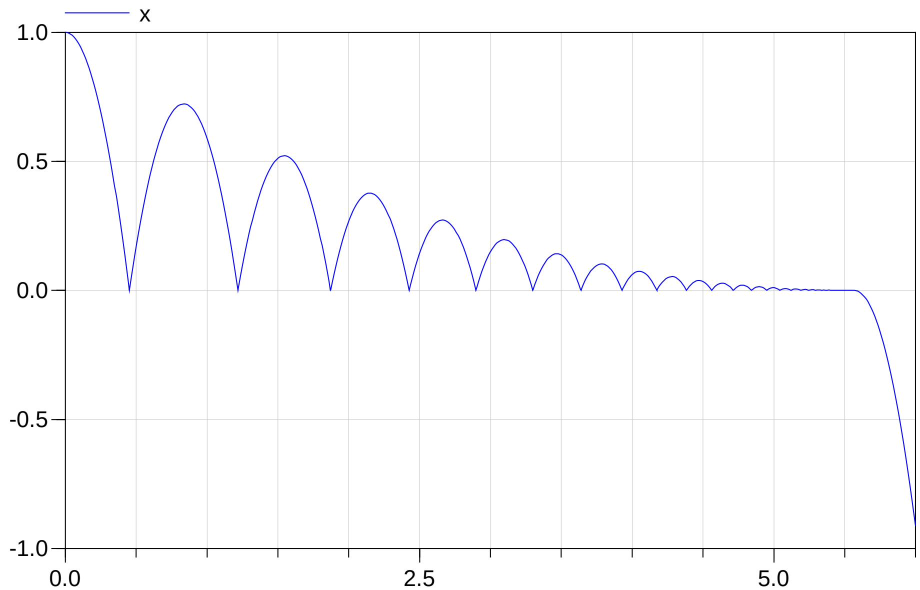
Virtual Physics
Equation-Based Modeling

TUM, January 13, 2015

Modeling and Simulation of Discontinuous Systems



Dr. Dirk Zimmer

German Aerospace Center (DLR), Robotics and Mechatronics Centre

Motivation

Today, we shall look at the problem of dealing with discontinuities in modeling and simulation.

- Models from engineering often exhibit discontinuities that describe situations such as switching, limiters, dry friction, impulses, or similar phenomena.
- The modeling environment must deal with these problems in special ways, since they influence strongly the numerical behavior of the underlying differential equation solver.

Standard ODE-Solvers

Robotics and Mechatronics Centre

What happens if we simply apply one of our ODE-solvers on a system with discontinuity?

- The discontinuity occurs in f(**x**(*t*),*t*).
- All ODE-solvers (and their error-estimations) are based on a polynomial approximation of f(**x**(*t*),t).
- Higher-Order methods (order > 1) even suppose that f(**x**,*t*) is differentiable multiple times.

Applying Standard ODE-Solvers

What happens if we simply apply on of our ODE-solvers on a system with discontinuity?

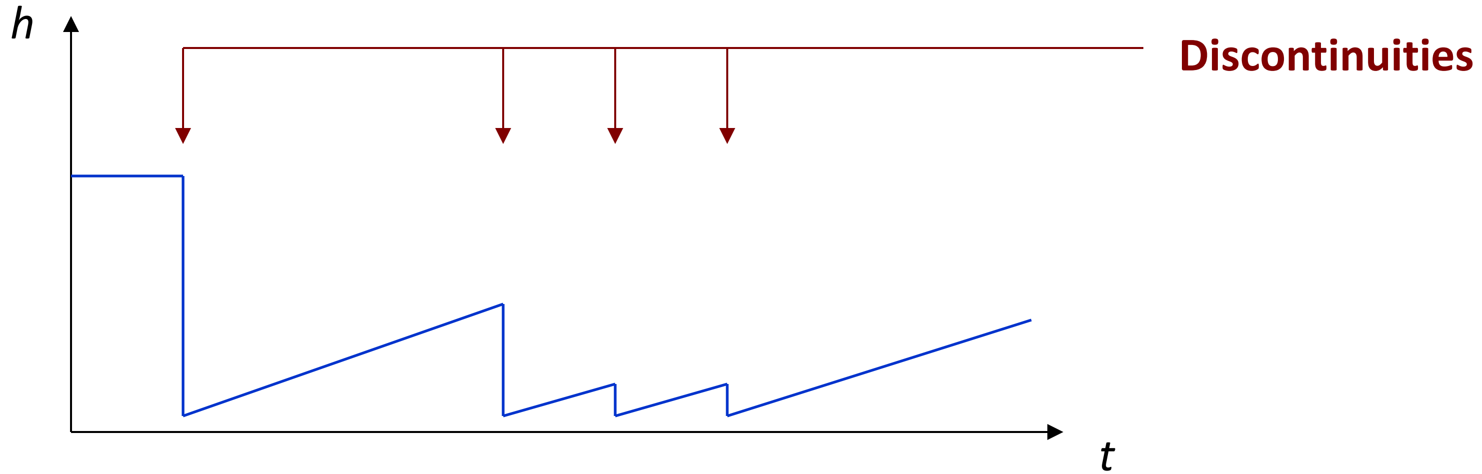
- Polynomials are always continuous and continuously differentiable functions.
- Therefore, when the state equations of the system:

  $$d\mathbf{x}/dt = f(\mathbf{x}(t), t)$$

  exhibit a discontinuity, the polynomial extrapolation is a very poor approximation of reality.
- Consequently, integration algorithms with a fixed step size exhibit a large integration error, whereas integration algorithms with a variable step size must reduce the step size dramatically in the vicinity of the discontinuity.

Applying Standard ODE-Solvers

- An integration algorithm of variable step size reduces the step size at every discontinuity.
- After passing the discontinuity, the step size is only slowly enlarged again, as the integration algorithm cannot distinguish between a discontinuity and a point of large local stiffness (with a large absolute value of the derivative).
- The step-size is constantly too small. The integration is inefficient at best if not even totally inaccurate.

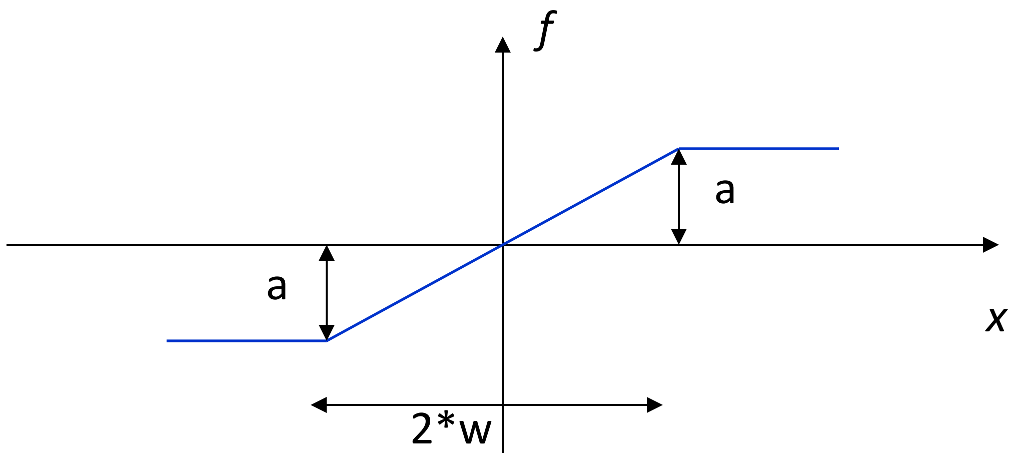


Applying Standard ODE-Solvers

Robotics and Mechatronics Centre

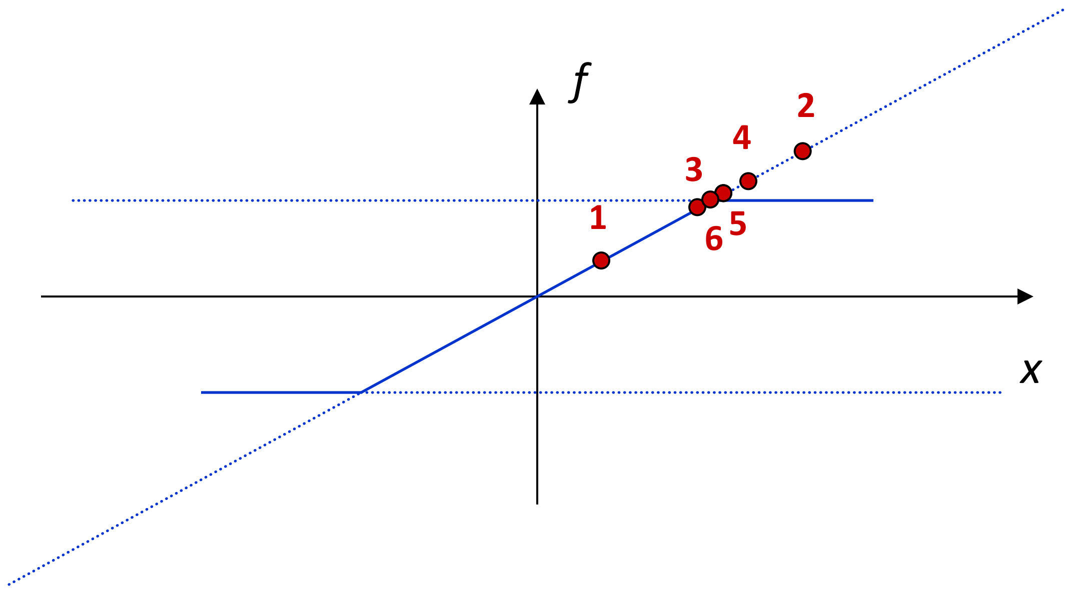
- Trying to handle discontinuities implicitly by standard ODE solvers is evidently not a good solution.
- We can avoid the occurring problems if we model the discontinuities explicitly.
- The expression is one way to do this in Modelica:

```
f = if x < -w then –a else if x<w then a*x/w else a
```



