Design representations

Data-flow Process Networks

Kris Kuchcinski
Krzysztof.Kuchcinski@cs.lth.se

Why Computational Models?

- To formalize the model of a system.
- To be able formally analyze, estimate parameters, verify and synthesize the system.
- To support specify-explore-refine scenario.
- Mathematical (assertions or properties) vs. constructive model (computational procedure; called executable also).
- ...

Why Specialized Models

- General models (Turing-complete) are too powerful and therefore difficult to handle.
- Different models provide different properties.
- Some problems may be undecidable in powerful models.
- Complexity of analysis algorithms for general models.
- There exist powerful models for particular application domains.
- Division between data and control flow models.

Why Specialized Models (cont'd)

- Most of the models of computations are sufficiently expressive to subsume most of other models
but
- this fails to acknowledge the strength and weakness of each model.
- For example, rendezvous is very good in resource management but very awkward for loosely coupled data-oriented computations.

Data vs. Control Flow Models

- Fuzzy distinction, yet useful for:
  - specification (language, model, ...)
  - synthesis (scheduling, optimization, ...)
  - validation (simulation, formal verification, ...)
- Rough classification:
  - control:
    - don’t know when data arrive (quick reaction)
    - time of arrival often matters more than value
  - data:
    - data arrive in regular streams (samples)
    - value matters most

"The purpose of models is not to fit the data but to sharpen the questions."

Samuel Karlin, (1923 - )
**Data vs. Control Flow Models (cont'd)**

- Specification, synthesis and validation methods emphasize:
  - for control:
    - event/reaction relation
    - response time
    - (Real Time scheduling for deadline satisfaction)
    - priority among events and processes
  - for data:
    - functional dependency between input and output
    - memory/time efficiency
    - (Dataflow scheduling for efficient pipelining)
    - all events and processes are equal

**Dataflow Models — Application Areas**

- Signal processing (including image processing) applications.
- Commercial systems using dataflow process networks:
  - SPW (Signal Processing Worksystem)- Alta Group of Cadence,
  - COSSAP — Synopsys,
  - DSP Station — Mentor Graphics,
  - MATLAB,
  - ...

**Second-Order Filter Section**

\[
x + dx + m_1 = o_1 \]

**Second-Order Filter Section (cont'd)**

\[
x + dx + m_2 + m_3 + m_4 = o_1 \]

**Data-flow Model Intuition**

- partial order
- multiple tokens
- multiple context
- "interleaving" computations
- feedback loops
- operation types (functions, etc.)

**Process Networks**

- communicating processes with directed flow.
- communication: token “stream” between two processes
- process: operations on tokens
- host language: process description
- coordination language: network description
Kahn process networks (1958)
- special class of process networks.
- stream is FIFO with unbounded capacity.
- process:
  - destructive read ("consumption") at process start,
  - non-destructive write ("production") at process end,
  - blocking read — process only executed if data available,
  - non-blocking write.

Kahn process networks example
\[ [1, 2, 3, 2, ...] \]
\[ [0, 1] \]

Kahn process networks — formalism
Sequence (a stream) \( X = [x_1, x_2, ...] \)
Prefix ordering \( [x_1, x_2] \subseteq [x_1, x_2, x_3] \)
Increasing chain of seq. \( X = (x_0, x_1, ...) \) where \( X_0 \subseteq X_1 \)
Least upper bound \( \text{lub} X \subseteq Y \) where \( X_i \subseteq Y \) for all \( X_i \subseteq X \)

Continuous process \( F \) (lub \( \chi \)) = lub \( F (\chi) \)

(Least) Upper Bound
- Given a subset \( Y \) of \( S \), an upper bound of \( Y \) is an element \( z \) of \( S \) such that \( z \) is larger than all elements of \( Y \)
- Consider now the set \( Z \) (subset of \( S \)) of all the upper bounds of \( Y \)
- If \( Z \) has a least element \( u \), then \( u \) is called the least upper bound (lub) of \( Y \)
- The least upper bound, if it exists, is unique
- Note: \( u \) might not be in \( Y \) (if it is, then it is the largest value of \( Y \))

Kahn process networks — formalism
p-tuple of sequences \( X = (X_0, X_1, ..., X_p) \subseteq S^p \)
ordered set of seq. \( X \subseteq X' \) if \( X_i \subseteq X'_i \) for each \( i \)
set of p-tuple of sequences \( \chi = (X_0, X_1, ...) \)
functional process \( F : S^p \rightarrow S^q \)

Continuous process \( F \) (lub \( \chi \)) = lub \( F (\chi) \)

Kahn process networks — formalism
- Monotonicity
  \( X \subseteq X' \Rightarrow F (X) \subseteq F (X') \)
- It can be proved that a continuous process is monotonous
  - given a part of the input sequence it may be possible to compute part of the output sequence.
Least Fixed Point semantics

- Let $X$ be the set of all sequences.
- A network is a mapping $F$ from the sequences to the sequences (where $I$ represents the input sequence): $X = F(X, I)$
- The behavior of the network is defined as the unique least fixed point of the equation \( LFP \).
- If $F$ is continuous then the least fixed point exists
  \[ \text{LFP} = \text{LUB} \left( \{ F^n (\bot, I) : n \geq 0 \} \right) \]

