System Specification, Verification and Synthesis (SSVS) – CS 4830/7485, Fall 2019

5: Formal System Modeling:
System Composition
Synchronous Composition

Stavros Tripakis

Northeastern University
Khoury College of Computer Sciences

Outline

- System composition
- Synchronous composition

SYSTEM COMPOSITION

Reminder: what is a system?

- Our definition so far:

$$\text{system} = \text{state} + \text{dynamics}$$

- But this definition is too **monolithic**
- Most systems are structured **hierarchically**: system, subsystems, subsubsystems, ...
- Examples:
  - The human body is made of organs, which are made of cells, which are made of ...
  - Matter is made of molecules, which are made of atoms, which are made of particles, which are made of ...
  - A society is made of people interacting with each other
  - In a highway there are several vehicles traveling
  - A digital circuit is made of gates, flip-flops, wires, ...
  - A piece of software is made of functions, classes, libraries, threads, ...
  - A distributed system is made of nodes communicating via a network
  - ...

Systems: non-monolithic definition:

- **System**: atomic or composite system
- **Atomic** system: state + dynamics (+ inputs/outputs)
- **Composite** system: set of (sub)systems + composition
- **Dynamics**: rules defining how state evolves in time
- **Composition**: rules defining how subsystems interact

Systems: non-monolithic definition:

- **System**: atomic or composite system
- **Atomic** system: state + dynamics (+ inputs/outputs)
- **Composite** system: set of (sub)systems + composition
- **Dynamics**: rules defining how state evolves in time
- **Composition**: rules defining how subsystems interact

Composition typically ignored in classic system theory.

System composition paradigms

Two major paradigms:

- **Synchronous** composition:
  - All sub-systems move together: in "lock-step".
  - Application: synchronous circuits, embedded control systems, ...
- **Asynchronous** composition:
  - Each sub-system moves "at its own pace".
  - **Interleaving**: only one sub-system makes a move at a time.
  - Applications: concurrent software (processes, threads, ...), non-synchronized distributed systems, asynchronous circuits, ...

System composition paradigms

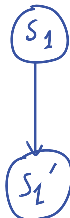
Two major paradigms:

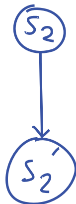
- **Synchronous** composition:
  - All sub-systems move together: in "lock-step".
  - Application: synchronous circuits, embedded control systems, ...
- **Asynchronous** composition:
  - Each sub-system moves "at its own pace".
  - **Interleaving**: only one sub-system makes a move at a time.
  - Applications: concurrent software (processes, threads, ...), non-synchronized distributed systems, asynchronous circuits, ...

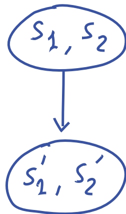
Common principle:

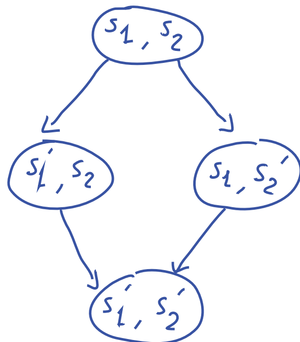
- The state-space of the composite system is the (cartesian) **product** of the state-spaces of its components (subsystems).

Synchronous and asynchronous composition on transition systems: intuitive drawing









Example: synchronous composition in nuXmv

```
MODULE main
VAR
  bit0 : counter_cell(TRUE);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);
SPEC
  AG AF bit2.carry_out

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := FALSE;
  next(value) := value xor carry_in;
DEFINE
  carry_out := value & carry_in;
```

SMV model counter.smv taken from `http://nusmv.fbk.eu/examples/examples.html`

Example: synchronous composition in nuXmv

```
MODULE main
VAR
  bit0 : counter_cell(TRUE);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);
SPEC
  AG AF bit2.carry_out

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := FALSE;
  next(value) := value xor carry_in;
DEFINE
  carry_out := value & carry_in;
```

Simulate it at home!

