DEVS Formalism

- Basics of Set theory
- Atomic model DEVS Formalism
  - Examples
- Coupled DEVS Specification
- Closure Under Coupling, Hierarchical Models
  - Example: the GenDevsTest package
A Discrete Event System Specification (DEVS) is a structure

\[ M = (X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{con}}, \lambda, ta) \]

where

- \( X \) is the set of input values
- \( S \) is a set of states,
- \( Y \) is the set of output values
- \( \delta_{\text{int}}: S \to S \) is the internal transition function
- \( \delta_{\text{ext}}: Q \times X^b \to S \)

is the external transition function, where

\[ Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\} \]

is the total state set

- \( e \) is the time elapsed since last transition
- \( X^b \) denotes the collection of bags over \( X \) (sets in which some elements may occur more than once).

- \( \delta_{\text{con}}: Q \times X^b \to S \)

is the confluent transition function,

- \( \lambda: S \to Y^b \) is the output function
- \( ta: S \to \mathbb{R}^+_{0,\infty} \) is the time advance function
Basics of Set Theory

• A set is a collection of things called elements and is denoted by \{ ..\} with no element repeated.
• The order in which elements are written does not count, e.g. \{a,b,c\} is the same set as \{c,b,a\}.
• The *null* or *empty* set contains no elements and is written \{ \}.
• The elements of a set can be said to be contained in it, be members of it, or belong to it. We denote set membership by \( \in \) , For example, \( C \in \) English alphabet.
Transformation

transformation: set of operands $\rightarrow$ set of transforms
(This is read as, “transformation maps or transforms set of operands into set of transforms”)

Ex: Operator: exposure to sun.
Transitions:
cold soil $\rightarrow$ warm soil
unexposed photographic plate $\rightarrow$ exposed plate
colored pigment $\rightarrow$ bleached pigment
Set of operands = \{cold soil, unexposed photographic plate, colored pigment\}
Set of transforms = \{ warm soil, exposed plate, bleached pigment\}

Ex: code
code A B ... Y Z
    B C ... Z A
Set of operands = English alphabet
Set of transforms = English alphabet
code: English alphabet $\rightarrow$ English alphabet
given by: A is coded to B, …Z is coded to A
Relation

A relation is a set whose elements are pairs, i.e., it is of the form \{ (a,b),
...(x,y) \}
A transformation as defined by Ashby is a relation.

Ex: exposure to sun causes the transformation or relation
\[ \{ \text{(cold soil, warm soil),}
\text{(unexposed photographic plate, exposed plate),}
\text{(colored pigment, bleached pigment)} \} \]
Relation

- One set is *included in* another if each element in the first is also found in the second.
- We write \( A \subseteq B \) to denote that \( A \) is included in \( B \). Said another way, for each element \( x \in A \), we have \( x \in B \). Or yet another way, \( A \) is wholly contained in \( B \), or is a *subset* of \( B \).
- The *domain* of a relation is the set of all its left hand elements, i.e., all the elements on the left hand sides of its pairs. The domain of \( R \) is denoted by \( \text{Domain}(R) \).
- The *range* of a relation is the set of all its right hand elements, i.e., all the elements on the right hand sides of its pairs. The range of \( R \) is denoted by \( \text{Range}(R) \).
Closure

Closure. When an operator acts on a set of operands it may happen that the set of transforms obtained contains no element that is not already present in the set of operands, i.e., the transformation creates no new element.

A relation is \textit{closed} if its range is included in its domain, i.e., \( \text{Range}(R) \subseteq \text{Domain}(R) \).

Ex: code is closed since its domain and range are the same English alphabet.
Ex: exposure to sun is not closed since its range contains warm soil which is not in its domain.
The *crossproduct* of sets A and B, written AxB is the set of all pairs (a,b) where a ∈ A and b ∈ B.

Notice that a crossproduct is a special relation – the relation in which all pairs are present. This is the least restrictive relation. Any other relation places some constraints on the simultaneous values that pairs can take on. For example, an identity relation constrains the pairs to the form (a,a).

We generalize the crossproduct to a number of sets by taking the crossproduct of the third with that of the first two, the crossproduct of the fourth with the result of the first three sets, etc.
Vector

A vector, is defined as a compound entity, having a definite number of components. It is conveniently written thus: \(( a_1, a_2, \ldots, a_n )\), which means that the first component has the particular value \(a_1\), the second the value \(a_2\), and so on.

Ex: weather vector: (height of barometer, temperature, cloudiness, humidity)

A vector of size \(n\) is an element of a crossproduct of \(n\) sets.

Ex: weather vector: (height of barometer, temperature, cloudiness, humidity) (for instance i(998 mbars, 56.2° F, 8, 72%) ) is an element of the crossproduct \(B \times T \times C \times H\) where

- \(B\) might be the set of integers from 0 to 1000
- \(T\) might be the set of numbers between –100 and 150
- \(C\) might be the set of integers between 0 and 10 (for levels of cloudiness)
- \(H\) might be the set of integers between 0 and 100.

We refer to components of a vector as variables and the corresponding sets of values they can take on as their range sets.