Non-monotonic processes

- "Canonical" non-monotonic process: fair merge

\[ \begin{align*}
\{ x_1, x_2, x_3, \ldots \} & \rightarrow \{ y_1, y_2, y_3, \ldots \} \\
\{ x_2, x_3, \ldots \} & \rightarrow \{ y_1, x_1, y_2, \ldots \} \\
\{ x_1, x_2, \ldots \} & \rightarrow \{ y_1, y_2, y_3, \ldots \} \\
\end{align*} \]

- In the previous example, we have:
  \( (\{ x_1, x_2 \}, \{ y_1, y_2, y_3, \ldots \}) \subseteq (\{ x_1, x_2, x_3, \ldots \}, \{ y_1, y_2, y_3, \ldots \}) \)
  but
  \( (\{ y_1, y_2, y_3, \ldots \}, \{ x_1, x_2, x_3, \ldots \}) \)
  are incomparable.
- The process is not monotonic (needs prediction of the future to be really fair).
- The least fixed point may not exist.

Dataflow Networks

- A data-flow network is a collection of functional nodes which are connected and communicate over unbounded FIFO queues.
- Nodes are commonly called actors.
- Data that are communicated over the queues are commonly called tokens.
- Each Khan’s process becomes an actor with define firing rule and function.

Intuitive semantics

- Actors (often stateless) perform computation.
- Unbounded FIFO’s perform communication via sequences of tokens carrying values
  - integer, float, fixed point, matrix of integer, float, fixed point, image of pixels.
- State implemented as self-loop.
- Determinacy (based on Khan’s results):
  - unique output sequences given unique input sequences,
  - Sufficient condition: blocking read (actor cannot test input queues for emptiness).

Intuitive semantics

- At each time, one actor is fired.
- When firing, an actor consumes input tokens and produces output tokens.
- Actors can be fired only if there are enough tokens in the input queues.
Firing Rules

- An actor with $p \geq 1$ input streams can have $N$ firing rules $R = \{R_1, R_2, \ldots, R_N\}$.
- The actor can fire iff one or more firing rules is satisfied.
- Typical firing rule $R_i = \{[*], [*]\}$ meaning that the actor fires iff each of two inputs have at least one token.

Continuous Data-flow Networks

- A sufficient conditions for data-flow process to be continuous:
  - each actor firing has to be functional — lacks side effects and output tokens are a function of input tokens.
  - set of firing rules has to be sequential — firing rules can be tested in a pre-defined order using only blocking reads.

An Example — FIR filter

- single input sequence $i(n)$
- single output sequence $o(n)$
- $o(n) = c_1 \cdot i(n) + c_2 \cdot i(n-1)$

Examples of Data-flow actors

SDF: Synchronous (Static) Data-flow
- fixed input and output tokens

BDF: Boolean Data-flow
- control token determines consumed and produced tokens

Sequential Firing Rules

1. Find an input such that $[*] \subseteq R_{i,j}$ for all $i = 1 \ldots N$. That is, find an input such that all the firing rules require at least one token from that input $(j)$. If no such input exists, fail.
2. For the choice of input $(j)$, divide the firing rules into subsets, one for each specific token value mentioned in the first position of $R_{i,j}$ for any $i = 1 \ldots N$. If $R_{i,j} = \{[*], \ldots\}$, then the firing rule should appear in all such subsets.
3. Remove the first element of $R_{i,j}$ for all $i = 1 \ldots N$.
4. If all subsets have empty firing rules, then succeed. Otherwise, repeat these four steps for any subset with any non-empty firing rules.
Sequential Firing Rules

Selector Example

\[ R_1 = \{ [\cdot], \bot, [T] \} \quad , \quad R_2 = \{ \bot, [\cdot], \{F\} \} \]

- \( j = 3 \)
- \( \{R_1\} \) and \( \{R_2\} \)
- \( R_1 = \{ [\cdot], \bot, \bot \} \quad , \quad R_2 = \{ \bot, [\cdot], \bot \} \)
- Remove \( R_{11} \) and \( R_{22} \)
- \( R_1 = \{ \bot, \bot, \bot \} \quad , \quad R_2 = \{ \bot, \bot, \bot \} \)
- Empty firing rules \( \Rightarrow \) sequential

Sequential Firing Rules

Merger Example

\[ R_1 = \{ [\cdot], \bot \} \quad , \quad R_2 = \{ \bot, [\cdot] \} \]

- Fails immediately in step 1 \( \Rightarrow \) non-sequential.

Properties of Dataflow Networks for Design

- Static scheduling is possible for static networks.
- Different trade-offs such as code size, buffer size, pipelining are possible.
- Static scheduling can be used for simulator generator, DSP code generation and HW synthesis.
- Modeling power is limited but ...
- Semi-static scheduling of if-then-else and loops.

Properties of Dataflow Networks

- It has been shown that the addition of only select actor and switch actor to synchronous data-flow model is sufficient to make it Turing complete.

Summary

- Advantages:
  - Easy to use (graphical languages)
  - Powerful algorithms for
    - verification (fast behavioral simulation)
    - synthesis (scheduling and allocation)
  - Explicit concurrency
- Disadvantages:
  - Efficient synthesis only for restricted models
  - Cannot describe reactive control (blocking read)

Literature