Hybrid dynamics in Modelica: should all events be considered synchronous

Ramine Nikoukhah
INRIA

EOOLT 2007

Modelica/Scicos

- **Modelica**: language for modeling physical systems. Originally continuous-time modeling extended to discrete-time.
- **Scicos**: block diagram environment for modeling dynamical systems. For discrete and explicit continuous-time models.
- **Modelica/Scicos**: same type of hybrid systems to model. They face similar problems.

Objective: Apply Scicos solutions to Modelica

OUTLINE

- Conditioning and sub-sampling in Modelica
  - *Synchronous versus simultaneous*
  - *Primary and secondary when clauses*
  - *Restrictions on the use of when and if*
  - *Continuous-time dynamics*
  - *Initial conditions*
  - *Union of events*
- *Back-end compiler*

Conditioning and sub-sampling in Modelica

- ***when-elsewhen*** *for sub-sampling and* ***if-then-else*** *for conditioning.*
- ***Scicos*** *counterparts: event generation and if-then-else and ESelect block.*
- *Two different types of* ***when:***
  - ***Primary when:*** *event is detected by zero-crossing or similar mechanism*
  - ***Secondary when:*** *event is the consequence of a jump of a discrete variable*

Synchronous versus simultaneous

Events considered synchronous if they can be traced back to a single event

```
when sample(0,1) then
  d=pre(d)+1;
end when;

when d>3 then
  a=pre(a)+1;
end when;
```

d jumps activating event

*First **when** is primary, the other is secondary and related to the first one.*

Events considered asynchronous if they cannot be traced back to a single event

x and **y** are theoretically identical but the numerical solver may find:

1. **x>2** first and then **y>2**
2. **y>2** first and then **x>2**
3. **both** detected simultaneously

We consider case 3 as 1 or 2:

- ***z=pre(z)+3; v=u+1; u=z+1;***
- ***u=z+1; z=pre(z)+3; v=u+1;***

Dymola considers three cases:

- ***z=pre(z)+3; v=u+1; u=z+1;***
- ***u=z+1; z=pre(z)+3; v=u+1;***
- ***z=pre(z)+3; u=z+1; v=u+1;***

```
equation
x=time*time/2;
der(y)=time;
  when (x>2) then
    z=pre(z)+3;
    v=u+1;
  end when;
  when (y>2) then
    u=z+1;
  end when
```

Both solutions lead to non-deterministic simulations

Some models are inherently nondeterministic: normal to have nondeterministic simulation

Explicit synchronization removes non-determinism:

```
equation
der(x)=1;
when x>1 then
  <eq1>
end when;
when x>1 then
  <eq2>
...
end when;
```

should be written

```
equation
der(x)=1;
E=x>1;
when E then
  <eq1>
end when;
when E then
  <eq2>
...
end when;
```

automatically transformed to

```
equation
der(x)=1;
E=x>1;
when E then
  <eq1>
  <eq2>
...
end when;
```

Note: we shall see later, when ***type Event*** is introduced, that ***E=x>1*** should be replaced with ***E=event(x>1).*** **In fact *E=x>1* should not always be allowed.**

Most cases synchronized events are declared explicitly by user. Only risk is use of **sample**

May want to consider *sample(0,1)* and *sample(0,1)* as synchronous:

```
equation
when sample(0,1) then
  a=b;
end when;
when sample(0,1) then
  b=c;
end when;
```

Difficult if come from different models or have different parameters: sample(1/3,2)

```
equation
E=sample(0,1);
when E then
  a=b;
end when;
when E then
  b=c;
end when;
```

Solutions:

1. Use *Boolean* (or ***Event***) and synchronize modules with a unique **clock**
2. Treat *sample* as a special pre-compilation directive (**Simulink** solution)

In **Simulink-like solution**, *samples* would be synchronized with each other by finding a fastest common clock. But they are not synchronized with other time events.

So even by adopting Simulink-like solution and introducing type *Event*, there is no reason to assume synchronized primary *when* clauses.

Basic assumption: primary *when* clauses are based on time events (of type zero-crossing) and are asynchronized.

This assumption is in contradiction with Dymola's interpretation of Modelica specification.