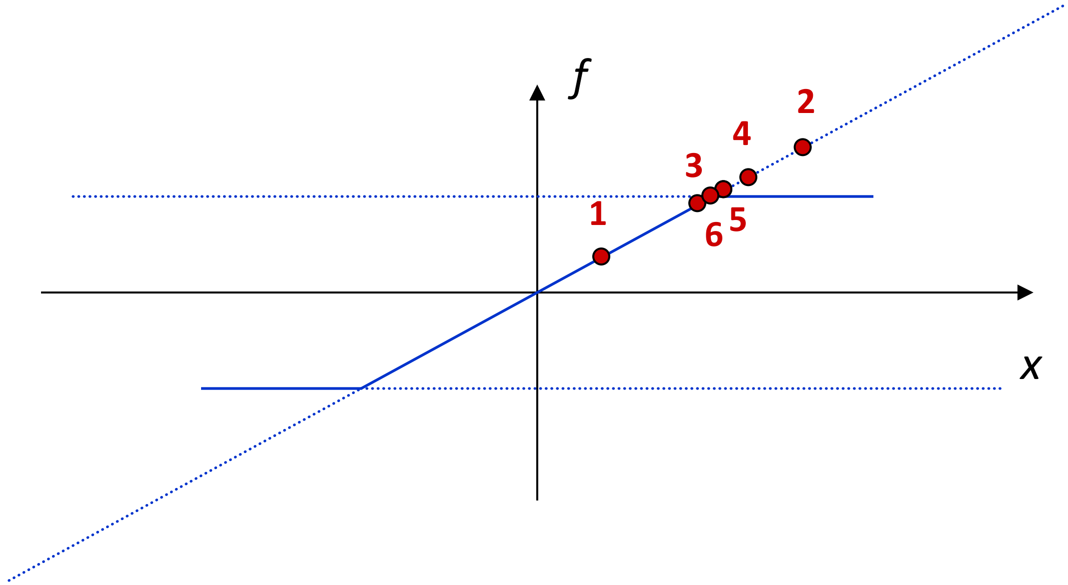
Applying Standard ODE-Solvers

- This if-statement models a state-event since the occurrence of the discontinuity is dependent on the state x.
- An integration algorithm may now precisely locate the event by iterating for the event.
- For instance, by using the bi-section algorithm:



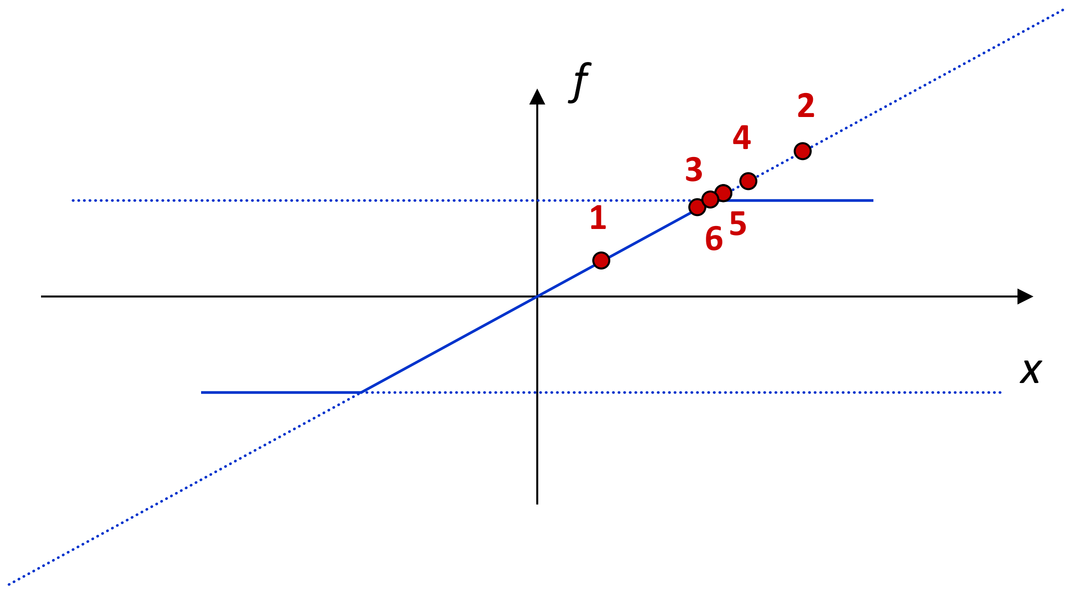
Applying Standard ODE-Solvers

- Bi-section converges slowly. Hence one may prefer the secant method or its “safer”-twin: regula-falsi.
- It is possible to combine the secant method and bi-section
  → Dekker’s methods, Brent’s method.



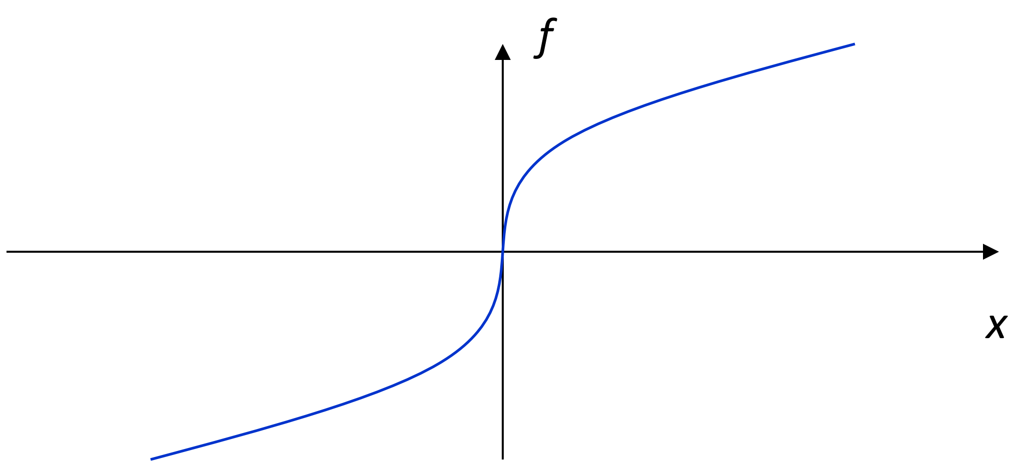
Applying Standard ODE-Solvers

- The iteration for the event-location is thereby performed on the current model equation (here `f = a*x/w`).
- The event itself changes then the model equation (here to `f = a`)



Applying Standard ODE-Solvers

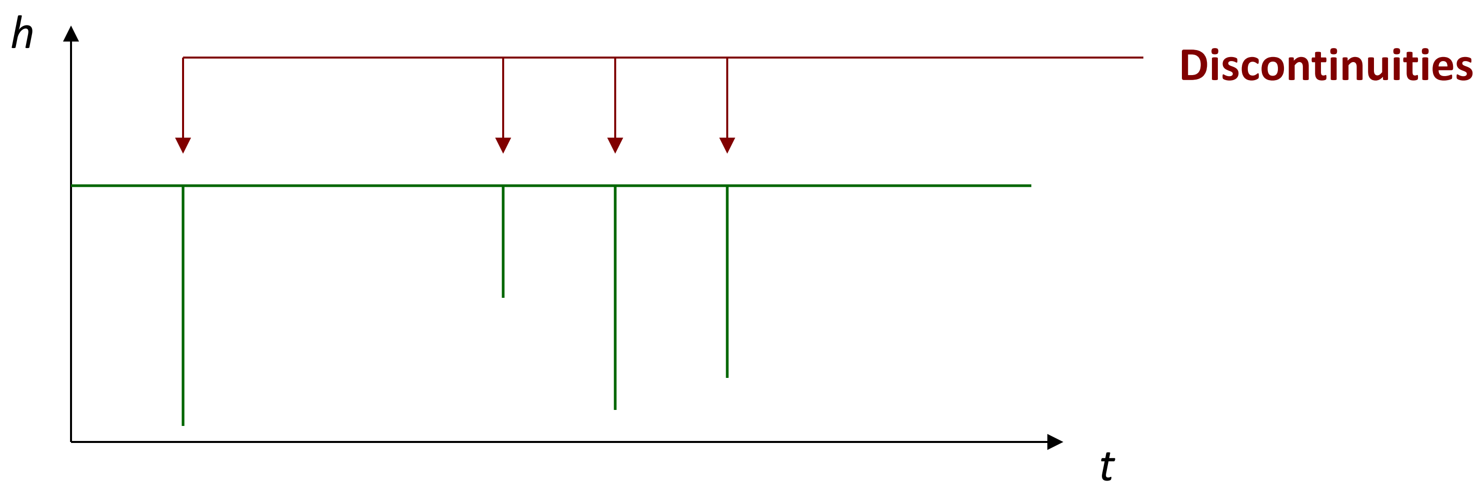
- Sometimes, the event iterations can cause errors in the evalution (division by zero, negative roots)
- Sometimes, the if-statement is used to model continuous functions.
- Hence the noEvent() clause exists: Example:
  ```
  f = noEvent(if x > 0 then sqrt(x) else -sqrt(-x));
  ```
- Here, an event iteration would be both, unnecessary and dangerous. Handling this function is now left to step-size control.



Applying Standard ODE-Solvers

Robotics and Mechatronics Centre

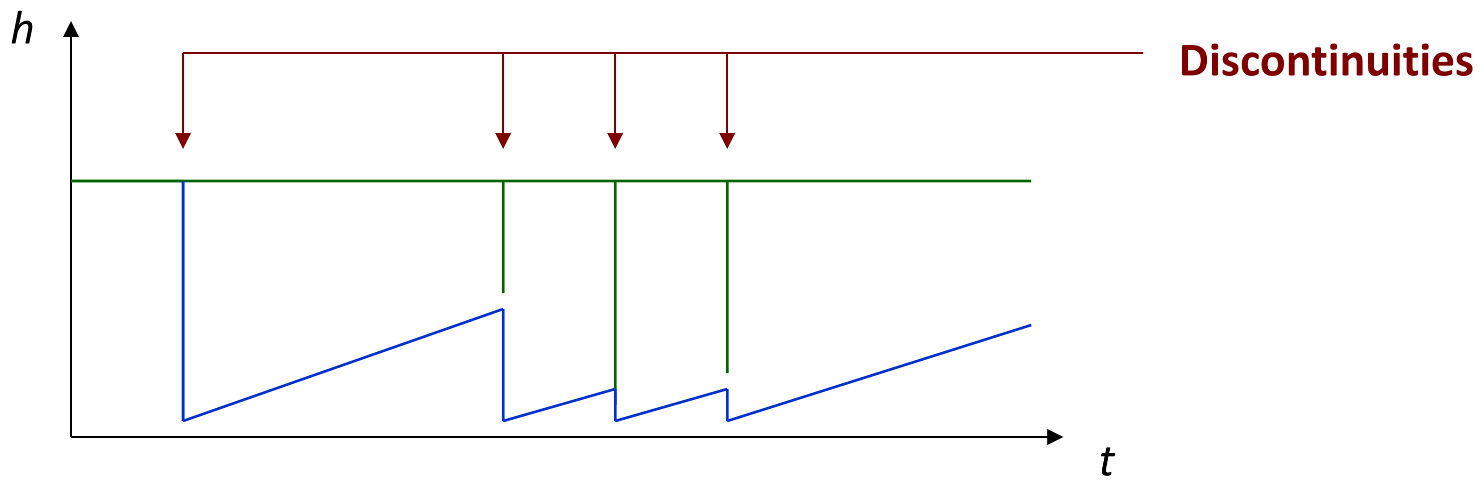
- If we know the precise location of the event, it is sufficient to reduce the size of one single step.
- After passing the discontinuity we switch the model equation and continue with the former step-size.