SMV model counter.smv taken from `http://nusmv.fbk.eu/examples/examples.html`

Example: asynchronous composition in Spin

```
// a small example spin model
// Peterson's solution to the mutual exclusion problem (1981)

bool turn, flag[2];         // the shared variables, booleans
byte ncrit;                 // nr of procs in critical section

active [2] proctype user()  // two processes
{
    assert(_pid == 0 || _pid == 1);
again:
    flag[_pid] = 1;
    turn = _pid;
    (flag[1 - _pid] == 0 || turn == 1 - _pid);

    ncrit++;
    assert(ncrit == 1);     // critical section
    ncrit--;

    flag[_pid] = 0;
    goto again
}
// analysis:
// $ spin -run peterson.pml
```

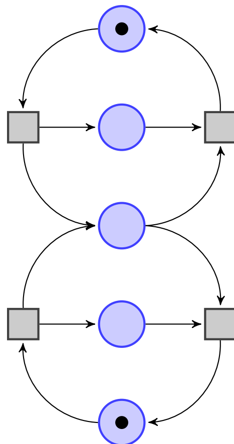
Example: asynchronous composition in Spin

```
// a small example spin model
// Peterson's solution to the mutual exclusion problem (1981)

bool turn, flag[2];         // the shared variables, booleans
byte ncrit;                 // nr of procs in critical section

active [2] proctype user()  // two processes
{
    assert(_pid == 0 || _pid == 1);
again:
    flag[_pid] = 1;
    turn = _pid;
    (flag[1 - _pid] == 0 || turn == 1 - _pid);

    ncrit++;
    assert(ncrit == 1);     // critical section
    ncrit--;

    flag[_pid] = 0;
    goto again
}
// analysis:
// $ spin -run peterson.pml
```

Simulate it at home!

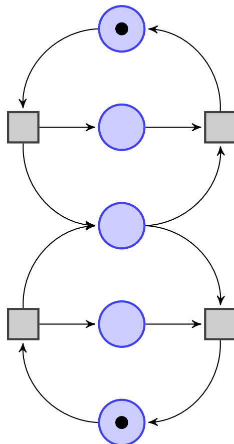
Example: Petri nets

Semantics usually based on interleaving (asynchronous).



Example: Petri nets

Semantics usually based on interleaving (asynchronous).



Simulate it at home!

Naive formal definitions of synchronous and asynchronous composition on transition systems

Consider two transition systems TS_1 and TS_2 with $TS_i = (S_i, S_0^i, R_i)$.

The synchronous composition of TS_1 and TS_2 is a new transition system

$$TS_1 \times TS_2 = (S_1 \times S_2, S_0^1 \times S_0^2, R_{sync})$$

where

$$R_{sync} = \{\big((s_1, s_2), (s_1', s_2')\big) \mid (s_1, s_1') \in R_1 \wedge (s_2, s_2') \in R_2\}$$

The asynchronous composition of TS_1 and TS_2 is a new transition system

$$TS_1 \| TS_2 = (S_1 \times S_2, S_0^1 \times S_0^2, R_{async})$$

where

$$R_{async} = \{\big((s_1, s_2), (s_1', s_2)\big) \mid (s_1, s_1') \in R_1\} \cup \{\big((s_1, s_2), (s_1, s_2')\big) \mid (s_2, s_2') \in R_2\}$$

Naive formal definitions of synchronous and asynchronous composition on transition systems

Consider two transition systems TS_1 and TS_2 with $TS_i = (S_i, S_0^i, R_i)$.

The synchronous composition of TS_1 and TS_2 is a new transition system

$$TS_1 \times TS_2 = (S_1 \times S_2, S_0^1 \times S_0^2, R_{sync})$$

where

$$R_{sync} = \{\big((s_1, s_2), (s_1', s_2')\big) \mid (s_1, s_1') \in R_1 \land (s_2, s_2') \in R_2\}$$

The asynchronous composition of TS_1 and TS_2 is a new transition system

$$TS_1 \| TS_2 = (S_1 \times S_2, S_0^1 \times S_0^2, R_{async})$$

where

$$R_{async} = \{\big((s_1, s_2), (s_1', s_2)\big) \mid (s_1, s_1') \in R_1\} \cup \{\big((s_1, s_2), (s_1, s_2')\big) \mid (s_2, s_2') \in R_2\}$$

Why are these definitions "naive"?

