Chapter 6

Concurrent Models of Computation

Hongwei Zhang

http://www.cs.wayne.edu/~hzhang/

Ack.: this lecture is prepared in part based on slides of Lee, Sangiovanni-Vincentelli, Seshia, Philip Asare.
Concurreny

Things happening (conceptually) simultaneously

Motors
Serial Port
Audio

Must respond to pedals AND steering
Model of Computation (MoC)

Three sets of rules:

- What is a component?
  - How is it allowed to behave?
- What is the concurrency mechanism?
- What is the communication mechanism?
Why do we care?

Used in
- Analysis and verification
- Execution of computer programs
  - Programming languages have these
- Simulation

Subtleties in semantics have consequences
- Will produce different behaviors for same ‘model’
Our Components: Actors

Ports (input, output) with types

Execution Actions

- How inputs are used to produce outputs
- How state changes occur

Communication Relation
Actors Examples

Helicopter

Scale

Integrator

\[ x \rightarrow a \rightarrow y \rightarrow x' \rightarrow \int \rightarrow y' \]
Other Pieces

Environment

- Must determine if model is well-formed (can be executed)
- Must come up with ‘schedule’ of execution
  - Decide when actors can react
  - Call each actor to react
  - Move data from one actor to another
  - Do other house keeping
Outline

Synchronous-reactive (SR) models
Data flow models of computation
Timed models of computation
Outline

Synchronous-reactive (SR) models
Data flow models of computation
Timed models of computation
Recall: Actor Model for State Machines

Expose inputs and outputs, enabling composition:
Recall: Actor Model of Continuous-Time Systems

A system is a function that accepts an input signal and yields an output signal.

The domain and range of the system function are sets of signals, which themselves are functions.

Parameters may affect the definition of the function $S$. 

$S: X \rightarrow Y$

$X = Y = (\mathbb{R} \rightarrow \mathbb{R})$
Angular velocity appears on both sides. The semantics (meaning) of the model is the solution to this equation.

\[ \dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t T_y(\tau) d\tau \]

\[ = \dot{\theta}_y(0) + \frac{K}{I_{yy}} \int_0^t (\psi(\tau) - \dot{\theta}_y(\tau)) d\tau \]

We will now generalize this notion of composition.
Side-by-Side Composition

Synchronous composition: the machines react simultaneously and instantaneously.

$$\text{(States, Inputs, Outputs, update, initialState)}$$

$$\text{(States}_A, \text{Inputs}_A, \text{Outputs}_A, \text{update}_A, \text{initialState}_A)$$

$$\text{(States}_B, \text{Inputs}_B, \text{Outputs}_B, \text{update}_B, \text{initialState}_B)$$
Cascade Composition

Synchronous composition: the machines react simultaneously and instantaneously, *despite the apparent causal relationship*!
Synchronous Composition: Reactions are *Simultaneous* and *Instantaneous*

Consider a cascade composition as follows:
Synchronous Composition:
Reactions are *Simultaneous* and *Instantaneous*

In this model, we must not think of machine A as reacting before machine B. If it did, the unreachable states would not be unreachable.

\[ S_C = S_A \times S_B \]
Feedback Composition

(turns out everything can be viewed as feedback composition…

[Diagram showing the composition of feedback systems]
Observation: Any Composition is a Feedback Composition

The behavior of the system is a “fixed point.”
We seek an $s \in S^N$ that satisfies $F(s) = s$.

Such an $s$ is called a fixed point.

We would like the fixed point to exist and be unique. And we would like a constructive procedure to find it.

It is the behavior of the system.
Data Types
As with any connection, we require compatible data types:

\[ V_y \subseteq V_x \]

Then the signal on the feedback loop is a function

\[ s : \mathbb{N} \rightarrow V_y \cup \{\text{absent}\} \]

Then we seek \( s \) such that

\[ F(s) = s \]

where \( F \) is the actor function, which for determinate systems has form

\[ F : (\mathbb{N} \rightarrow V_x \cup \{\text{absent}\}) \rightarrow (\mathbb{N} \rightarrow V_y \cup \{\text{absent}\}) \]
Firing Functions

With synchronous composition of determinate state machines, we can break this down by reaction. At the $n$-th reaction, there is a (state-dependent) function

$$f(n) : V_x \cup \{absent\} \rightarrow V_y \cup \{absent\}$$

such that

$$s(n) = (f(n))(s(n))$$

This too is a fixed point.
Well-Formed Feedback

At the $n$-th reaction, we seek $s(n) \in V_y \cup \{\text{absent}\}$ such that

$$s(n) = (f(n))(s(n))$$

There are two potential problems:

1. It does not exist.
2. It is not unique.

In either case, we call the system **ill formed**. Otherwise, it is **well formed**.