Applying Standard ODE-Solvers

Robotics and Mechatronics Centre

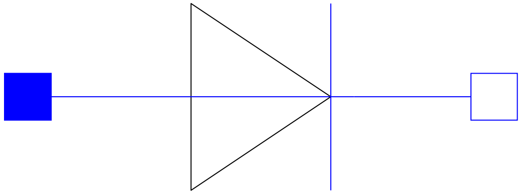
- Evidently, this is much better than abusing step-size control for the treatment of discontinuities.
- We can take much larger step-sizes.



Modeling a Diode

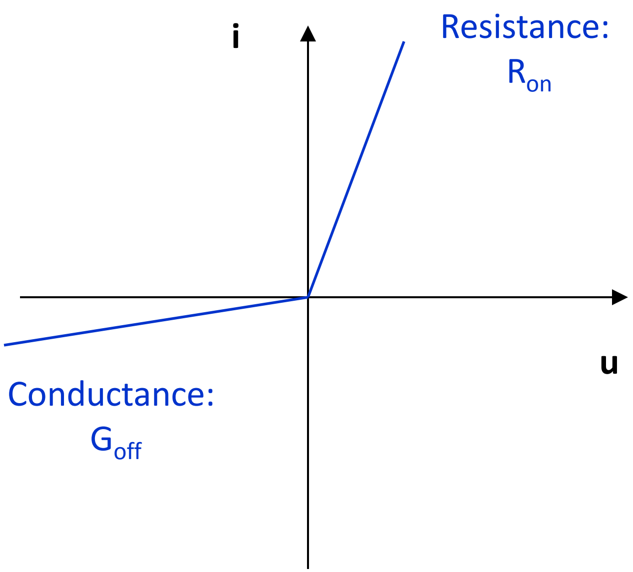
Let us see what we can model with the if-statement.

- For instance, the model of an electrical diode with the following curve.



- Here is one way to model it:

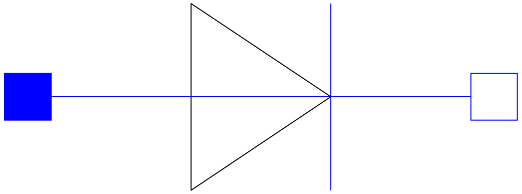
```
u = R*i
R = if u>0 then R_on else 1/G_off;
```



Modeling a Diode

A more compact form is also possible:

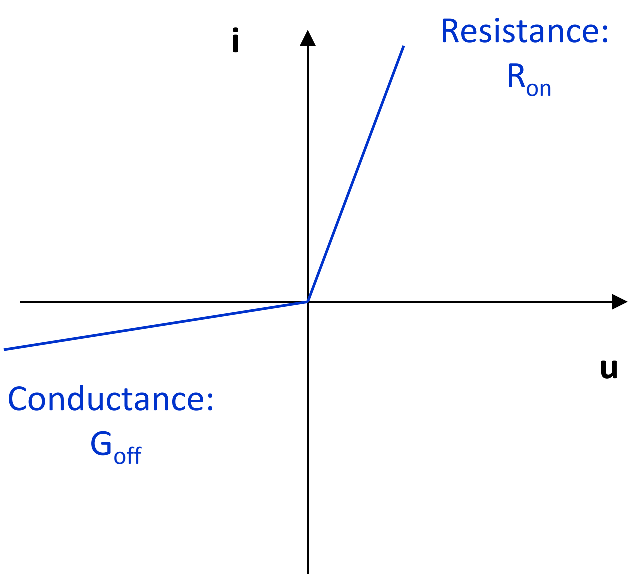
```
u = if u>0 then R_on*i else i/G_off;
```



- This is possible because if-expressions are non-causal in Dymola.
- Internally, the if-expressions may be translated into:

```
u = s*R_on*i + (1-s)*i/G_off;
```

with
s = 1 if u>0
s = 0 if u<0

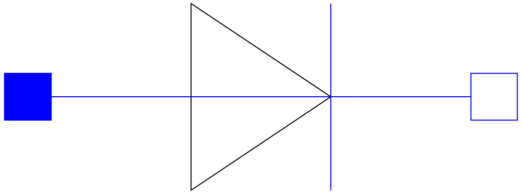


- The equation can be solved for u or i.

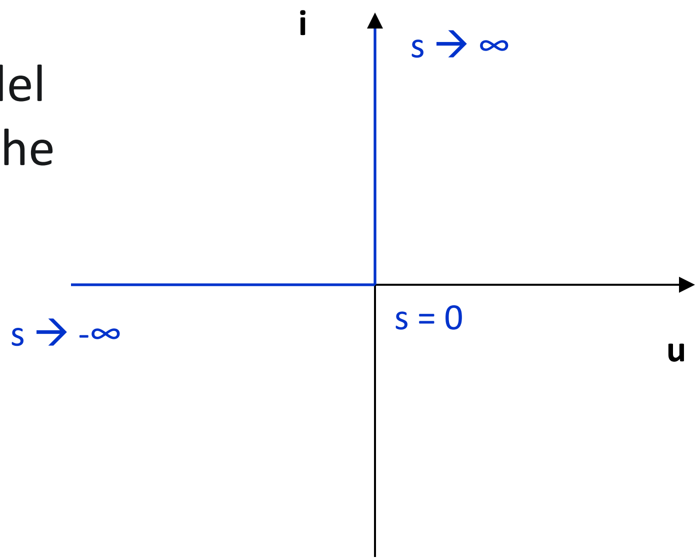
Modeling an Ideal Diode

Robotics and Mechatronics Centre

Unfortunately, a truly ideal diode cannot be modeled in this way.



- R_{on} and G_{off} are 0 for an ideal diode.
- The model would be singular in either case.
- We need a different approach. Let us model the diode by a parameterized curve with the curve parameter s.



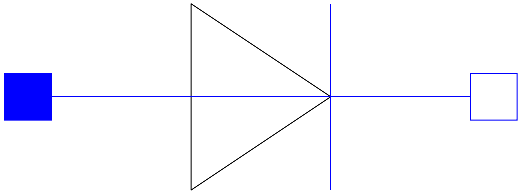
Blocking diode u=s with s < 0

Open diode i=s with s > 0

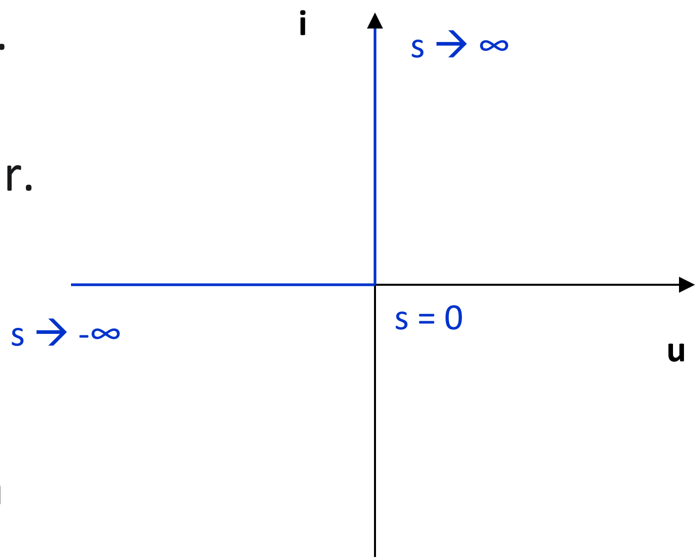
Modeling an Ideal Diode

Here are the corresponding model equations

```
u = if s>0 then 0 else s;
i = if s>0 then s else 0;
```



- These are 2 equations over 3 variables.
  Which are the 2 unknowns?
  If u is known the model is singular for u=0.
  If i is known the model is singular for i=0.
  Only if s is known the model will be regular.

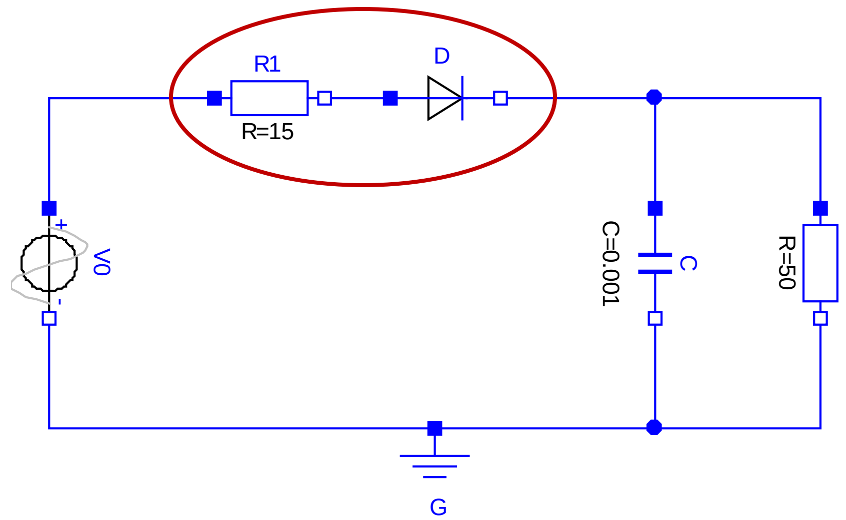


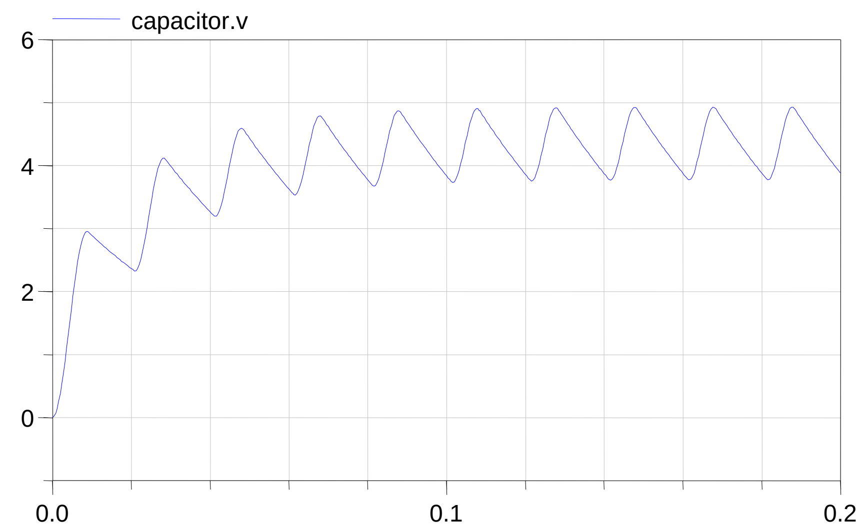
- But s depends itself on u and i. Hence the model needs to be placed in an algebraic loop and s must be chosen as tearing variable of this loop. (Fortunately, Dymola has an in-built heuristics for this…)

Halfway-Rectifier

Here is an appropriate example: the halfway-rectifier.

