# Treatment of Discontinuities

- Today, we shall look at the problem of dealing with discontinuities in models.
- 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.





# Table of Contents


- Numerical differential equation solvers
- Discontinuities in state equations
- Integration across discontinuities
- State events
- Event handling
- Multi-valued functions
- The electrical switch
- The ideal diode
- Friction






# Numerical Differential Equation Solvers

- Most of the ***differential equation solvers*** that are currently on the market operate on ***polynomial extrapolation***.
- The value of a state variable $x$ at time $t+h$, where $h$ is the current ***integration step size***, is approximated by fitting a ***polynomial of $n^{th}$ order*** through known supporting values of $x$ and $dx/dt$ at the current time $t$ as well as at past instances of time.
- The value of the extrapolation polynomial at time $t+h$ represents the approximated solution of the differential equation.
- In the case of ***implicit integration algorithms***, the state derivative at time $t+h$ is also used as a supporting value.





# Examples

***Explicit Euler Integration Algorithm of 1st Order:***

$$x(t+h) \approx x(t) + h \cdot \dot{x}(t)$$

***Implicit Euler Integration Algorithm of 1st Order:***

$$x(t+h) \approx x(t) + h \cdot \dot{x}(t+h)$$

# Discontinuities in State Equations

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

$$\dot{x}(t) = f(x(t),t)$$

exhibit a discontinuity, the polynomial extrapolation is a very poor approximation of reality.

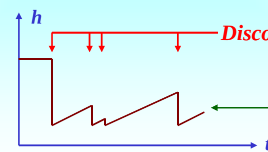
- Consequently, ***integration algorithms*** with a fixed step size exhibit a large ***integration error***, whereas integration algorithms with a variable step size reduce the step size dramatically in the vicinity of a discontinuity.





# Integration Across Discontinuities

- 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*** on one hand and a ***point of large local stiffness*** (with a large absolute value of the derivative) on the other.




***The step size is constantly too small. Thus, the integration algorithm is at least highly inefficient, if not even inaccurate.***

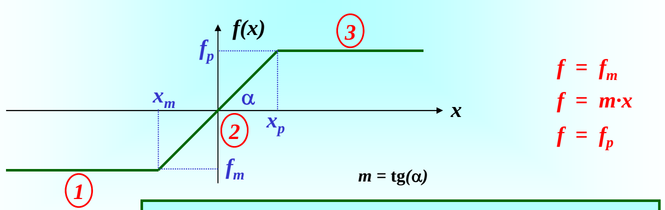




# The State Event

- These problems can be avoided by telling the integration algorithm explicitly, when and where discontinuities are contained in the model description.

***Example: Limiter Function***



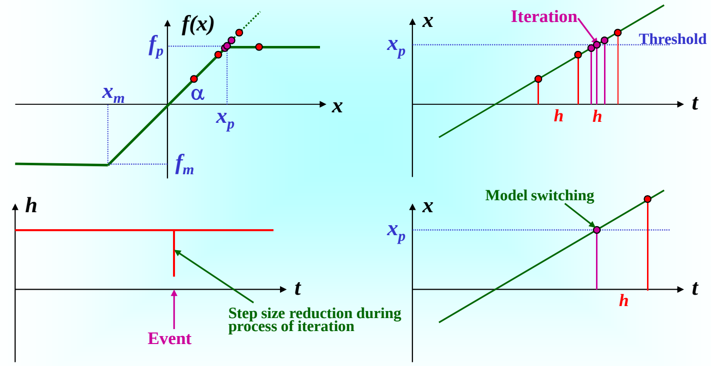

$$f = f_m$$
$$f = m \cdot x$$
$$f = f_p$$

**f = if x < xm then fm else if x < xp then m*x else fp ;**

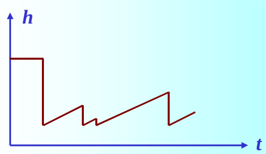




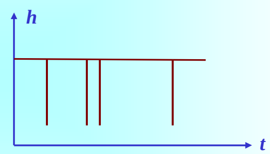
# Event Handling I



# Event Handling II




Step size as function of time without event handling




Step size as function of time with event handling





# Representation of Discontinuities

*f* = **if** *x* < *xm* **then** *fm* **else if** *x* < *xp* **then** *m*x* **else** *fp* ;

- In ***Modelica***, discontinuities are represented as ***if-statements***.
- In the process of translation, these statements are transformed into correct ***event descriptions*** (sets of ***models with switching conditions***).
- The modeler does not need to concern him- or herself with the mechanisms of event descriptions. These are hidden behind the ***if-statements***.





# Problems

- The modeler needs to take into account that the discontinuous solution is temporarily left during iteration.