Naive formal definitions of synchronous and asynchronous composition on transition systems

Consider two transition systems TS_1 and TS_2 with $TS_i = (S_i, S_0^i, R_i)$.

The synchronous composition of TS_1 and TS_2 is a new transition system

$$TS_1 \times TS_2 = (S_1 \times S_2, S_0^1 \times S_0^2, R_{sync})$$

where

$$R_{sync} = \{\big((s_1, s_2), (s_1', s_2')\big) \mid (s_1, s_1') \in R_1 \wedge (s_2, s_2') \in R_2\}$$

The asynchronous composition of TS_1 and TS_2 is a new transition system

$$TS_1 \| TS_2 = (S_1 \times S_2, S_0^1 \times S_0^2, R_{async})$$

where

$$R_{async} = \{\big((s_1, s_2), (s_1', s_2)\big) \mid (s_1, s_1') \in R_1\} \cup \{\big((s_1, s_2), (s_1, s_2')\big) \mid (s_2, s_2') \in R_2\}$$

Why are these definitions "naive"?
Because they don't model interaction (inputs, outputs, shared vars, ...).

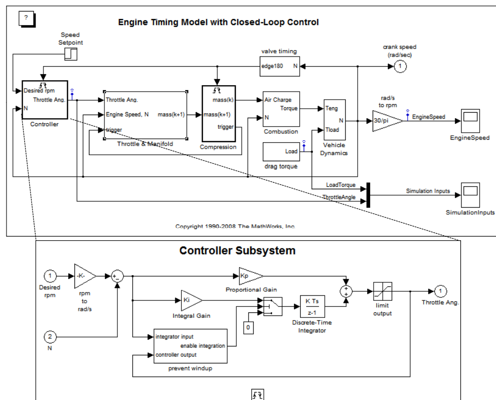
In practice

- Every model-checker has their own language with its own composition features and semantics.
  - Shared variables
  - Inputs and outputs
  - Communication channels
  - Labels and rendez-vous synchronization
  - ...
- The semantics is defined at the level of that language: given a program in that language, that program defines a (big) transition system (for the entire product system).
- Direct compositions at the level of transition systems (LTSs and Kripke structures) also exist, e.g., [Milner, 1980, Hoare, 1985].
- One needs to be careful with composition, as there are several subtleties: we illustrate some in the next few slides.
- Many more things to say about composition, not enough time. We will return to the topic later when we talk about compositionality.

SYNCHRONOUS COMPOSITION

Block Diagrams

We should be able to express any such diagram using formal composition operators.

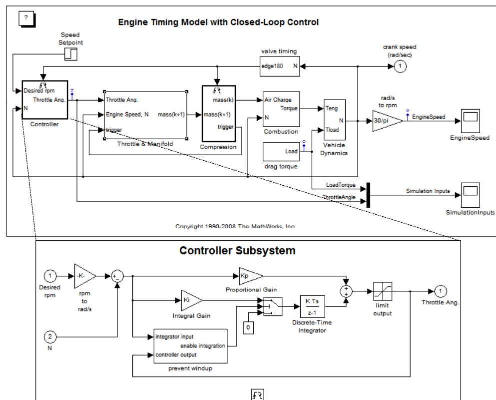


Block Diagrams

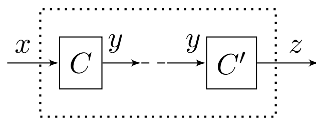
We should be able to express any such diagram using formal composition operators.

Basic primitives:

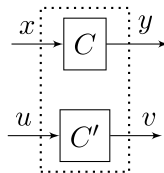
- Serial composition.
- Parallel composition.
- Feedback composition.