- The ideal diode D and the resistor R1 form an algebraic loop that determines the voltage drop between source and capacitance.
- The tearing-variable is the curve-parameter s.



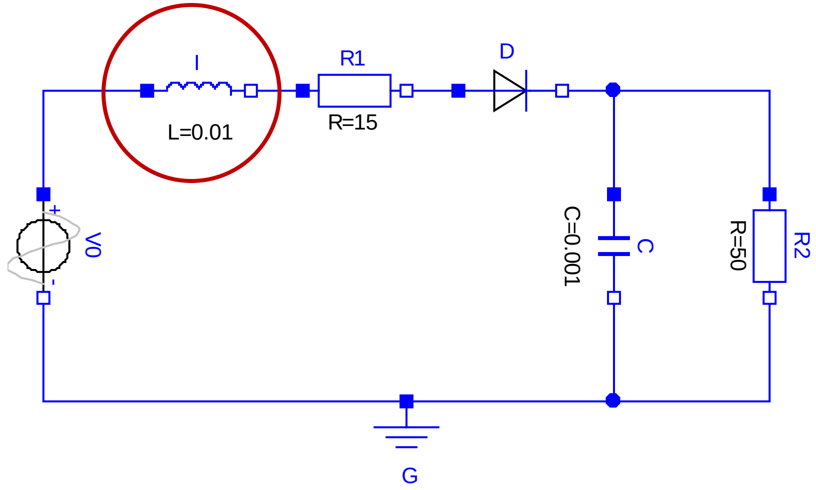


Halfway-Rect. w. Line-Inductance

Robotics and Mechatronics Centre

But if we modify this circuit slightly we run into a serious problem.

- We add an inductance in front of the diode.
- Since the natural state-variable of the inductance is the current, the causality of the resistor is fixed and the diode is not part of algebraic loop anymore.
- The simulation fails.
- Let us look closer at this problem.



The inductance L contains the differential equation:

$$di/dt*L = u$$

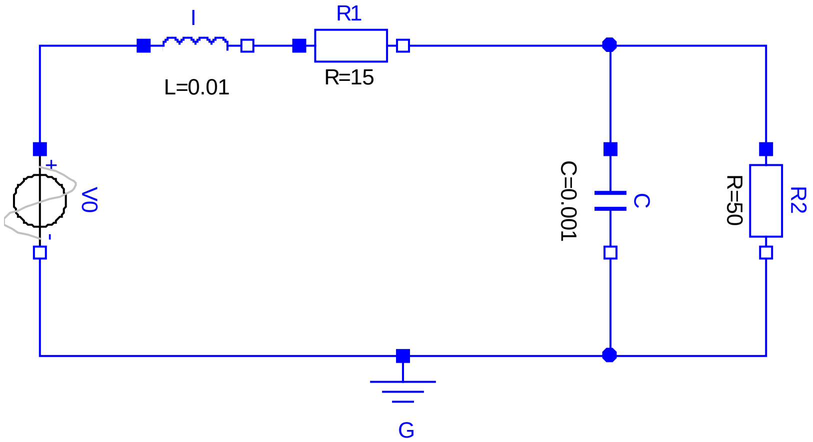
Hence, i is supposed to be known and the causality of the diode is fixed.

Halfway-Rect. w. Line-Inductance

Robotics and Mechatronics Centre

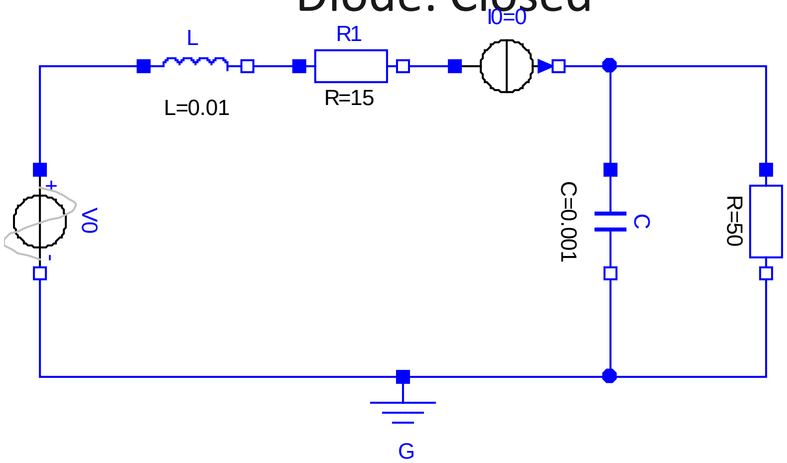
The two circuits below represent the two different states of the diode. (either fully open or fully blocking)

Diode: Open



States (C.v, L.i); Index: 0

Diode: Closed



States (C.v); Index: 1

- The two different states of the diode lead to two different system with different state variables and different perturbation index.
- A severe structural change has been caused by a seemingly harmless equation.
- Dymola is currently unable to handle such variable-structure systems.

Inline-Integration

So what can we do?

- One solution is to use a non-ideal diode and to avoid the structural change at all. However this implements an artificial stiffness into the system that may be unwanted.
- Fortunately, there is another trick: Inline Integration.
- Inline integration means that we inline the time-discrete equation of the integration algorithm into the model equations.
- To this end, we need to replace the corresponding differential equations.

Inline-Integration: Example

Let us use inline integration for the halfway rectifier with line inductance.

- We want to inline Backward Euler (BE or BDF1) into the model of the inductance.
- Hence the differential equation of the inductance:

$$di/dt * L = u$$

- gets replaced by:

$$(i_t - i_{t-h})/h*L = u_t$$

or

$$i_t = i_{t-h} + u_t/L*h$$

with i_t or u_t as potential unknowns

Inline-Integration: Example

What is the advantage of inline-integration?

- By using inline-integration with BE, we have transformed the equation of the inductance into:

  $$i_t = i_{t-h} + u_t/L*h$$

- This equation is structurally equivalent to a resistor equation. It can be solved for i_t as well as for u_t. Hence it can be also part of an algebraic loop.

- For the halfway-rectifier with line-inductance this means that the equations of the inductance L, the resistor R1, and the Diode D form one algebraic loop using the curve parameter s as tearing variable.

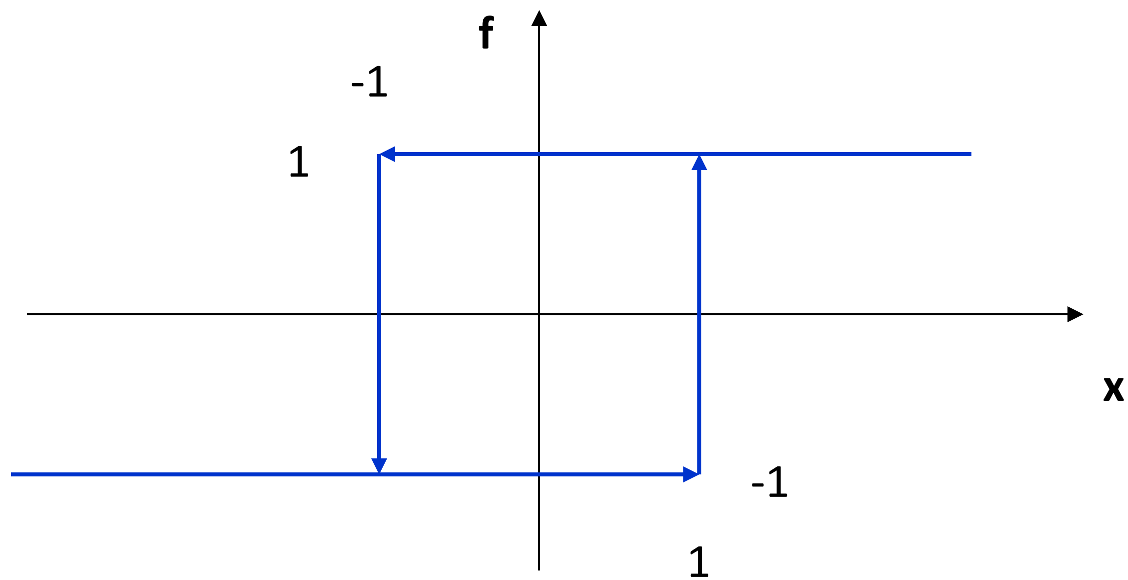
- This kind of inline-integration is also not supported by Dymola. Dymola may perform inline-integration but after the differential index-reduction has taken place. Hence this trick does currently not work in Dymola.

Multi-Valued Functions

Robotics and Mechatronics Centre

So far, we have only looked at events that could be modeled by if-expressions. However, also multi-valued functions do frequently occur in engineering systems.

- One example is a function for a hysteretic controller (As used, for instance, in a refrigerator or many other devices that require a binary control).



Multi-Valued Functions

To model such functions, the when-statement has been introduced in Modelica.

```
when x > 10 then
        y = -10
end when;
```

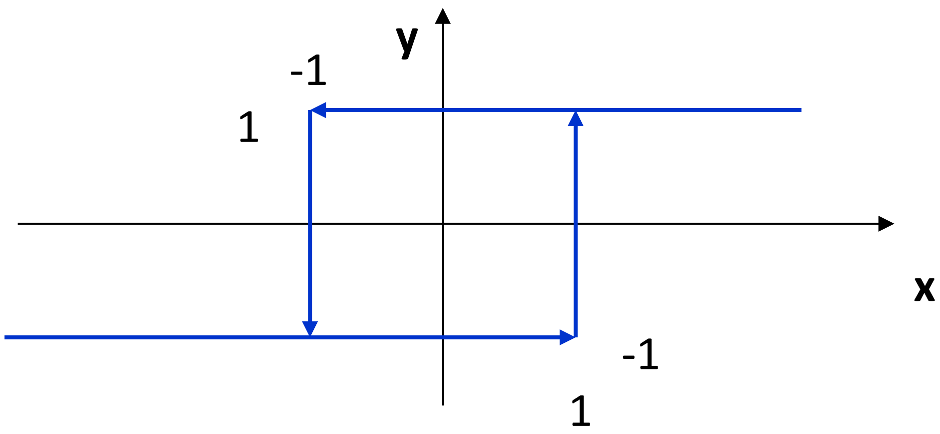
- The when statement becomes active exactly when its condition becomes true.
- The equation is rather an assignment: The unknown must be placed on the left.
- The equation is only active for this particular time-instance. Right after, it is deactivated again.
- The value of the unknown is held constant until the next activation of the same when-statement.

