Numerical Simulation of Dynamic Systems XXII

Prof. Dr. François E. Cellier
Department of Computer Science
ETH Zurich

May 7, 2013

Event Descriptions of Discontinuous Functions

In the previous presentation, we have created a numerical framework for safely dealing with discontinuities in model descriptions.

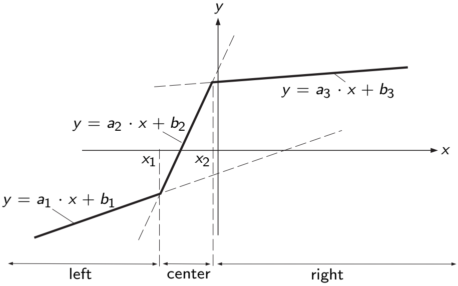
Event Descriptions of Discontinuous Functions

In the previous presentation, we have created a numerical framework for safely dealing with discontinuities in model descriptions.

In the current presentation, we shall analyze how this framework can be embedded in an object-oriented modeling environment, i.e., how we can formulate model descriptions containing discontinuities in such a way that the model compiler can generate from that description simulation code that can be executed in a robust and efficient manner.

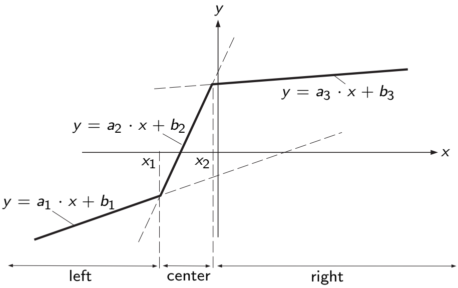
Event Descriptions of Discontinuous Functions II

Let us discuss the function:



Event Descriptions of Discontinuous Functions II

Let us discuss the function:

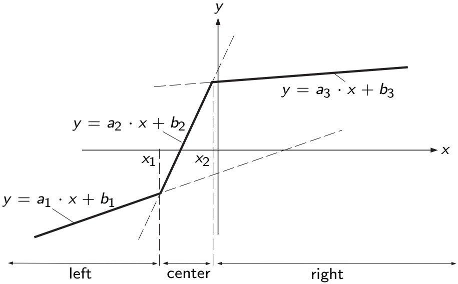


Functional description:

if $x < x_1$ **then** $y = a_1 \cdot x + b_1$
else if $x < x_2$ **then** $y = a_2 \cdot x + b_2$
else $y = a_3 \cdot x + b_3$;

Event Descriptions of Discontinuous Functions II

Let us discuss the function:



Functional description:

if $x < x_1$ **then** $y = a_1 \cdot x + b_1$
else if $x < x_2$ **then** $y = a_2 \cdot x + b_2$
else $y = a_3 \cdot x + b_3$;

Event description:

case *region*
left : $y = a_1 \cdot x + b_1$;
schedule *Center* **when** $x - x_1 == 0$;
center : $y = a_2 \cdot x + b_2$;
schedule *Left* **when** $x - x_1 == 0$;
schedule *Right* **when** $x - x_2 == 0$;
right : $y = a_3 \cdot x + b_3$;
schedule *Center* **when** $x - x_2 == 0$;
end;

event *Left*
region := *left*;
end *Left*;

event *Center*
region := *center*;
end *Center*;

event *Right*
region := *right*;
end *Right*;

Event Descriptions of Discontinuous Functions III

- The *functional description* is very compact, but if the model is being simulated in this form, the simulation will include the discontinuities, and we shall need to rely on the step-size control algorithm to detect and isolate these discontinuities.

Event Descriptions of Discontinuous Functions III

- The *functional description* is very compact, but if the model is being simulated in this form, the simulation will include the discontinuities, and we shall need to rely on the step-size control algorithm to detect and isolate these discontinuities.
- The *event description* is safe from a numerical point of view; it does not include discontinuities within the model equations; yet it is not compact, and it is anything but object oriented.

Event Descriptions of Discontinuous Functions III

- The *functional description* is very compact, but if the model is being simulated in this form, the simulation will include the discontinuities, and we shall need to rely on the step-size control algorithm to detect and isolate these discontinuities.

- The *event description* is safe from a numerical point of view; it does not include discontinuities within the model equations; yet it is not compact, and it is anything but object oriented.

- Furthermore, the event description, as presented, is not even complete. The variable *region*, which changes its value only at event times, is a *discrete state variable* that needs to be initialized. Somewhere in the section containing the *initial equations* we'll need a statement:

if $x < x_1$ **then** *region* := *left*;
  else if $x < x_2$ **then** *region* := *center*;
  else *region* := *right*;

Event Descriptions of Discontinuous Functions IV

Can the augmented event description be correctly simulated in all situations?

Event Descriptions of Discontinuous Functions IV

Can the augmented event description be correctly simulated in all situations?

- Let us assume that:

$$x(t) = \frac{x_2 - x_1}{2} \cdot \sin(t) + \frac{x_1 + x_2}{2}$$

Event Descriptions of Discontinuous Functions IV

Can the augmented event description be correctly simulated in all situations?

- Let us assume that:

$$x(t) = \frac{x_2 - x_1}{2} \cdot \sin(t) + \frac{x_1 + x_2}{2}$$

- $x(t)$ stays thus always in the *center* region. However, it may happen that $x = x_2$ exactly at the end of a step. In that case, the *Right* event gets scheduled, and the region switches to *right*.