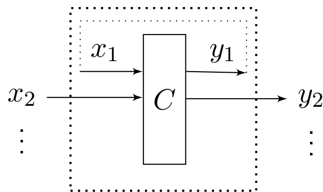
Three basic composition primitives



Serial composition

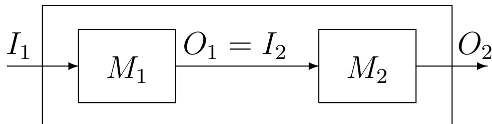


Parallel composition



Feedback composition

Synchronous serial composition of FSMs: formalization



Given two Mealy machines M_1 and M_2 with

$$M_i = (I_i, O_i, S_i, s_0^i, \delta_i, \lambda_i)$$

such that $O_1 = I_2$, the serial synchronous composition of M_1 and M_2 is a new Mealy machine

$$M = (I_1, O_2, S_1 \times S_2, (s_0^1, s_0^2), \delta, \lambda)$$

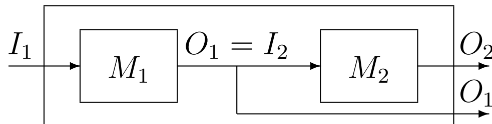
where

- $\delta\big((s_1, s_2), a\big) = \big(\delta_1(s_1, a), \delta_2(s_2, \lambda_1(s_1, a))\big)$
- $\lambda\big((s_1, s_2), a\big) = \lambda_2(s_2, \lambda_1(s_1, a))$

Synchronous serial composition of FSMs: formalization

Quizzes:

- Define another version of serial composition where the outputs of both machines are observable to the external world, as shown below:

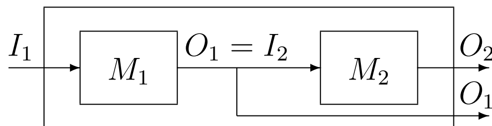


- Adapt the previous definitions to Moore machines.
- Adapt the previous definitions to Moore $\rightarrow$ Mealy.
- Adapt the previous definitions to Mealy $\rightarrow$ Moore.

Synchronous serial composition of FSMs: formalization

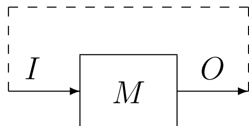
Quizzes:

- Define another version of serial composition where the outputs of both machines are observable to the external world, as shown below:



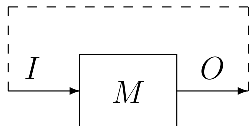
- Adapt the previous definitions to Moore machines.
- Adapt the previous definitions to Moore $\rightarrow$ Mealy.
- Adapt the previous definitions to Mealy $\rightarrow$ Moore.
- Is the serial composition of two Mealy machines a Mealy machine?
- Is the serial composition of two Moore machines a Moore machine?
- Is the serial composition Moore $\rightarrow$ Mealy a Moore or a Mealy machine?
- Is the serial composition Mealy $\rightarrow$ Moore a Moore or a Mealy machine?

Synchronous feedback composition of FSMs



Does this make sense? For Moore machines? For Mealy machines?

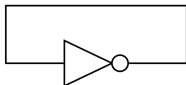
Synchronous feedback composition of FSMs



Does this make sense? For Moore machines? For Mealy machines?

Homework: Formalize feedback composition for Moore machines.

Synchronous feedback for Mealy machines: not always well defined

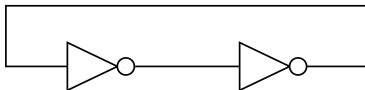


Models the equation:

$$x = \neg x$$

No solutions.

Synchronous feedback for Mealy machines: not always well defined

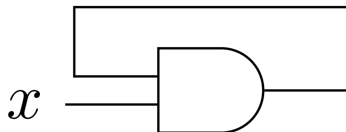


Models the equation:

$$x = x$$

Two solutions.
Which one to pick?
Ambiguous semantics.

Synchronous feedback for Mealy machines: sometimes well defined



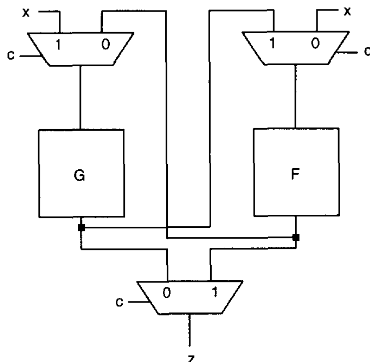
Models the equation:

$$y = y \wedge x$$

If $x = 0$, then $y = 0$ (unique solution).
If $x = 1$, then the equation becomes $y = y$ (multiple solutions).