Multi-Valued Functions

Robotics and Mechatronics Centre

Hence, the following code seems appropriate to model the hysteresis.

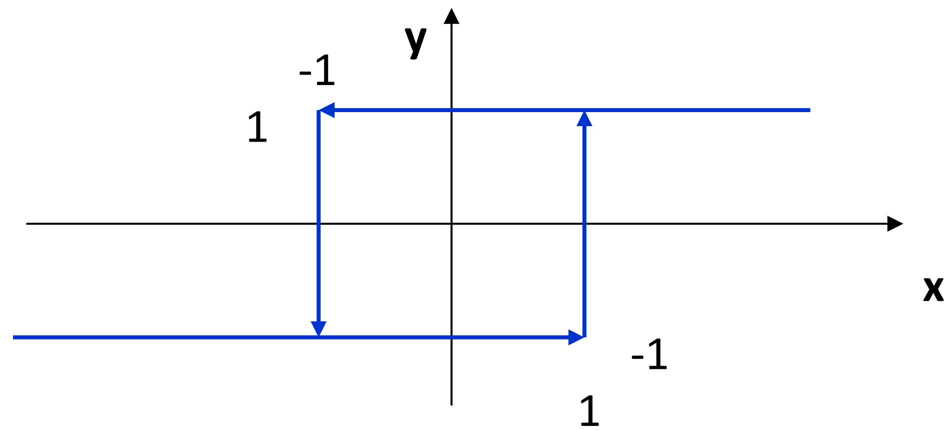
```
when x>1 then
      y = 1;
end when;

when x<-1 then
      y = -1;
end when;
```



Multi-Valued Functions

Hence, the following code seems appropriate to model the hysteresis.

```
when x>1 then
      y = 1;
end when;

when x<-1 then
      y = -1;
end when;
```



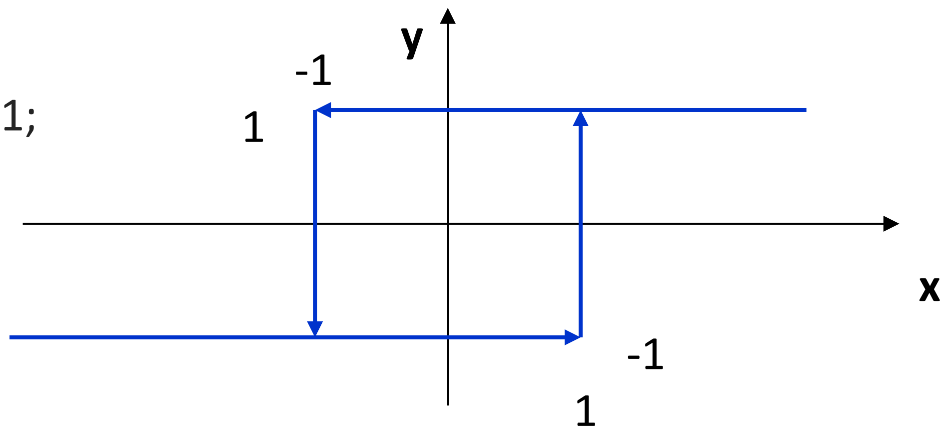
- However, this is illegal in Modelica since the variable y is determined in two distinct when-statements. In order to avoid problems with simultaneous events, this is not allowed.

- Of course, these two events are mutually exclusive, but Dymola does not know this and it is impossible in general to derive this automatically.

Multi-Valued Functions

Here is an alternative formulation:

when x>1 **or** x<-1 **then**
    y = **if** x>0 **then** 1 **else** -1;
end when;

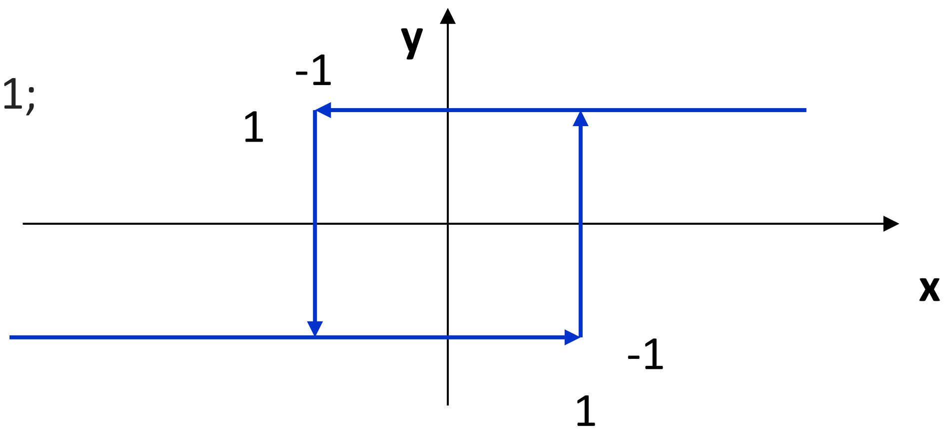


- This is perfectly legal. We have simply merged the two events into a single when-statement.
- By doing so, we have created another problem. Given a large step-size we might jump directly from x=-1 to x=1. In this case, no event is triggered at all.

Multi-Valued Functions

Here is an alternative formulation:

```
when {x>1, x< -1} then
        y = if x>0 then 1 else -1;
end when;
```



- This is perfectly legal. We have simply merged the two events into a single when-statement.
- By doing so, we have created another problem. Given a large step-size we might jump directly from x=-1 to x=1. In this case, no event is triggered at all.
- To cope with this problem, Modelica enables to state a condition-vector. Now, we are fine.

Multi-Valued Functions

Robotics and Mechatronics Centre

In the Modelica Standard Library, the hysteresis is modeled even differently:

y = **if** x > 1 **or** (pre(y>0) **and** (x>=-1)) **then** 1 **else** -1;

- The operator pre(…) can be used in order to access the value of a variable just right before the event.
- Using this operator, we can formulate multi-valued functions without the use of when-statements.
- In fact, the statement:

  ```
  when g(…) then
     y = f(…);
  end when;
  ```

  is internally transformed to….

  ```
  if g(…) and not pre(g(…)) then
     y = f(…);
  else
     y = pre(y);
  end if;
  ```

Discrete State Changes

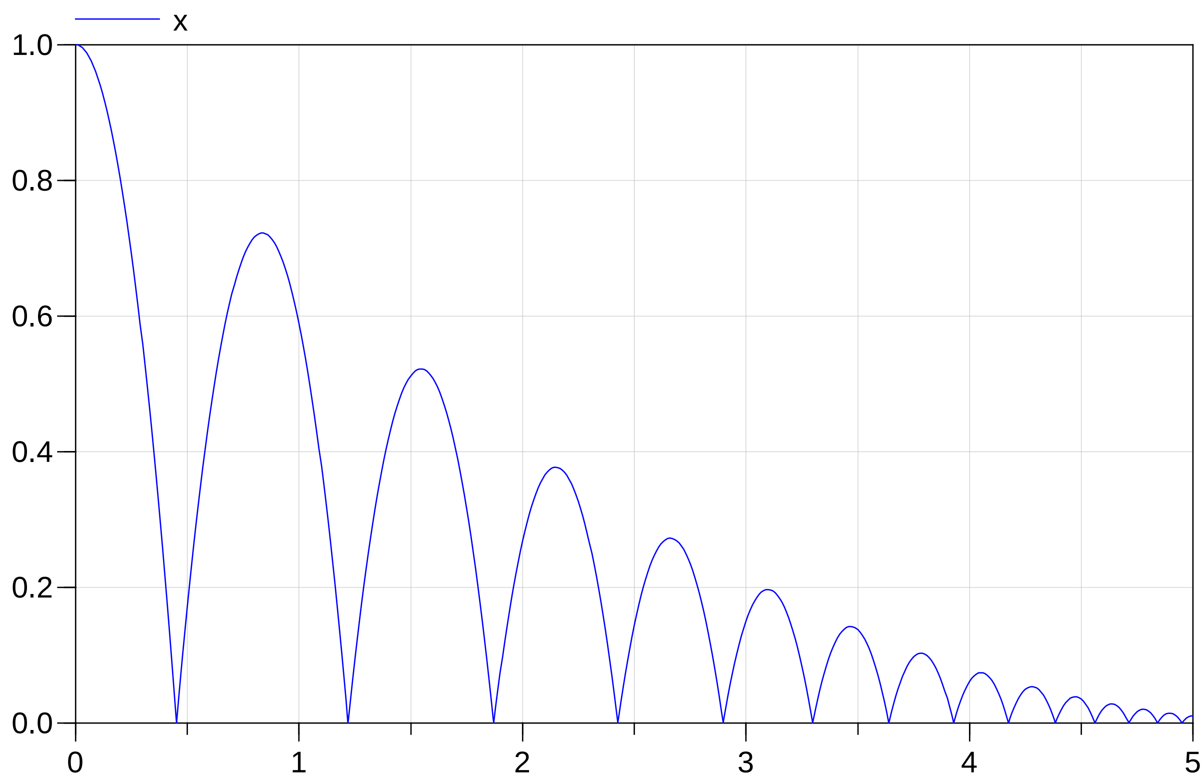
So far, we have only looked at discrete changes in the function $f(\mathbf{x}(t),t)$

$$d\mathbf{x}/dt = f(\mathbf{x}(t),\mathbf{u},t)$$

- But there are also cases where the actual state is changing discretely (e.g. mechanical collisions/impulses) . Here $d\mathbf{x}/dt$ becomes of infinite value. What shall we do?
- This problem corresponds to the re-initialization of the system.
- In current Modelica, this is only weakly supported by the function `reinit(state, newValue)`.
- Let us look at an example: The bouncing ball.

Bouncing Ball

Let us model a bouncing ball that is being dropped from an initial height and is bouncing on a table.



Initializing the Revolute Joint

Robotics and Mechatronics Centre

Let us model a bouncing ball that is being dropped from an initial height and is bouncing on a table.

