Dataflow: Directed graph of interconnected operators
  - Asynchronous:
    - Operators execute independently: processes without an orchestrator
    - Operators can freely communicate with the network (read/write); do silent computation steps
    - Networks are unbounded FIFO queues

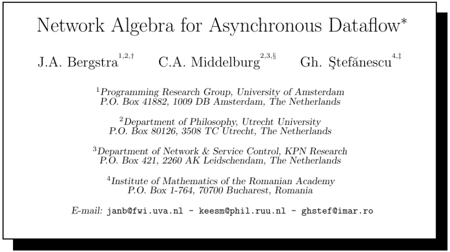
A Good Foundation



- Nondeterministic Asynchronous Dataflow
  - Dataflow: Directed graph of interconnected operators
  - Asynchronous:
    - Operators execute independently: processes without an orchestrator
    - Operators can freely communicate with the network (read/write); do silent computation steps
    - Networks are unbounded FIFO queues
  - Nondeterministic:
    - Operators can make nondeterministic choices
    - Operators are relations between inputs and outputs sequences

The Algebra for Nondeterministic Asynchronous Dataflow

- Bergstra et al. presents an algebra for Nondeterministic Asynchronous Dataflow
- Primitives:
  sequential and parallel composition; feedback loop...
- The 52 axioms
- An process calculus instance



Network Algebra for Asynchronous Dataflow*

J.A. Bergstra[1,2,†] C.A. Middelburg[2,3,§] Gh. Ştefănescu[4,‡]

[1]Programming Research Group, University of Amsterdam
P.O. Box 41882, 1009 DB Amsterdam, The Netherlands

[2]Department of Philosophy, Utrecht University
P.O. Box 80126, 3508 TC Utrecht, The Netherlands

[3]Department of Network & Service Control, KPN Research
P.O. Box 421, 2260 AK Leidschendam, The Netherlands

[4]Institute of Mathematics of the Romanian Academy
P.O. Box 1-764, 70700 Bucharest, Romania

E-mail: janb@fwi.uva.nl - keesm@phil.ruu.nl - ghstef@imar.ro

Isabelle/HOL Preliminaries

Isabelle/HOL

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

Isabelle/HOL

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

- Isabelle/HOL: Isabelle's flavor of HOL

Isabelle/HOL

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

- Isabelle/HOL: Isabelle's flavor of HOL

Why Isabelle/HOL?

- Codatatypes: (possibly) infinite data structures (e.g., lazy lists, streams)
- Corecursion: always eventually produces some codatatype constructor
- Coinductive predicate: infinite number of introduction rule applications
- Coinduction: reason about coinductive predicates

Operators as a Codatatype

Operators

$$
\begin{array}{l}
\textbf{\textit{codatatype}}\ (\mathrm{inputs}\colon 'i, \mathrm{outputs}\colon 'o, 'd)\ op = \\
\quad \mathrm{Read}\ 'i\ ('d \Rightarrow ('i, 'o, 'd)\ op) \mid \mathrm{Write}\ (('i, 'o, 'd)\ op)\ 'o\ 'd \\
\quad \mathrm{Silent}\ ('i, 'o, 'd)\ op \mid \mathrm{Choice}\ (('i, 'o, 'd)\ op)\ cset
\end{array}
$$

- Codatatypes

Examples

- Codatatypes

Operators Equivalences: Motivation

- foo

Operators Equivalences: Strong Bisimilarity

- foo

Operators Equivalences: Weak Bisimilarity

- foo

Asynchronous Dataflow Operators

Buffer Infrastructure

- foo

Asynchronous Dataflow Properties

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