Who cares?

Motivation: cyclic combinational circuits [Malik, 1994]:[1]

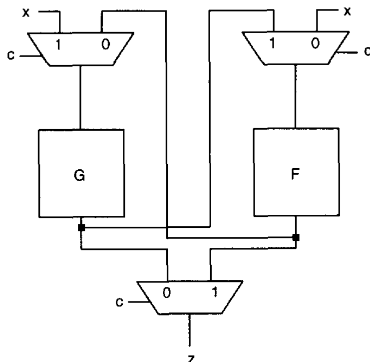


Is there an equivalent acyclic circuit?

[1] This is the same Sharad Malik of later SAT fame

Who cares?

Motivation: cyclic combinational circuits [Malik, 1994]:[1]

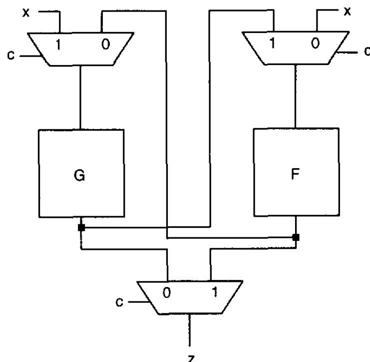


Is there an equivalent acyclic circuit?
Is the cyclic circuit better?

[1] This is the same Sharad Malik of later SAT fame

Who cares?

Motivation: cyclic combinational circuits [Malik, 1994]:[1]

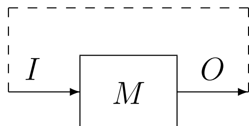


Is there an equivalent acyclic circuit?

Is the cyclic circuit better? Yes: it's smaller (assuming F and G are large).

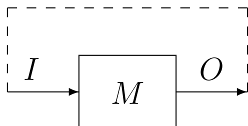
[1]This is the same Sharad Malik of later SAT fame

Synchronous feedback: various approaches



1. Forbid feedback unless "broken" by Moore components (e.g., flip-flops or unit-delays): Simulink, Lustre, ...
2. Define the semantics of feedback using fixpoint theory: Esterel, Ptolemy, ... [Malik, 1994, Shiple et al., 1996, Edwards and Lee, 2003]
3. Nondeterministic approach: up to the user to make sure model makes sense: standard approach in verification languages, e.g., nuXmv.

Synchronous feedback: various approaches



1. Forbid feedback unless "broken" by Moore components (e.g., flip-flops or unit-delays): Simulink, Lustre, ...
2. Define the semantics of feedback using fixpoint theory: Esterel, Ptolemy, ... [Malik, 1994, Shiple et al., 1996, Edwards and Lee, 2003]
3. Nondeterministic approach: up to the user to make sure model makes sense: standard approach in verification languages, e.g., nuXmv.

In practice people follow 1. We will do the same in this course.
But note that nuXmv does not warn us in case of errors, so we have to be careful ourselves!

Why call this the nondeterministic approach?

```
MODULE identity(input)
VAR
  output : boolean;
TRANS
  output = input

MODULE inverter(input)
VAR
  output : boolean;
TRANS
  output = !input


MODULE main
VAR
gate1 : identity(gate2.output);
-- gate1 : inverter(gate2.output);
gate2 : inverter(gate1.output);

SPEC AG ( gate1.output )
```

Why call this the nondeterministic approach?

```
MODULE identity(input)
VAR
  output : boolean;
TRANS
  output = input

MODULE inverter(input)
VAR
  output : boolean;
TRANS
  output = !input


MODULE main
VAR
gate1 : identity(gate2.output);
-- gate1 : inverter(gate2.output);
gate2 : inverter(gate1.output);

SPEC AG ( gate1.output )
```

This says:

$$\texttt{id.out} = \texttt{id.in}$$

$$\texttt{inv.out} = \neg\texttt{inv.in}$$

$$\texttt{id.in} = \texttt{inv.out} \wedge \texttt{inv.in} = \texttt{id.out}$$

Why call this the nondeterministic approach?

```
MODULE identity(input)
VAR
  output : boolean;
TRANS
  output = input

MODULE inverter(input)
VAR
  output : boolean;
TRANS
  output = !input


MODULE main
VAR
gate1 : identity(gate2.output);
-- gate1 : inverter(gate2.output);
gate2 : inverter(gate1.output);

SPEC AG ( gate1.output )
```