$q = \sqrt{|\Delta p|}$

$\Delta p = p_1 - p_2$ ;
*absΔp* = **if** *Δp* > *0* **then** *Δp* **else** *–Δp* ;
*q* = **sqrt**(*absΔp*) ;

may be dangerous, since ***absΔp*** can become temporarily negative.

⇒ $\Delta p = p_1 - p_2$ ;
*absΔp* = **noEvent(** **if** *Δp* > *0* **then** *Δp* **else** *–Δp* **)** ;
*q* = **sqrt**(*absΔp*) ;

solves this problem.





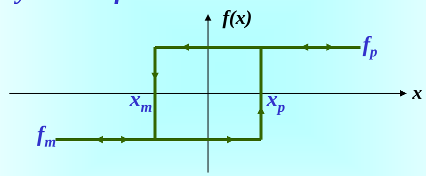
# The "noEvent" Construct

$\Delta p = p_1 - p_2$ ;
*absΔp* = **noEvent(** **if** *Δp* > *0* **then** *Δp* **else** *–Δp* **)** ;
*q* = **sqrt**(*absΔp*) ;

- The ***noEvent construct*** has the effect that ***if-statements*** or ***Boolean expressions***, which normally would be translated into simulation code containing correct event handling instructions, are handed over to the integration algorithm untouched.
- Thereby, management of the simulation across these discontinuities is left to the step size control of the numerical Integration algorithm.

# Multi-valued Functions I

- The language constructs that have been introduced so far don't suffice to describe ***multi-valued functions***, such as the ***dry hysteresis function*** shown below.

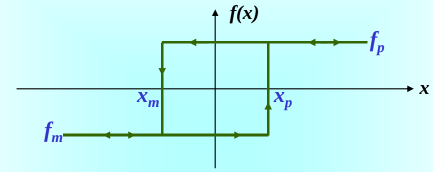



- When *x* ***becomes greater*** than $x_p$, *f* must be switched from $f_m$ to $f_p$.
- When *x* ***becomes smaller*** than $x_m$, *f* must be switched from $f_p$ to $f_m$.





# Multi-valued Functions II




```
when initial() then
  reinit(f , fp);
end when;
when  x > xp  or  x < xm  then
  f  =  if  x > 0  then  fp  else  fm;
end when;
```

*becomes larger* / *is larger*

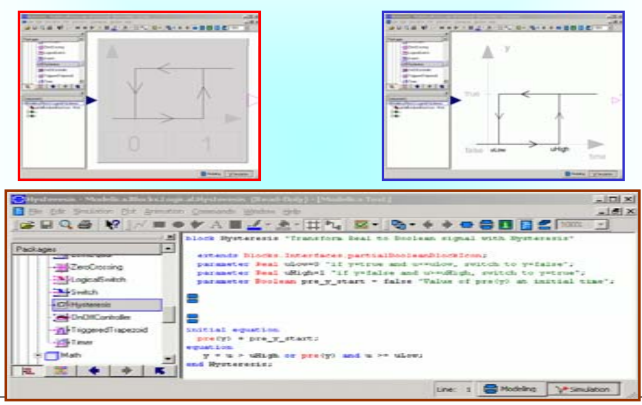
**Executed *at the beginning* of the simulation.**

**These statements are only executed, when either *x* *becomes larger than* $x_p$, or when *x* *becomes smaller than* $x_m$.**





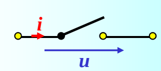
# Multi-valued Functions III







# The Electrical Switch I




When the switch is ***open***, the current is ***i=0***.
When the switch is ***closed***, the voltage is ***u=0***.

***0 = if open then i else u ;***

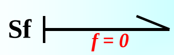
The ***if-statement*** in ***Modelica*** is ***a-causal***. It is being sorted together with all other statements.

# The Electrical Switch II

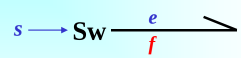
***Possible Implementation:*** Switch open: $s = 1$
Switch closed: $s = 0$

$$\Rightarrow \quad 0 = s \cdot i + (1 - s) \cdot u$$

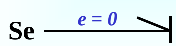
Switch open:




$\Rightarrow$




Switch closed:

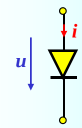


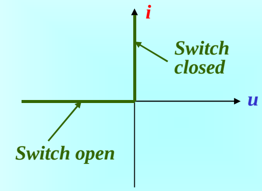

*The causality of the switch element is a function of the value of the control signal s.*





# The Ideal Diode I

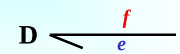



When $u < 0$, the switch is open. No current flows through.

When $u > 0$, the switch is closed. Current may flow. The ideal diode behaves like a short circuit.

*open* = *u* < *0* ;
*0* = **if** *open* **then** *i* **else** *u* ;








# The Ideal Diode II

- Since current flowing through a diode cannot simply be interrupted, it is necessary to slightly modify the diode model.

