Block Diagrams

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

Basic primitives:

- Serial composition.
- Parallel composition.
- Feedback composition.
The term **compositionality** has many meanings.

One useful meaning:

* A language is compositional if
  1. It has a notion of *component*.
  2. It has mechanisms to *compose* components.
  3. The composition of two or more components is a *component*.

Is the serial composition of FSMs an FSM? What about parallel composition? What about feedback?
Compositionality

The term *compositionality* has many meanings.

One useful meaning:

A language is compositional if

1. *It has a notion of component.*
2. *It has mechanisms to compose components.*
3. *The composition of two or more components is a component.*

Is the serial composition of FSMs an FSM? What about parallel composition? What about feedback?
Given two Mealy machines $M_1$ and $M_2$ with

$$M_i = (I_i, O_i, S_i, s^i_0, \delta_i, \lambda_i)$$

the serial synchronous composition of $M_1$ and $M_2$ is a new Mealy machine

$$M = (I_1, O_2, S_1 \times S_2, (s^1_0, s^2_0), \delta, \lambda)$$

where

$$\delta((s_1, s_2), a) = \quad$$
Synchronous Serial Composition of FSMs

Given two Mealy machines $M_1$ and $M_2$ with

$$M_i = (I_i, O_i, S_i, s_i^0, \delta_i, \lambda_i)$$

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

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

- $\delta((s_1, s_2), a) = (\delta_1(s_1, a), \delta_2(s_2, \lambda_1(s_1, a)))$
- $\lambda((s_1, s_2), a) = \lambda_2(s_2, \lambda_1(s_1, a))$
Synchronous Parallel Composition of FSMs

Quiz: Is the synchronous parallel composition of two Moore machines a Moore machine?
Synchronous Parallel Composition of FSMs

Quiz: Is the synchronous parallel composition of two Moore machines a Moore machine?

Is the synchronous parallel composition of two Mealy machines a Mealy machine?
Parallel Composition of Mealy Machines –
the Monolithic Approach

Given two Mealy machines $M_1$ and $M_2$ with

$$M_i = (I_i, O_i, S_i, s^i_0, \delta_i, \lambda_i)$$

the parallel composition of $M_1$ and $M_2$ is a new Mealy machine

$$M = (I_1 \times I_2, O_1 \times O_2, S_1 \times S_2, (s^1_0, s^2_0), \delta, \lambda)$$

where

$$\delta((s_1, s_2), a) = \begin{array}{ll}
\end{array}$$
Parallel Composition of Mealy Machines – the Monolithic Approach

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 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

- $\delta((s_1, s_2), a) = (\delta_1(s_1, a), \delta_2(s_2, a))$
- $\lambda((s_1, s_2), a) = $
Parallel Composition of Mealy Machines – the Monolithic Approach

Given two Mealy machines $M_1$ and $M_2$ with

$$M_i = (I_i, O_i, S_i, s^i_0, \delta_i, \lambda_i)$$

the parallel composition of $M_1$ and $M_2$ is a new Mealy machine

$$M = (I_1 \times I_2, O_1 \times O_2, S_1 \times S_2, (s^1_0, s^2_0), \delta, \lambda)$$

where

- $\delta((s_1, s_2), a) = (\delta_1(s_1, a), \delta_2(s_2, a))$
- $\lambda((s_1, s_2), a) = (\lambda_1(s_1, a), \lambda_2(s_2, a))$
Problem with the Monolithic Approach

The monolithic approach is **not** compositional.
Problem with the Monolithic Approach

The monolithic approach is **not** compositional.

Note: replacing the *CodeReuseSubsystem* with a normal *Subsystem* allows Simulink to run the model without problems. Why?
Problem with the Monolithic Approach

The monolithic approach is **not** compositional.

---

- Note: replacing the `CodeReuseSubsystem` with a normal `Subsystem` allows Simulink to run the model without problems. Why?
  - Simulink *flattens Subsystems* but not `CodeReuseSubsystems`. 
Problem with the Monolithic Approach

The monolithic approach is **not** compositional.

The two compositions are not equivalent!
The Non-Monolithic Approach
[Lublinerman and Tripakis, 2008, Lublinerman et al., 2009]

Solution: non-monolithic code

A Mealy machine with multiple output functions

```
P.fire1( x1 ) returns y1 {
    return A.fire( x1 );
}

P.fire2( x2 ) returns y2 {
    return B.fire( x2 );
}
```
Non-monolithic interface

P.fire1( x1 ) returns y1 ;

P.fire2( x2 ) returns y2 ;

P.fire1( x1 ) returns y1 ;

P.fire2( x2 ) returns y2 ;
Non-monolithic interface

```
P.fire1( x1 ) returns y1 ;
```

```
P.fire2( x2 ) returns y2 ;
```

```
P.fire1( x1 ) returns y1 ;
```

```
P.fire2( x2 ) returns y2 ;
```
Non-monolithic interface

interface does not restrict usage

P.fire1( x1 ) returns y1 ;

P

y1

x1

x2

y2

P.fire2( x2 ) returns y2 ;
Bottom-up interface synthesis

Given interfaces for sub-blocks A, B, C, compute interface for composite block P.
Simple non-monolithic interface

\[ \text{P.fire1( x1 ) returns } y1; \]
\[ \text{P.fire2( x2 ) returns } y2; \]
What about more complex diagrams?

what about this?

or this?
Interface synthesis for block diagrams
= graph clustering

block diagram

interface
How it’s done

A → B
C → D
P
How it’s done

Interface for A

Interface for B

Interface for C

Interface for D
How it’s done

clustering
How it’s done

Interface for P

P

P.fire1()

P.fire2()

P.fire3()
Different clusterings => different interfaces

A non-monolithic interface for P

A monolithic interface for P

trade-off: interface size vs. reusability
Different clustering algorithms = different tradeoffs

<table>
<thead>
<tr>
<th>Clustering method</th>
<th>Complexity</th>
<th>Achieves maximal reusability?</th>
<th>Achieves minimal interf. size?</th>
<th>Modularity bound?</th>
<th>Achieves minimal code size?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“step-get”</td>
<td>Polynomial</td>
<td>No</td>
<td>Almost</td>
<td>&lt;=2 functions</td>
<td>Yes</td>
</tr>
<tr>
<td>“dynamic”</td>
<td>Polynomial</td>
<td>Yes</td>
<td>Yes</td>
<td>&lt;=N+1 functions*</td>
<td>No</td>
</tr>
<tr>
<td>“disjoint”</td>
<td>NP-complete</td>
<td>Yes</td>
<td>Yes</td>
<td>?</td>
<td>Yes</td>
</tr>
<tr>
<td>“greedy”</td>
<td>Polynomial</td>
<td>Yes</td>
<td>No</td>
<td>?</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* N = number of block outputs
In 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL’09), pages 78–89. ACM.

Modularity vs. Reusability: Code Generation from Synchronous Block Diagrams.
In Design, Automation, and Test in Europe (DATE’08), pages 1504–1509. ACM.