Concurrent State Machines

CS 447– Wireless Embedded Systems
Outline

• Overview
• Side-by-side synchronous
• Side-by-side asynchronous
• Shared variables
• Cascades composition
• General composition
Overview

• State machines are great way to model systems

• Problem: for interesting systems, number of states is large
Overview

• Time-honored practice in engineering: complicated systems describe as composition of simpler ones

Jargon:

• Syntax – rules of model notation
• Semantics – meaning of notation (see this later)
Overview

E.g., use an actor model for an extended state machine
• Two inputs
• Two outputs
• “don’t care” about internals
• Abstraction..
Overview

Two composition techniques:

• Concurrent composition (today’s lecture)

• Hierarchical composition (next lecture)
Overview

Concurrent composition

• Two or more state machines that react *simultaneously or independently*

• **Synchronous composition** – reactions are simultaneous
• **Asynchronous composition** – reactions are independent
Overview

Concurrent composition

• Begin w/ simplest case: side-by-side composition

• Composed state machines either:
  • Do not communicate directly
  • Communicate via \textit{shared variables}
  • Communicate via \textit{ports} (e.g., serial)
Outline

• Overview
• Side-by-side synchronous
• Side-by-side asynchronous
• Shared variables
• Cascades composition
• General composition
Side-by-Side Synchronous Composition

- Inputs and outputs are disjoint
- State machines do not communicate
- A: input $i_1$, output $o_1$
- B: input $i_2$, output $o_2$

- If A, B are extended state machines, then variables disjoint as well
Side-by-Side Synchronous Composition

- C: composition of two actors
- Two inputs \((i_1, i_2)\)
- Two outputs \((o_1, o_2)\)
Side-by-Side Synchronous Composition

Synchronous composition:
• Reaction of C is *simultaneous* reaction of A and B
Side-by-Side Synchronous Composition

What does this machine do?
Side-by-Side Synchronous Composition

C acts like an oscillator: output is “bababababa...”

Remember: A, B are simultaneous
Side-by-Side Synchronous Composition

Can machine C ever reach state \((s_1, s_4)\)? What about \((s_2, s_3)\)?
Side-by-Side Synchronous Composition

SbS synchronous machines are:

• Modular
• Deterministic
• Compositional
Side-by-Side Synchronous Composition

**Modular** – composition itself can be used as an atomic component
Side-by-Side Synchronous Composition

**Deterministic** – If A and B are both deterministic, C is as well
Side-by-Side Synchronous Composition

**Compositional** – property held by components is also property of composition

- E.g., determinism is a *compositional property*
Side-by-Side Synchronous Composition

NOTE: previous example can be drawn as single state machine
Outline

• Overview
• Side-by-side synchronous
• **Side-by-side asynchronous**
• Shared variables
• Cascades composition
• General composition
Side-by-Side Asynchronous Composition

Asynchronous composition:
• Components (A, B) react independently
Side-by-Side Asynchronous Composition

Semantics #1
• Reaction of C is reaction of A or B (just one)
• A, B never react simultaneously (they are interleaved)
• Choice of A vs. B is nondeterministic
Side-by-Side Asynchronous Composition

Semantics #2

• Reaction of C is reaction of A, B or both
• Choice is nondeterministic
Side-by-Side Asynchronous Composition

Previous example modeled as asynchronous SBS.

red arrows mark nondeterminism
Side-by-Side Asynchronous Composition

Under **semantics #1**
- C in \((s_1, s_3)\)
- C reacts
- C moves to \((s_1, s_4)\), emits b
- **OR:** C moves to \((s_2, s_3)\), emits \(\emptyset\)
Side-by-Side Asynchronous Composition

Under **semantics #2**

- C in \((s_1, s_3)\)
- C reacts
- Could also move to \((s_2, s_4)\)
- Could also output \(\emptyset b\) OR \(b \emptyset\)
Side-by-Side Asynchronous Composition