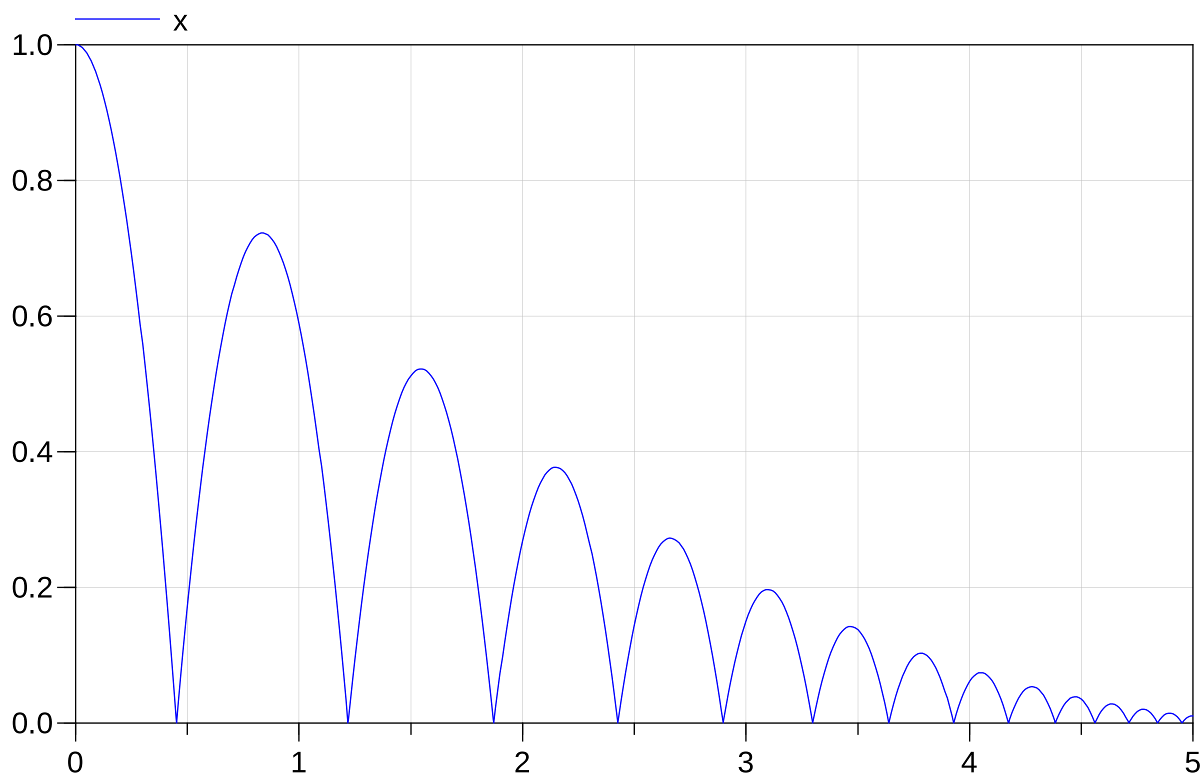
- The motion is described by the variables x, v, and a.
- The elasticity of the impulse is determined by the coefficient μ.
- The reinit command is used in a when-clause.
- The pre(…) operator is used to access the prior value of v in order to compute the new velocity.

```
model BouncingBall

  Real x;
  Real v;
  Real a;

  parameter Real mu = 0.85;


initial equation
  v = 0;
  x = 1;

equation
  v = der(x);
  a = der(v);
  a = -9.81;

  when x<0 then
    reinit(v,-mu*pre(v));
  end when;

end BouncingBall;
```

Bouncing Ball

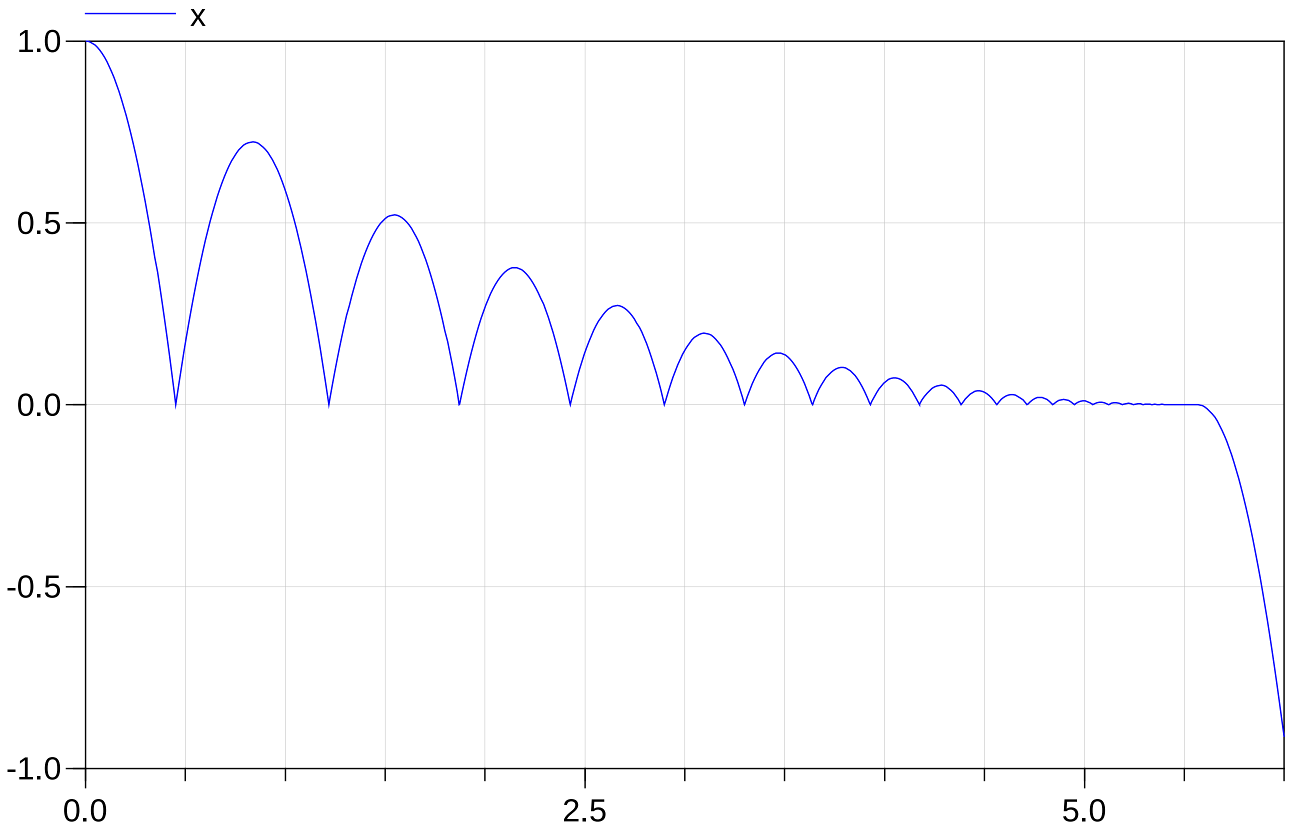
Robotics and Mechatronics Centre

This looks fine. But what happens if we simulate for longer time periods?



Bouncing Ball

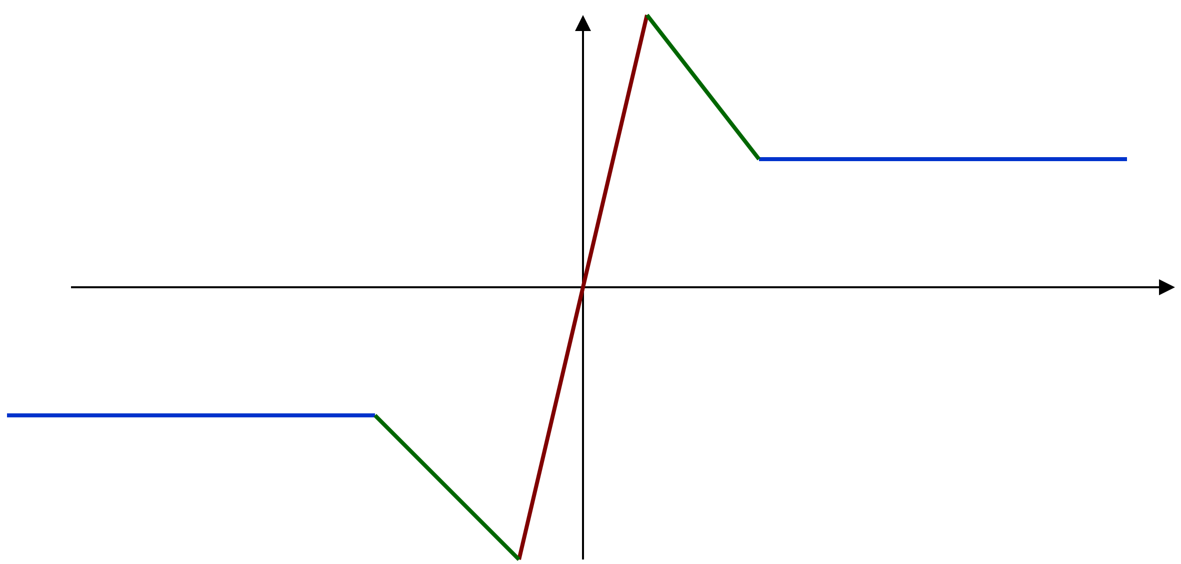
OOOPS!?! This is a common problem among many simulators. The increasingly smaller bounces lead to a failure in the event detection. Modeling a resting state by events is evidently not a good idea.



Dry Friction Revisited

Let us combine what we have learned today by modeling an ideal model for dry-friction.

- For the characteristic curve, we have used so far a regularization. Here is a piecewise linear regularization:

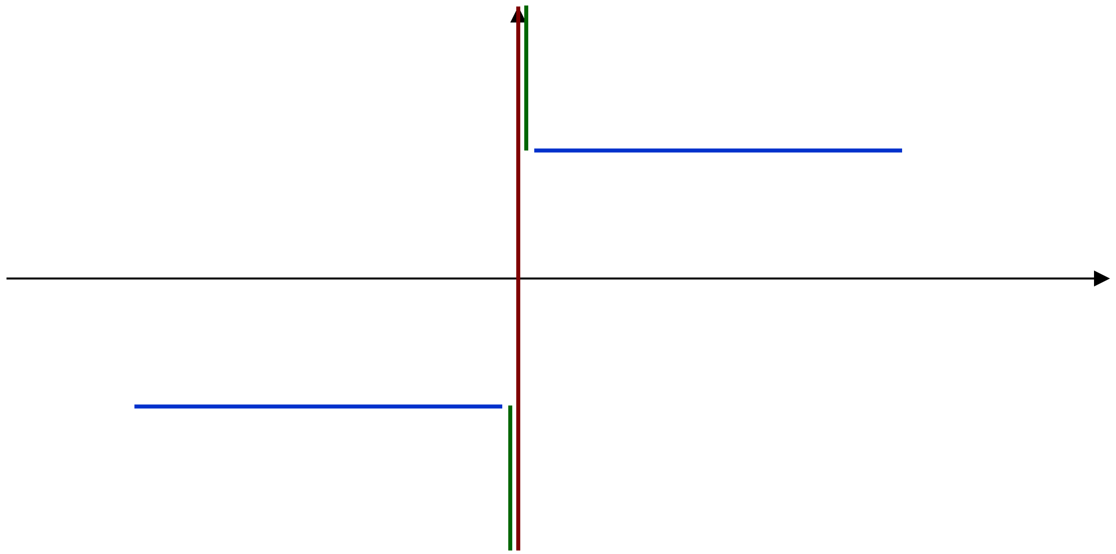


Dry Friction Revisited

Robotics and Mechatronics Centre

Let us combine what we have learned today by modeling an ideal model for dry-friction.