This says:

$$\texttt{id.out} = \texttt{id.in}$$

$$\texttt{inv.out} = \neg\texttt{inv.in}$$

$$\texttt{id.in} = \texttt{inv.out} \land \texttt{inv.in} = \texttt{id.out}$$

Put it all together and simplify:

$$x = y \land y = \neg x$$

Why call this the nondeterministic approach?

```
MODULE identity(input)
VAR
  output : boolean;
TRANS
  output = input

MODULE inverter(input)
VAR
  output : boolean;
TRANS
  output = !input


MODULE main
VAR
gate1 : identity(gate2.output);
-- gate1 : inverter(gate2.output);
gate2 : inverter(gate1.output);

SPEC AG ( gate1.output )
```

This says:

$$\texttt{id.out} = \texttt{id.in}$$

$$\texttt{inv.out} = \neg\texttt{inv.in}$$

$$\texttt{id.in} = \texttt{inv.out} \wedge \texttt{inv.in} = \texttt{id.out}$$

Put it all together and simplify:

$$x = y \wedge y = \neg x$$

nuXmv issues a warning about “fair states set” being empty.

Why call this the nondeterministic approach?

```
MODULE identity(input)
VAR
  output : boolean;
TRANS
  output = input

MODULE inverter(input)
VAR
  output : boolean;
TRANS
  output = !input


MODULE main
VAR
-- gate1 : identity(gate2.output);
gate1 : inverter(gate2.output);
gate2 : inverter(gate1.output);

SPEC AG ( gate1.output )
```

If we use two inverters instead:

$$x = \neg y \wedge y = \neg x$$

nuXmv says the spec is false and gives a counter-example.

Justification for the nondeterministic approach

The fact that different communities take different approaches to the synchronous feedback composition problem is not an accident:

- Circuits, synchronous languages, control communities:
  - Focus is building circuits, controllers: these are deterministic systems.
  - ⇒ Determinism is extremely important.
  - ⇒ Need compiler to catch errors that may result in nondeterministic behavior.

- Verification community:
  - Focus is checking that property holds over **all possible** system behaviors.
  - ⇒ Systems are typically nondeterministic: they have many possible behaviors (e.g., due to unknown inputs, environment behavior, over-approximations, ...).

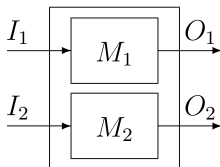
Justification for the nondeterministic approach

The fact that different communities take different approaches to the synchronous feedback composition problem is not an accident:

- Circuits, synchronous languages, control communities:
  - Focus is building circuits, controllers: these are deterministic systems.
  - $\Rightarrow$ Determinism is extremely important.
  - $\Rightarrow$ Need compiler to catch errors that may result in nondeterministic behavior.
- Verification community:
  - Focus is checking that property holds over **all possible** system behaviors.
  - $\Rightarrow$ Systems are typically nondeterministic: they have many possible behaviors (e.g., due to unknown inputs, environment behavior, over-approximations, ...).

Having said that: must be careful of unintended effects during composition.

Synchronous parallel composition of FSMs



Given two Mealy machines M_1 and M_2 with $M_i = (I_i, O_i, S_i, s_0^i, \delta_i, \lambda_i)$ the synchronous parallel composition of M_1 and M_2 is a new Mealy machine

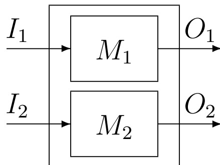
$$M = (I_1 \times I_2, O_1 \times O_2, S_1 \times S_2, (s_0^1, s_0^2), \delta, \lambda)$$

where

- $\delta\big((s_1, s_2), (a_1, a_2)\big) = \big(\delta_1(s_1, a_1), \delta_2(s_2, a_2)\big)$
- $\lambda\big((s_1, s_2), (a_1, a_2)\big) = \big(\lambda_1(s_1, a_1), \lambda_2(s_2, a_2)\big)$

We will call this the **monolithic** definition. We will see why.