Ex: in the weather vector, the first variable is barometric reading and its range is \(B\), the second variable is temperature and its range is \(T\).
Other Concepts

A transformation is single-valued if it converts each operand to only one transform. Otherwise it is multi-valued.
Ex: the transformation
A B C D
B A A D
is single valued.

but the transformation
A       B       C       D
B or D  A B or C D D
is not single- valued.

A relation is *single-valued* if no two pairs have the same left hand element.
Ex: the relation
\[
\{(A, B), (B, A), (C, A), (D, D)\}
\]
is single valued because each element in the domain has only one range element it is associated with.

But the relation
\[
\{(A, B), (A, D), (B, A), (B, B), (B, D), (C, D), (D, D)\}
\]
is not single valued, because A is mapped to both B and D, for example.

A relation that is single-valued is called a *function* or a *mapping*. In other words, a function is a single-valued relation.

Of the single-valued transformations, a type of some importance is *one-one*. In this case the transforms are all different from one another.
# Collections, Maps, Relations

## The GenCol package in DEVSJAVA

<table>
<thead>
<tr>
<th>Collection</th>
<th>Defining Property</th>
<th>Useful For</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>-Indefinite size</td>
<td>-Variable sized collections</td>
</tr>
<tr>
<td>List</td>
<td>-Indexed elements</td>
<td>-sequencing</td>
</tr>
<tr>
<td></td>
<td>-insert/remove anywhere</td>
<td>- basis for queues/stacks</td>
</tr>
<tr>
<td>Set</td>
<td>-admission based on equality (no duplicates)</td>
<td>- unique tracking of object occurrences</td>
</tr>
</tbody>
</table>
| Bag        | • no admission criteria  
|            | • multiplicities are counted | - word frequency counts |
| Map        | - one-one correspondence (keys to values) | - dictionary (one meaning per word) |
| Relation   | - many-many correspondence | - dictionary (multiple meanings per word) |

Dr. Xiaolin Hu
A Discrete Event System Specification (DEVS) is a structure
\[ M = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{con}}, \lambda, t_a \rangle \]
where
- \( X \) is the set of input values
- \( S \) is a set of states,
- \( Y \) is the set of output values
- \( \delta_{\text{int}}: S \rightarrow S \) is the internal transition function
- \( \delta_{\text{ext}}: Q \times X^b \rightarrow S \)
  is the external transition function, where
  \( Q = \{(s, e) \mid s \in S, 0 \leq e \leq t_a(s)\} \) is the total state set
- \( e \) is the time elapsed since last transition
- \( X^b \) denotes the collection of bags over \( X \)
  (sets in which some elements may occur more than once).
- \( \delta_{\text{con}}: Q \times X^b \rightarrow S \)
  is the confluent transition function,
- \( \lambda: S \rightarrow Y^b \) is the output function
- \( t_a: S \rightarrow \mathbb{R}_{0,\infty}^+ \) is the time advance function
A diver drops immediately to 60 ft depth, stays there for 35 min., then rises to surface and stay for one hour. He then drops to 40 ft, depth staying 25 min., and finally spends 5 minutes at 5ft before rising to the surface. If there is emergency call, the diver needs to come up to 5 ft and stay there for at least 5 minutes before rising to the surface.
Basic DEVS: Example Scuba Model

\[
DEVS_{\text{scuba}} = (X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \lambda, \tau a)
\]

where

\[X = \{\text{emergency call}\}\]
\[Y = \{1\}\]
\[S = \{“sixty”,”forty”,”five”,”surface1”,”surface2”\} \times \mathbb{R}_0^+\]
\[\delta_{\text{int}} (“sixty”, \sigma) = (“surface1”, 60)\]
\[\delta_{\text{int}} (“surface1”, \sigma) = (“forty”, 25)\]
\[\delta_{\text{int}} (“forty”, \sigma) = (“five”, 5)\]
\[\delta_{\text{int}} (“five”, \sigma) = (“surface2”, \infty)\]

\[\lambda (\text{phase}, \sigma) = 1\]
\[\tau a (\text{phase}, \sigma) = \sigma\]

\[\delta_{\text{ext}}(\text{phase}, \sigma, e, x) = (“five”, 5) \text{ if phase} \neq \text{“surface1”,”surface2”,”5”}\]
\[\quad = (\text{phase}, \sigma-e) \text{ otherwise}\]
\[\delta_{\text{con}}(\text{phase}, \sigma, e, x) = \delta_{\text{ext}}(\text{phase}, \sigma, e, x) \text{ //pay attention to external event (call)}\]
Fire-once Neuron

This neuron fires exactly once after a fixed delay while ignoring all subsequent inputs.