*open* = *u* <= *0* **and not** *i* > *0* ;
*0* = **if** *open* **then** *i* **else** *u* ;

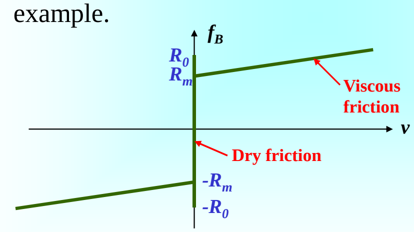
- The variable ***open*** must be declared as ***Boolean***. The value to the right of the Boolean expression is assigned to it.





# The Friction Characteristic I

- More complex phenomena, such as friction characteristics, must be carefully analyzed case by case.
- The approach is discussed here by means of the friction example.




When $v \neq 0$, the friction force is a function of the velocity.

When $v = 0$, the friction force is computed such that the velocity remains $0$.

# The Friction Characteristic II

- We distinguish between five situations:

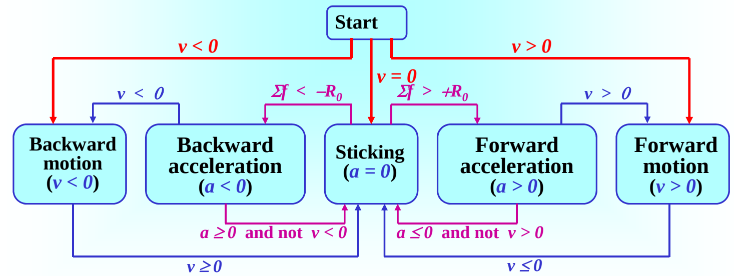
| | |
|---|---|
| $v = 0$<br>$a = 0$ | ***Sticking:*** The friction force compensates the sum of all forces attached, except if $\lvert \Sigma f \rvert > R_0$. |
| $v > 0$ | ***Moving forward:*** The friction force is computed as: $f_B = R_v \cdot v + R_m$. |
| $v < 0$ | ***Moving backward:*** The friction force is computed as: $f_B = R_v \cdot v - R_m$. |
| $v = 0$<br>$a > 0$ | ***Beginning of forward motion:*** The friction force is computed as: $f_B = R_m$. |
| $v = 0$<br>$a < 0$ | ***Beginning of backward motion:*** The friction force is computed as: $f_B = -R_m$. |





# The State Transition Diagram

- The set of events can be described by a ***state transition diagram***.








# The Friction Model I

```
model Friction;
  parameter Real R0, Rm, Rv;
  parameter Boolean ic=false;
  Real fB, fc;
  Boolean Sticking (final start = ic);
  Boolean Forward (final start = ic), Backward (final start = ic);
  Boolean StartFor (final start = ic), StartBack (final start = ic);

  fB = if Forward   then Rv*v + Rm else
       if Backward then Rv*v - Rm  else
       if StartFor   then Rm          else
       if StartBack  then -Rm         else fc;
  0 = if Sticking or initial() then a else fc;
```





# The Friction Model II

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

Forward =  initial()        and v > 0 or
           pre(StartFor)    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;
```

## The Friction Model III

```
  StartFor  = pre(Sticking)   and fc > R0 or
              pre(StartFor)   and not (v > 0 or a <= 0 and not v > 0);
  StartBack = pre(Sticking)   and fc < -R0 or
              pre(StartBack)  and not (v < 0 or a >= 0 and not v < 0);
  Sticking = not (Forward or Backward or StartFor or StartBack);

end Friction;
```





## References I

- Cellier, F.E. (1979), *Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools*, PhD Dissertation, Swiss Federal Institute of Technology, ETH Zürich, Switzerland.
- Elmqvist, H., F.E. Cellier, and M. Otter (1993), "Object-oriented modeling of hybrid systems," *Proc. ESS'93, SCS European Simulation Symposium*, Delft, The Netherlands, pp.xxxi-xli.
- Cellier, F.E., M. Otter, and H. Elmqvist (1995), "Bond graph modeling of variable structure systems," *Proc. ICBGM'95, 2nd SCS Intl. Conf. on Bond Graph Modeling and Simulation*, Las Vegas, NV, pp. 49-55.





## References II

- Elmqvist, H., F.E. Cellier, and M. Otter (1994), "Object-oriented modeling of power-electronic circuits using Dymola," *Proc. CISS'94, First Joint Conference of International Simulation Societies*, Zurich, Switzerland, pp. 156-161.
- Glaser, J.S., F.E. Cellier, and A.F. Witulski (1995), "Object-oriented switching power converter modeling using Dymola with event-handling," *Proc. OOS'95, SCS Object-Oriented Simulation Conference*, Las Vegas, NV, pp. 141-146.