Note that if a state is not reachable, then it is irrelevant to determining whether the machine is well formed.
In state \textit{s1}, we get the unique \( s(n) = \text{absent} \).
In state \textit{s2}, we get the unique \( s(n) = \text{present} \).
Therefore, \( s \) alternates between \text{absent} and \text{present}.
Well-Formed Example: Composite Machine

**Input:** $a$: pure

**Output:** $b$: pure

- $a / a / b$
- $\neg a / b$

**Output:**
- $b$: pure

- $true / b$
- $true /$

- $s$

- $s1$
- $s2$

- $b$
- $s$
Ill-Formed Example 1 (Existence)

In state $s_1$, we get the unique $s(n) = \text{absent}$.
In state $s_2$, there is no fixed point.
Since state $s_2$ is reachable, this composition is ill formed.
Ill-Formed Example 2 (Uniqueness)

In state $s1$, both $s(n) = \text{absent}$ and $s(n) = \text{present}$ are fixed points. In state $s2$, we get the unique $s(n) = \text{present}$. Since state $s1$ is reachable, this composition is ill formed.
Constructive Semantics: Single Signal

For each state $i$ at the $n$-th reaction,

1. Start with $s(n)$ unknown.

2. Determine as much as you can about $(f(n))(s(n))$.

3. If $s(n)$ becomes known (whether it is present, and if it is not pure, what its value is), then we have a unique fixed point.

A state machine for which this procedure works is said to be constructive.
For the above constructive machine, in state $s1$, we can immediately determine that the machine *must not* produce an output. Therefore, we can immediately conclude that the output is *absent*, even though the input is unknown.

In state $s2$, we can immediately determine that the machine *must* produce an output, so we can immediately conclude that the output is *present*. 
In state \( s1 \), if the input is unknown, we cannot immediately tell what the output will be. We have to try all the possible values for the input to determine that in fact \( s(n) = \textit{absent} \) for all \( n \).

For non-constructive machines, we are forced to do \textit{exhaustive search}. This is only possible if the data types are finite, and is only practical if the data types are small.
Constructive Semantics: Multiple Signals

1. Start with $s_1(n), \cdots, s_N(n)$ unknown.

2. Determine as much as you can about $(f(n))(s_1(n), \cdots, s_N(n))$.

3. Using new information about $s_1(n), \cdots, s_N(n)$, repeat step (2) until no information is obtained.

4. If $s_1(n), \cdots, s_N(n)$ all become known, then we have a unique fixed point and a constructive machine.

A state machine for which this procedure works is said to be constructive.
Constructive Semantics: Multiple Actors

Procedure is the same.
Constructive Semantics: Arbitrary Structure

Procedure is the same.

A state machine language with constructive semantics will reject all compositions that in any iteration fail to make all signals known.

Such a language rejects some well-formed compositions.
Synchronous Reactive (SR) Models: Summary

+ Strong formal properties that yield quite effectively to formal analysis and verification techniques
  - The emphasis of synchronous composition is on *determinate* and *analyzable* concurrency.
  - Although there are subtleties with synchronous programs, all constructive synchronous programs have a unique and well-defined meaning.
  - Automated tools can systematically explore *all* possible behaviors.

- In SR MoC, actors react simultaneously and instantaneously, at least conceptually. Achieving this with realistic computation requires tight coordination of the computation.
Outline

Synchronous-reactive (SR) models
Data flow models of computation
Timed models of computation
Dataflow MoC

Much more asynchronous than SR

- Reactions may occur simultaneously, or they may not. Whether they do or do not is not an essential part of the semantics.

When reactions are dependent on one another, the dependence is due to the flow of data, rather than to the synchrony of events.

- If a reaction of actor A requires data produced by a reaction of actor B, then the reaction of A must occur after the reaction of B.

A MoC where such data dependencies are the key constraints on reactions is called a dataflow model of computation
Dataflow

Tokens are used

Actor has:

- Firing function: how output tokens are produced from input tokens
- Firing rule: when it can fire based on the number of input tokens

Tokens must be buffered until actor is ready to consume;

Two key related issues for unbounded execution:

- Absence of deadlock
- Bounded buffers
Synchronous/Static Dataflow (SDF)

A constrained form of dataflow where, for each actor, every firing consumes a fixed number of input tokens on each input port and produces a fixed number of output tokens on each output port.

Input quantity defines firing rule

- A produces 2 tokens each time it fires
- B consumes 1 token and produces 3 tokens each time it fires
Synchronous Dataflow (SDF)

Environment must solve balance equations

\[ q_A K = q_B L \]
\[ q_B M = q_C N \]

To find integers \( q_A, q_B, \) and \( q_C \)

Try it for \( K=3, L=2, M=2, N=3 \)
Other Variants of Dataflow MoC

SDF
- Both bounded buffers and deadlock are decidable for SDF models
- Not very expressive, e.g., not able to express conditional firing

Dynamic Dataflow (DDF)
- DDF actors can have multiple firing rules, and they are not constrained to produce the same number of output tokens on each firing
- Delay, Select, Switch actors
- Bounded buffers and deadlock are undecidable for DDF models; similar to goto semantics, thus not recommended

Structured Dataflow
- Higher-order actor (an actor having one or more models as parameters) enabling structured languages

Process Networks
Petri-Nets
Outline

Synchronous-reactive (SR) models
Data flow models of computation
Timed models of computation
Extending with Time

- Define ticks of the synchronous clock to occur at a *time* in some model of time.
- Signals between actors now consist of *time-stamped events*.

Timed MoC

Time-Triggered Models
- time-triggered architecture (TTA) by Kopetz & colleagues

Discrete Event Systems
- events are endowed with a time stamp

Continuous Time Systems

Useful for cyber-physical systems where time is an essential concept.
Summary

Synchronous-reactive (SR) models
Data flow models of computation
Timed models of computation
Assignment

Exercise #8
  ● Chapter 6: Exercises 2, 8

TinyExam #2