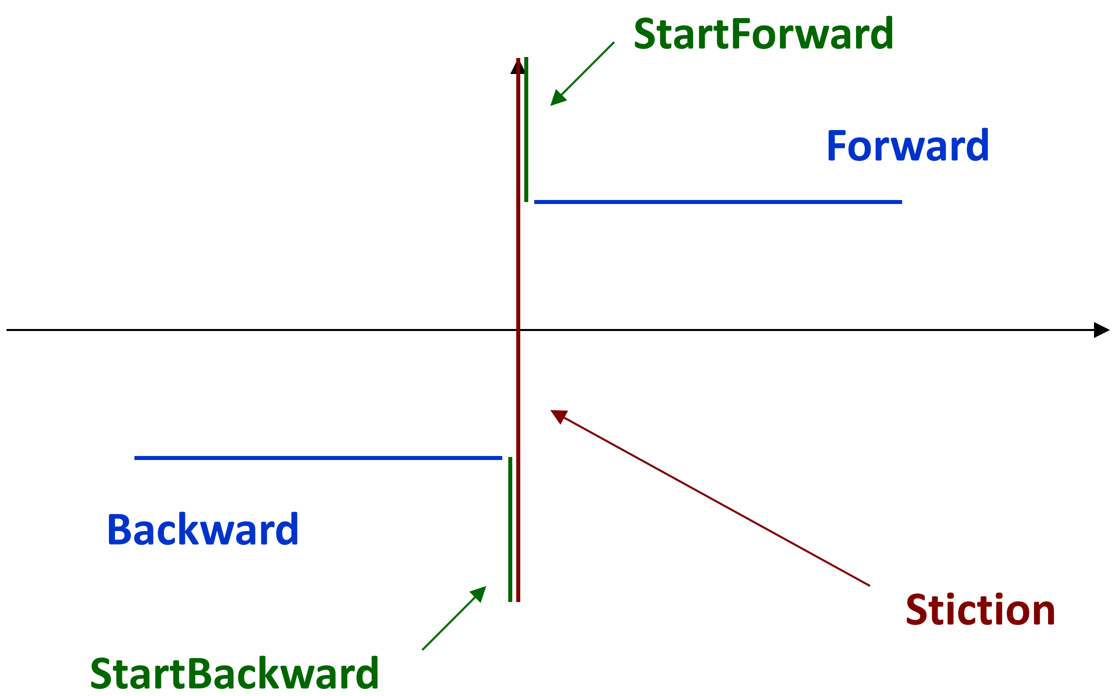
- In the ideal model, this is a multi-valued function.



Modeling Dry Friction

Let us combine what we have learned today by modeling an ideal model for dry-friction.

- In the ideal model, this is a multi-valued function.
- The function contains several modes:

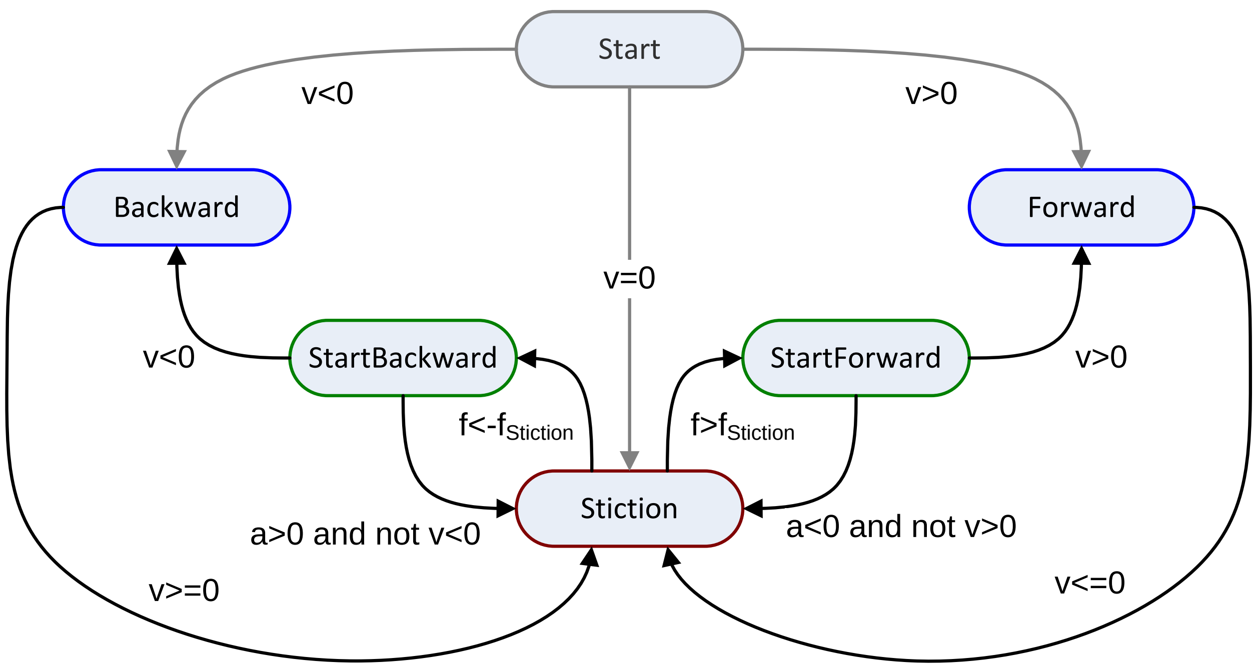


Dry Friction: Mode-Transitions

Robotics and Mechatronics Centre

We need to carefully model the transitions between these modes.

- This can be prepared by a mode-transition diagram:



Modeling Dry Friction

Robotics and Mechatronics Centre

Let us setup the model:

- We use the standard translational interface and derive the velocity and acceleration.
- Two parameters values describe the friction characteristics.
- The modes are represented by a set of Boolean variables.

```
model DryFriction
  parameter SI.Force S = 10;
  parameter SI.Force R = 8;
  Flange_a flange_a;
  SI.Velocity v;
  SI.Acceleration a;
  SI.Force fR;

  Boolean Stiction;
  Boolean StartForw;
  Boolean Forward;
  Boolean StartBack;
  Boolean Backward;


equation
  v = der(flange_a.s);
  a = der(v);
  […]




end DryFriction;
```

Modeling Dry Friction

Let us setup the model:

- The friction force (flange_a.f) is now dependent on the current mode.

```
model DryFriction
  parameter SI.Force S = 10;
  parameter SI.Force R = 8;
  Flange_a flange_a;
  SI.Force fR;
  […]

equation
  […]

  flange_a.f =
    if Forward then R
    else if Backward then - R
    else if StartForw then R
    else if StartBack then -R
    else fR;


  0 =
   if Stiction or initial() then a
   else fR;


end DryFriction;
```

Modeling Dry Friction

Robotics and Mechatronics Centre

Let us setup the model:

- The friction force (flange_a.f) is now dependent on the current mode.
- The internal operator `initial()` becomes true just at the moment of initialization. Otherwise, it is false.
- Initially or at Stiction, the acceleration is set to zero and the friction force fR is free.
- The conditional constraint a=0 should actually be v=0 at least or s=const, but this would cause a structural change and cannot be handled by Modelica/Dymola.

```
model DryFriction
  parameter SI.Force S = 10;
  parameter SI.Force R = 8;
  Flange_a flange_a;
  SI.Force fR;
  […]

equation
  […]

  flange_a.f =
    if Forward then R
    else if Backward then - R
    else if StartForw then R
    else if StartBack then -R
    else fR;

  0 =
   if Stiction or initial() then a
   else fR;

end DryFriction;
```

Modeling Dry Friction

Robotics and Mechatronics Centre

Let us setup the model:

- Now we have to model the mode-transitions according to the diagram.
- We can use the pre() operator for this purpose.
- All states must be exclusive.

```
model DryFriction
  parameter SI.Force S = 10;
  parameter SI.Force R = 8;
  Flange_a flange_a;
  SI.Force fR;
  […]

equation
  […]
  Forward = initial() and v > 0 or
            pre(StartForw) and v > 0 or
            pre(Forward) and not v <= 0;
  Backward = initial() and v < 0 or
             pre(StartBack) and v < 0 or
             pre(Backward) and not v >= 0;
  StartForw = pre(Stiction) and fR > S or
              pre(StartForw) and not
              (v>0 or a<=0 and not v>0);
  StartBack = pre(Stiction) and fR<- S or
              pre(StartBack) and not
              (v<0 or a>=0 and not v<0);
  Stiction = not (Forward or Backward or
                  StartForw or StartBack);
end DryFriction;
```

Modeling Dry Friction

Let us setup the model:

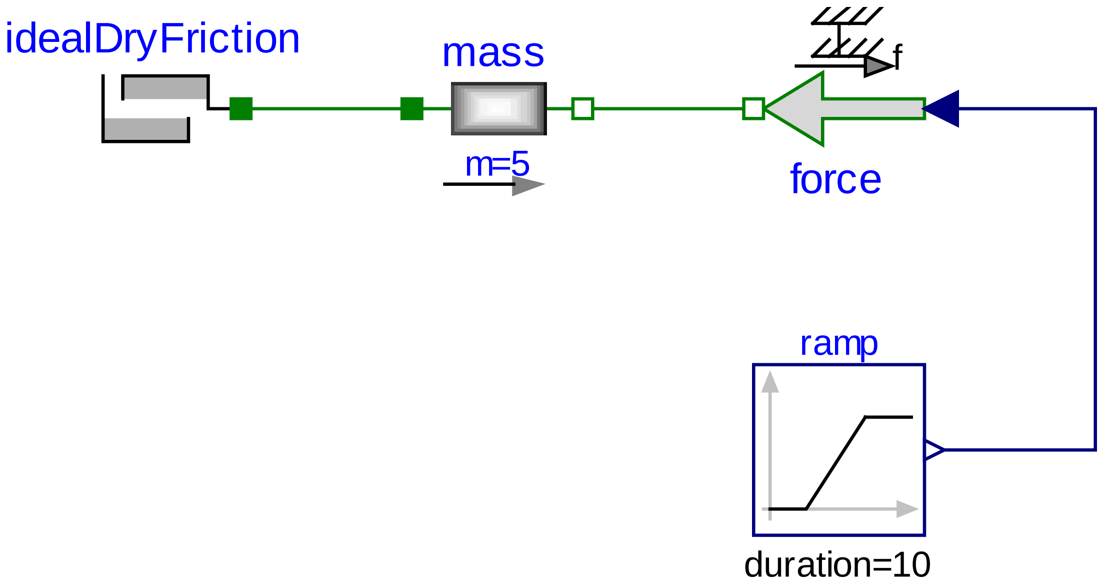
- Finally, there is a last issue:
- When the velocity crosses zero and stiction is enforced, we need to set the velocity explicitly to zero.
- To this end, we use the reinit()-command. Hence v must be a state-variable.

```
model DryFriction
  parameter SI.Force S = 10;
  parameter SI.Force R = 8;
  Flange_a flange_a;

  SI.Velocity v(
    stateSelect=StateSelect.always
  );

  […]



equation

  […]

  when Stiction and not initial() then
   reinit(v,0);
  end when;



end DryFriction;
```

Simulating Dry Friction

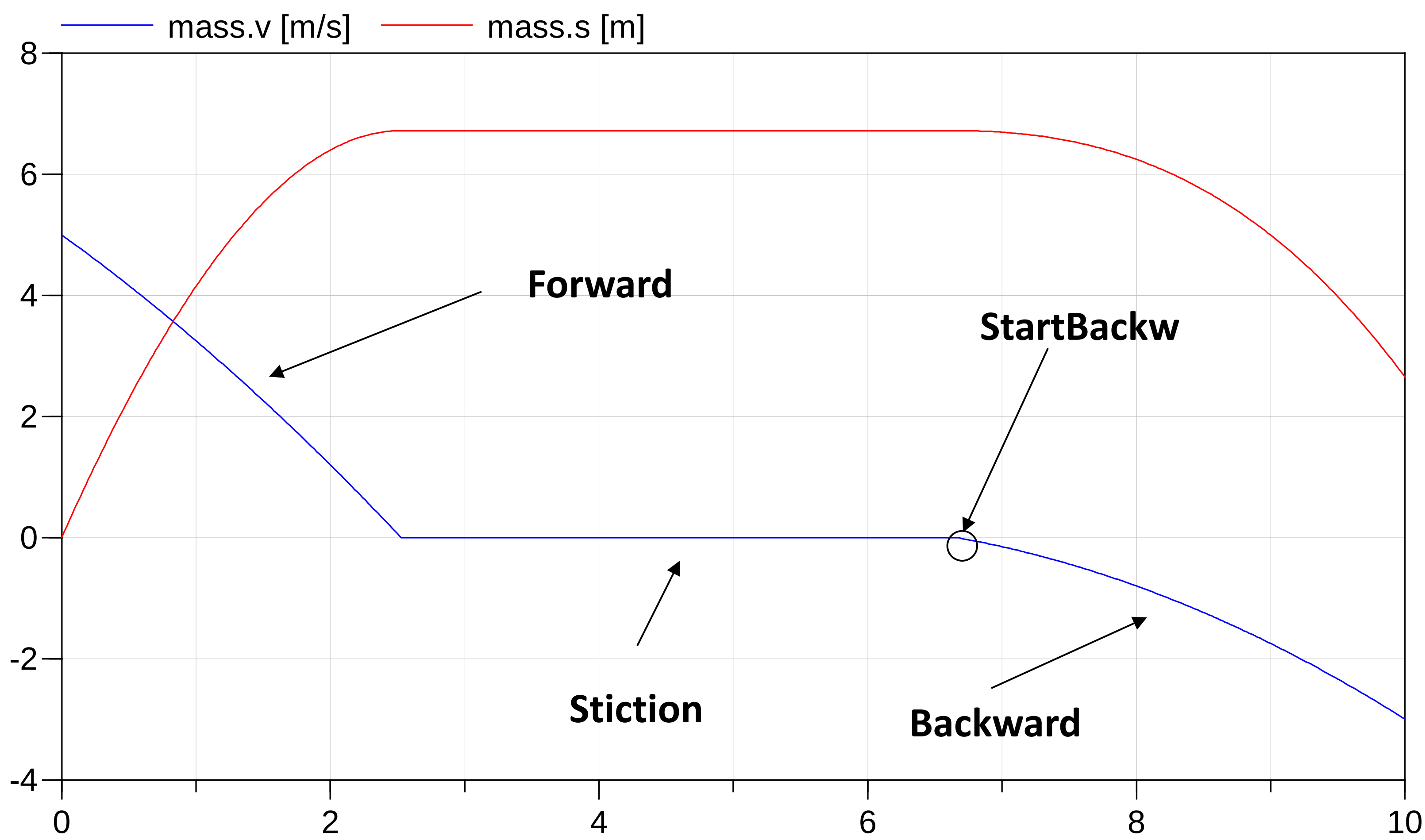
Let us test our dry-friction model:

- The mass (5kg) has an initial speed of 5m/s
- The (negative) force is ramped up from 0 to 15N



Simulating Dry Friction

Here is the simulation result:

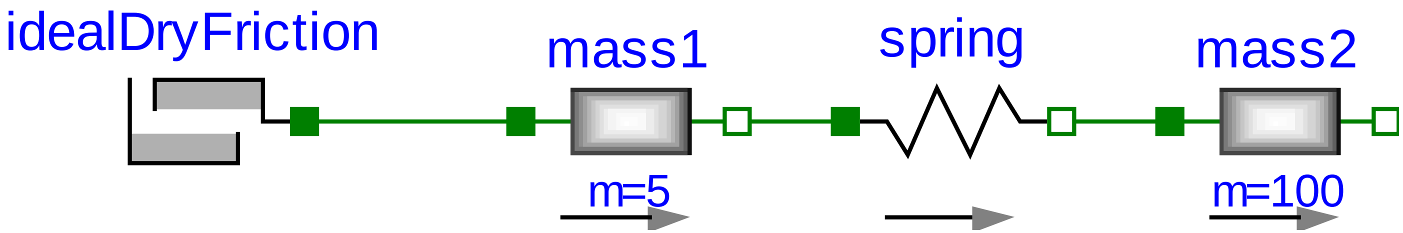


Simulating Dry Friction

Robotics and Mechatronics Centre

This system is more fun.

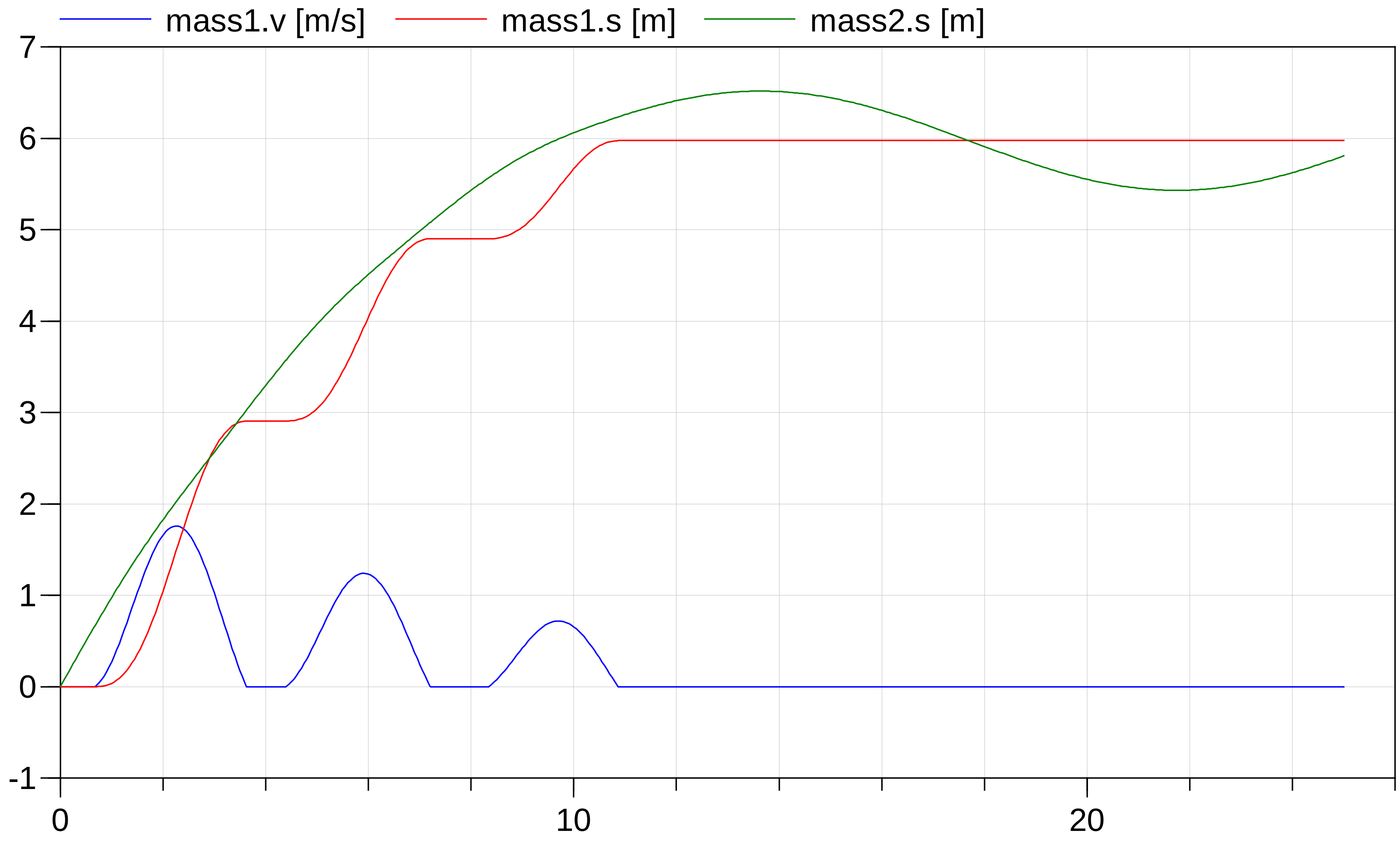
- Mass1 (5kg) is initially at rest.
- Mass2 (100kg) starts with v=1m/s.



Simulating Dry Friction

Robotics and Mechatronics Centre

Here is the simulation result:



Questions ?