Firing delay >0
Basic DEVS: Example FireOnce Neuron and Modifications

\[ \text{DEVS}_{\text{fire,once}} = (X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \tau_a) \]

where

\[ X = \mathbb{R} \]
\[ Y = \mathbb{R} \]

\[ S = \{ \text{“receptive”, “fire”, “refract”} \} \times \mathbb{R} \times \{ 0, \infty \} \times \mathbb{R} \]

\[ \delta_{\text{ext}}(\text{“receptive”, } \sigma, \text{size}, e, x) = (\text{“fire”, } \sigma - e, \text{size}) \]
\[ \delta_{\text{ext}}(\text{“fire”, } \sigma, \text{size}, e, x) = (\text{“fire”, } \sigma - e, \text{size}) \]
\[ \delta_{\text{ext}}(\text{“refract”, } \sigma, \text{size}, e, x) = (\text{“refract”, } \sigma - e, \text{size}) \]

\[ \delta_{\text{int}}(\text{“fire”, } \sigma, \text{size}) = (\text{“refract”, } \infty, \text{size}) \]

\[ \lambda(\text{“fire”, } \sigma, \text{size}) = \text{size} \]

\[ \tau_a(\text{phase, } \sigma, \text{size}) = \sigma \]

Initial state = \{“receptive”, \infty, 1\}

Use `Continue(e)` to retain next event time of original pulse.
### Basic DEVS: Example FireOnce Neuron and Modifications (Continued)

a) the second input pulse cancels the scheduled output pulse and sends the model to refract.
Basic DEVS: Example FireOnce Neuron and Modifications (Continued)

a) the second input pulse cancels the scheduled output pulse and sends the model to refract

\[ \delta_{\text{ext}}(\text{fire}, \sigma, \text{size}, e, x) = (\text{refract}, \infty, \text{size}) \]

b) the second input pulse cancels the scheduled output pulse and re-schedules this output after a time given by `fireDelay`.

\[ \delta_{\text{ext}}(\text{fire}, \sigma, \text{size}, e, x) = (\text{fire}, \text{fireDelay}, \text{size}) \]

c) the second input pulse causes the scheduled output pulse to occur earlier than it would have – the new time left to fire is equal to half what it would have been had no pulse arrived.

\[ \delta_{\text{ext}}(\text{fire}, \sigma, \text{size}, e, x) = (\text{fire}, (\sigma - e)/2, \text{size}) \]

d) same as b) and in addition, the size of the output pulse is increased by an amount that equals the ratio of the interval between the arrivals of the two pulses to the time that was left to fire. This gives the output pulse “credit” for the cancelled pulse in proportion to how late in the firing phase it was aborted (near zero credit for nearly co-incident pulses; almost full credit (= 1) for a input pulse that comes almost at the end of the firing phase.)

\[ \delta_{\text{ext}}(\text{fire}, \sigma, \text{size}, e, x) = (\text{fire}, \text{fireDelay}, \text{size} + e/\sigma) \]
Avoiding Some Common Formalism Violations

\( X^b \) is a bag of inputs whose elements are in \( X \),

\[ \delta_{ext}: S \times X^b \rightarrow S \]

This means that the state after receiving a bag of inputs is uniquely determined by the current state, the elapsed time, and in particular the bag of inputs. Since a bag is an unordered collection, the result cannot depend on the order used in examining the inputs. In the example on the left, the order of examining the bag \{“a”,”b”\} matters since if “a” is in the first content to be examined the result is to do A, while if “b” is the first content, the result is to do B. The example on the right completely examines the bag for occurrence of “a” and then occurrence of “b”. The result is always to do A first and B next for the bag \{“a”,”b”\}.

```java
public void deltext(double e, message x) {
    Continue(e);
    for (int i = 0; i < x.getLength(); i++) {
        if (messageOnPort(x, "a", i)) <do A>
        else
            if (messageOnPort(x, "b", i)) <do B>
    }
}
```

result is defined

```
public void deltext(double e, message x) {
    Continue(e);
    for (int i = 0; i < x.getLength(); i++){
        if (messageOnPort(x, "a", i))
            <do A>
    }
    for (int i = 0; i < x.getLength(); i++)
        if (messageOnPort(x, "b", i))
            <do B>
}
```

result is uniquely defined
Avoiding Some Common Formalism Violations

\[ \lambda : S \rightarrow Y \]

This means that the output function does not have an effect on the state – it can’t change it, it can only look at it. The following violates this requirement.

```java
public message out() {
    message m = new message();
    if (phaseIs("transmit")){
        m.add(makeContent("out", new entity("packet " + count++;"+destination)));
        count = count + 1;
    }
}
```

Here the intent is that the state variable, count is incremented by the call to the output function. But DEVS simulator is guaranteed only to use the return result of the call as specified in the DEVS simulation protocol, not to obey the side-effect of changing the count. The correct way to update the count is in the internal transition function which is called immediately after the output function:

```java
public void int() {
    if (phaseIs("transmit"))
        count = count + 1;
}
```
Coupled Model Specification

\[ \text{DN} = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} > \]

- **X**: a set of input events.
- **Y**: a set of output events.
- **D**: an index set (names) for the components of the coupled model.

For each \( i \in D \),

- \( M_i \) is a component DEVS model.

For each \( i \in D \cup \text{self} \), \( I_i \) is the set of influencees of \( i \).

For each \( j \in D \cup \text{self} \),

- \( Z_{i,j} : Y_i \rightarrow X_j \) is the output translation mapping.
Every DEVS coupled model has a DEVS Basic equivalent.
HW1 Code, Q&A