Primary and secondary when clauses

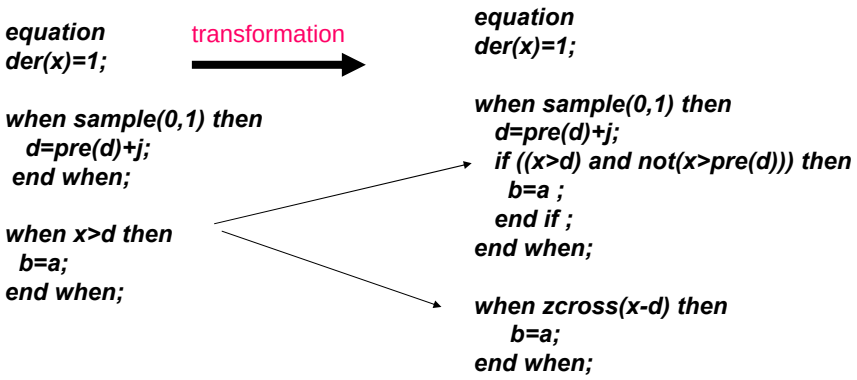
- All *when*'s cannot be classified as primary or secondary. Some are **mixed**.

```
equation
der(x)=1;

when sample(0,1) then
  d=pre(d)+j;
end when;

when x>d then
  b=a;
end when;
```

Two possibilities:
1. *x* crosses constant *d* (zero-crossing)
2. *d* jumps across the condition at sample time

Mixed when clause

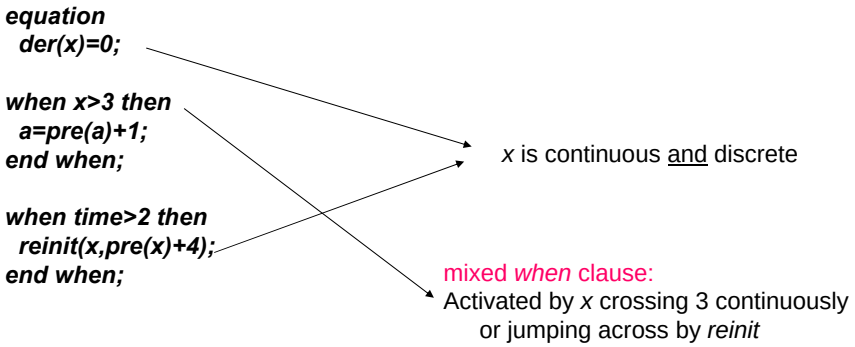
Removing mixed when clauses



Note: inside *sample(0,1) when*, *pre(x)=x* so
(x>d) and not(pre(x)>pre(d)) ≡ ***(x>d) and not(x>pre(d))*** ≠ ***edge(x>d)***

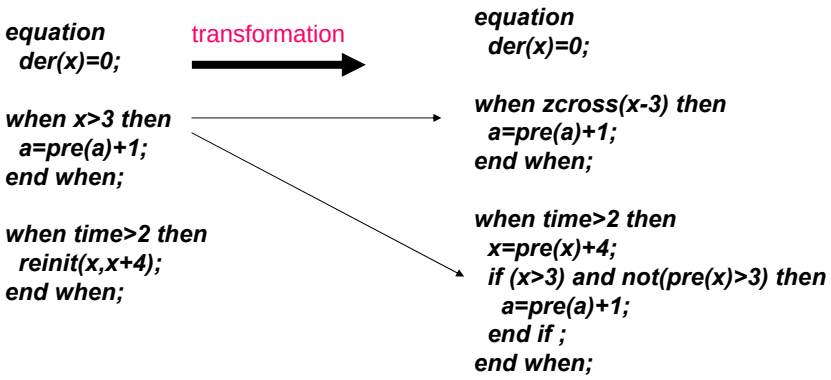
Note: resulting model does not respect single assignment rule.
But that is OK because the two *when* clauses are primary (asynchronous).
Moreover, the *if* does not have an *else* branch defining *b* (OK too, will see later).

Other type of mixed when clause



Must take into account the dual nature of *x*.

Removing the mixed types

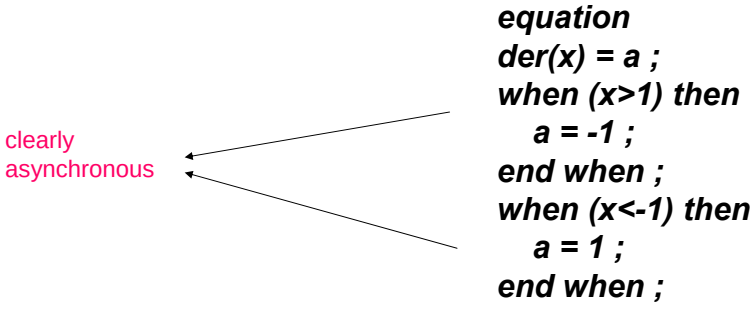


Note: synchronization aspect of *reinit* is ambiguous in the specification.
reinit(x,pre(x)+4) ≡ ***x=pre(x)+4*** is a possible interpretation.

Note: resulting model does not respect single assignment rule.
OK because the two *when* clauses are primary (asynchronous)

Restrictions on use of when and if

- **Case of when**: Single assignment rule must be removed for transformed model.



We may consider removing it also for the original model.