Take home point: **asynchronous composition** can be treacherous

- You must be clear about semantics
- E.g., A must always come before B
Outline

• Overview
• Side-by-side synchronous
• Side-by-side asynchronous
• Shared variables
• Cascades composition
• General composition
Shared Variables

• Extended state machines have \textit{local} variables
• Variables read / written during transitions

When composing state machines:
• Helpful to allow shared variables among machines
• Can be used to model interrupts and threads
Shared Variables

E.g., two servers receiving network requests

- Shared variable “pending” integer that counts jobs

- When request arrives to composition machine C, one of two servers chosen non-deterministically
  - Assumes asynchronous composition under semantics #1
  - I.e., mutual exclusion
Shared Variables

E.g., two servers receiving network requests

When request arrives:
• If server idle, proceeds to serve request
• If server busy:
  • Coincidentally finish serving current, output done, proceed to new
  -OR-
  • Increment count of pending
  • Continue serving current
**Shared Variables**

**shared variable:** `pending`: int

**input:** `request`: pure

**outputs:** `doneA`, `doneB`: pure

**A**

- **input:** `request`: pure
- **output:** `done`: pure

- `\neg request \land pending = 0 \land done`
- `request \land pending > 0 \land done`
- `pending := pending - 1`
- `request /`
- `request /`
- `pending := pending - 1`
- `request /`
- `pending := pending + 1`

**B**

- **input:** `request`: pure
- **output:** `done`: pure

- `\neg request \land pending = 0 \land done`
- `request \land pending > 0 \land done`
- `pending := pending - 1`
- `request /`
- `request /`
- `pending := pending + 1`
- `request /`
- `pending := pending - 1`
Shared Variables

Example shows subtleties of concurrent systems (1)

- **Interleaving semantics (#1):** access to shared “pending” are modeled as atomic

- This can be challenging to do in practice
Shared Variables

Example shows subtleties of concurrent systems (2)

- **Semantics #1** seems reasonable
- But could lead to idle machines...

E.g., what if machine A is serving, B is idle, request arrives
- But A is picked *non deterministically* (=> pending++)
- B never used...

In Semantics #1, this is possible.
Shared Variables

Example shows subtleties of concurrent systems (3)

• Semantics #2 also reasonable (“simultaneous” execution)

• But requires updating “pending” in an **atomic manner**

• This can be more challenging to implement.
Shared Variables

What about shared variables in synchronous composition?

• Write from A before reading from B?
• Read from B before writing from A?
• Must more challenging!

• Must use interleaved semantics (i.e., scheduling)
Outline

- Overview
- Side-by-side synchronous
- Side-by-side asynchronous
- Shared variables
- Cascades composition
- General composition
Cascades Composition (aka Serial Composition)

• Two machines (A, B) composed in series:

• Output of A is input to B
Cascades Composition (aka Serial Composition)

- Consider only *synchronous composition*
- Asynchronous composition would require buffering
Cascades Composition (aka Serial Composition)

- Reaction of C is reaction of A and B
- A reacts first, then B
- We assume this occurs instantly (in zero time)
- Two reactions (A, B) are *simultaneous* and *instantaneous*
Cascades Composition (aka Serial Composition)

Example (remember, transitions are simultaneous!)
• When does machine C output “c”? 

![Diagram of cascades composition](image)
Equivalent machine:
Cascades Composition (aka Serial Composition)

Another example: modify traffic light model to include pedestrian light as well.

- Cascades composition: turn on RED light => turn on ”walk”
  - Causal..
Outline

• Overview
• Side-by-side synchronous
• Side-by-side asynchronous
• Shared variables
• Cascades composition
• General composition
General Composition

• Side-by-side and cascades composition provide basic building blocks.
• These models can be combined: e.g.,
General Composition

• A1, A3 are side-by-side (define B)
• B, A2 are cascades composition in opposite order (feedback)
• Question: which machine should react first?