Synchronous parallel composition of FSMs



Given two Mealy machines M_1 and M_2 with $M_i = (I_i, O_i, S_i, s_0^i, \delta_i, \lambda_i)$ the synchronous parallel composition of M_1 and M_2 is a new Mealy machine

$$M = (I_1 \times I_2, O_1 \times O_2, S_1 \times S_2, (s_0^1, s_0^2), \delta, \lambda)$$

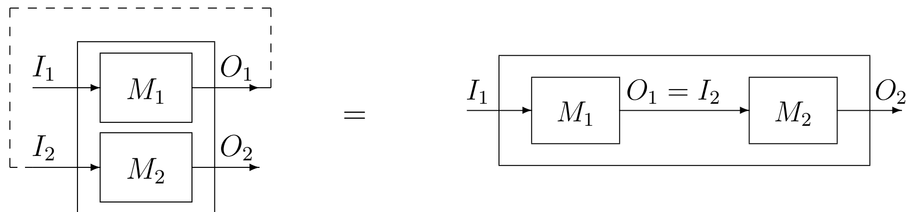
where

- $\delta\big((s_1, s_2), (a_1, a_2)\big) = \big(\delta_1(s_1, a_1), \delta_2(s_2, a_2)\big)$
- $\lambda\big((s_1, s_2), (a_1, a_2)\big) = \big(\lambda_1(s_1, a_1), \lambda_2(s_2, a_2)\big)$

We will call this the **monolithic** definition. We will see why.
Quizzes: Similar to those for serial composition.

Problem: the monolithic definition is not compositional!

These two block diagrams should be equivalent:

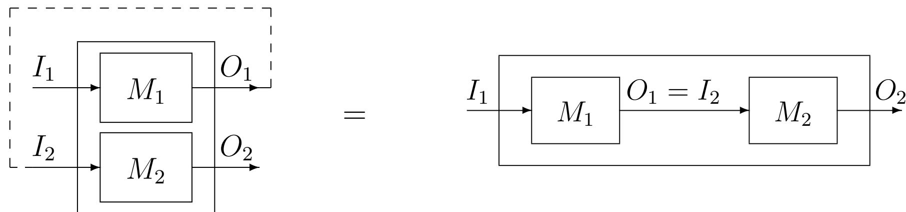


But if we use the monolithic definition, we cannot form the diagram to the left.

Solution: non-monolithic Mealy machines
[Lublinerman and Tripakis, 2008, Lublinerman et al., 2009].

Problem: the monolithic definition is not compositional!

These two block diagrams should be equivalent:



But if we use the monolithic definition, we cannot form the diagram to the left.
Solution: non-monolithic Mealy machines
[Lublinerman and Tripakis, 2008, Lublinerman et al., 2009].

Note: problem does not arise if M_1 is a Moore machine. Why?

Bibliography

Edwards, S. and Lee, E. (July 2003).
The semantics and execution of a synchronous block-diagram language.
Science of Computer Programming, 48:21–42(22).

Hoare, C. (1985).
Communicating Sequential Processes.
Prentice Hall.

Lublinerman, R., Szegedy, C., and Tripakis, S. (2009).
Modular Code Generation from Synchronous Block Diagrams – Modularity vs. Code Size.
In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'09)*, pages 78–89. ACM.

Lublinerman, R. and Tripakis, S. (2008).
Modularity vs. Reusability: Code Generation from Synchronous Block Diagrams.
In *Design, Automation, and Test in Europe (DATE'08)*, pages 1504–1509. ACM.

Malik, S. (1994).
Analysis of cyclic combinational circuits.
IEEE Trans. Computer-Aided Design, 13(7):950–956.

Milner, R. (1980).
A Calculus of Communicating Systems, volume 92 of *LNCS*.
Springer-Verlag.

Shiple, T., Berry, G., and Touati, H. (1996).
Constructive analysis of cyclic circuits.
In *European Design and Test Conference (EDTC'96)*. IEEE Computer Society.