Accept this model with a warning

Consider following modifications:

1. Restriction be lifted for primary *when* clauses.
2. Restriction be lifted in all *when* clauses as long as the equations defining common variables are identical. For example for all conditions *c1*, *c2* (synchronous or not), accept:

```
equation
when c1 then
      b=a;
end when;

when c2 then
      b=a;
end when;
```

Note: Second modification concerns only transformed model.

- **Case of if**: remove conditions on number of equations in different branches after transformation for <u>discrete</u> variables.
  Accept:

```
equation
when sample(0,1) then
 if u>0 then
   v=1;
 end if;
end when;
```

not equivalent to

```
equation
when sample(0,1) then
 if u>0 then
   v=1;
 else
   v=pre(v) ;
 end if;
end when;
```

Note: allowing absence of else branch not just a facility but real sub-sampling.

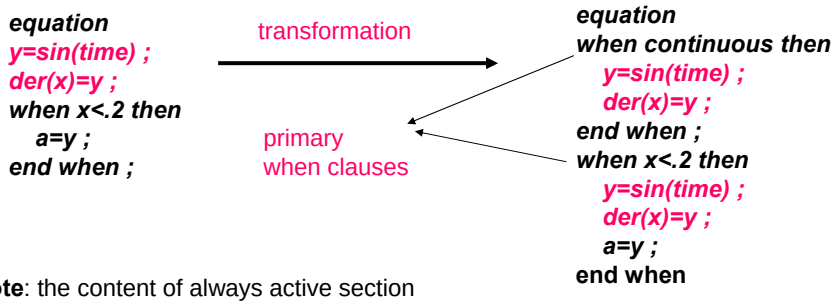
Similar restriction on ***elsewhen*** should be removed as well, at least for the transformed model.

Consider accepting as original model with a <u>warning</u>

Continuous-time dynamics

- **Equations in *equation* section but outside when clauses are always active (Scicos terminology).**
- **Scicos defines a fictitious activation clock. It activates all the time except at event times. So:**

1. **It is asynchronous with other events.**
2. **Its union with other events yields "always activation".**



Note: the content of always active section copied inside all when clauses.

Initial conditions

- **All initial conditions grouped inside a single *when* clause after the front-end compilation:**

```
when initial then
 a=0 ;
 d=3 ;
 if …..
 …
end when ;
```

when initial considered asynchronous with all other events.

when terminal can be defined similarly.

Union of events

- ***when*** and ***elsewhen*** clauses can be activated at the union of events

```
when {c1,c2,c3} then
  < eq1 >
  < eq2 >
  .....
end when;
```

c1, c2, c3 may be synchronous or not

The content of synchronous *when* clauses should not be executed more than once during synchronous activation:

```
equation
der(x)=x;
when x>1 then
  d=pre(d)+1;
end when;
when {d>2,2*d>4} then
  a=pre(a)+1 ;
end when;
```

a incremented once

a incremented twice

```
equation
der(x)=x;
when x>1 then
  d=pre(d)+1;
end when;
when x>1 then
  e=pre(e)+1;
end when;
when {d>2,e>2} then
  a=pre(a)+1 ;
end when;
```

Dymola's interpretation

- **Example*:***

```
when sample(0,3) then
  d=pre(d)+1;
end when;
when time>=3 then
  e=pre(e)+1;
end when;
when {d>1,e>0} then
  a=pre(a)+1 ;
end when;
```

Dymola increments *a* once

In **Dymola** *d>1, e>0* are synchronous (*a* is incremented only once at time 3). This interpretation must be avoided even if treating *sample* as in **Simulink**.

Union of events in Scicos: sum of activation signals

A precompiler separates sums in **Scicos**. Same can be done for **Modelica**:

```
when {c1,c2,c3} then
   < eq1 >
   < eq2 >
   …..
end when;
```

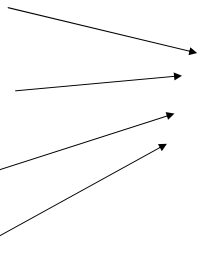
```
when c1 then
   < eq1 >
   < eq2 >
   …..
end when;
when c2 then
   < eq1 >
   < eq2 >
   …..
end when;
when c3 then
   < eq1 >
   < eq2 >
   …..
end when;
```

This transformation removes all union-of-events constructors

Back-end compiler

- The application of the transformations (on a **flat Modelica** model) leads to:

```
when initial then
  …
end when;
when continuous then
  ….
end when;
when <xxx> then
  …
end when;
when <yyy> then
  …
end when ;
……
```



primary and secondary when clauses

Model now contains a series of when clauses: primary and secondary

Compiler: phase I

- Secondary *when* clauses are removed to obtain a model containing only primary *when* clauses (similar to Scicos compiler phase 1)

Compiler: phase II

- Each primary *when* is associated with an asynchronous (independent) event → content of each *when* compiled separately
- For each content, do static scheduling and generate code (Scicos compiler phase 2)

During simulation, depending on the event, the corresponding (and only the corresponding) *when* section is executed.