Event Descriptions of Discontinuous Functions IV

Can the augmented event description be correctly simulated in all situations?

- Let us assume that:

$$x(t) = \frac{x_2 - x_1}{2} \cdot \sin(t) + \frac{x_1 + x_2}{2}$$

- $x(t)$ stays thus always in the *center* region. However, it may happen that $x = x_2$ exactly at the end of a step. In that case, the *Right* event gets scheduled, and the region switches to *right*.
- x becomes immediately smaller than x_2 again, but as the value x_2 is not reached a second time, the *Center* event doesn't get scheduled, and the model remains in the wrong region.

Event Descriptions of Discontinuous Functions IV

Can the augmented event description be correctly simulated in all situations?

- Let us assume that:

$$x(t) = \frac{x_2 - x_1}{2} \cdot \sin(t) + \frac{x_1 + x_2}{2}$$

- $x(t)$ stays thus always in the *center* region. However, it may happen that $x = x_2$ exactly at the end of a step. In that case, the *Right* event gets scheduled, and the region switches to *right*.
- x becomes immediately smaller than x_2 again, but as the value x_2 is not reached a second time, the *Center* event doesn't get scheduled, and the model remains in the wrong region.
- To avoid this problem, we need to build a hysteresis around each threshold and schedule two events, each time we pass through a threshold: an *arrival event*, and a *departure event*.

Event Descriptions of Discontinuous Functions IV

Can the augmented event description be correctly simulated in all situations?

- Let us assume that:

$$x(t) = \frac{x_2 - x_1}{2} \cdot \sin(t) + \frac{x_1 + x_2}{2}$$

- $x(t)$ stays thus always in the *center* region. However, it may happen that $x = x_2$ exactly at the end of a step. In that case, the *Right* event gets scheduled, and the region switches to *right*.
- x becomes immediately smaller than x_2 again, but as the value x_2 is not reached a second time, the *Center* event doesn't get scheduled, and the model remains in the wrong region.
- To avoid this problem, we need to build a hysteresis around each threshold and schedule two events, each time we pass through a threshold: an *arrival event*, and a *departure event*.
- This is how **Dymola** tackles this problem.

Event Descriptions of Discontinuous Functions V

Does the hysteretic event description cover all cases?

Event Descriptions of Discontinuous Functions V

Does the hysteretic event description cover all cases?

- Unfortunately, this is still not the case.

Event Descriptions of Discontinuous Functions V

Does the hysteretic event description cover all cases?

- Unfortunately, this is still not the case.
- It can happen that one event triggers immediately a second event in another discontinuity, which in turn triggers immediately another event back at the original discontinuity.

Event Descriptions of Discontinuous Functions V

Does the hysteretic event description cover all cases?

- Unfortunately, this is still not the case.
- It can happen that one event triggers immediately a second event in another discontinuity, which in turn triggers immediately another event back at the original discontinuity.
- Thus, we may be confronted with *algebraic loops* formed by chains of simultaneous events.

Event Descriptions of Discontinuous Functions V

Does the hysteretic event description cover all cases?

- Unfortunately, this is still not the case.
- It can happen that one event triggers immediately a second event in another discontinuity, which in turn triggers immediately another event back at the original discontinuity.
- Thus, we may be confronted with *algebraic loops* formed by chains of simultaneous events.
- For this reason, we need to *iterate after each event* to ensure that we once again have a *consistent set of initial conditions* for the subsequent continuous simulation segment.

Event Descriptions of Discontinuous Functions V

Does the hysteretic event description cover all cases?

- Unfortunately, this is still not the case.
- It can happen that one event triggers immediately a second event in another discontinuity, which in turn triggers immediately another event back at the original discontinuity.
- Thus, we may be confronted with *algebraic loops* formed by chains of simultaneous events.
- For this reason, we need to *iterate after each event* to ensure that we once again have a *consistent set of initial conditions* for the subsequent continuous simulation segment.

It becomes evident that manual coding of discontinuous models by means of event descriptions is a hopeless undertaking in all but the most trivial cases. We definitely need something better.

Object-oriented Descriptions of Discontinuities

What is wrong with the compact and convenient functional description of the discontinuous function formulated originally?

Object-oriented Descriptions of Discontinuities

What is wrong with the compact and convenient functional description of the discontinuous function formulated originally?

- The *functional description* contains the complete information of what needs to happen.

Object-oriented Descriptions of Discontinuities

What is wrong with the compact and convenient functional description of the discontinuous function formulated originally?

- The *functional description* contains the complete information of what needs to happen.
- The only problem with this description is that it cannot be safely simulated.

Object-oriented Descriptions of Discontinuities

What is wrong with the compact and convenient functional description of the discontinuous function formulated originally?

- The *functional description* contains the complete information of what needs to happen.
- The only problem with this description is that it cannot be safely simulated.
- However, since the description contains the complete information, what prevents us from formulating the model in this fashion and leave it up to the *model compiler* to decompose the functional description into a complete and consistent event description?

Object-oriented Descriptions of Discontinuities

What is wrong with the compact and convenient functional description of the discontinuous function formulated originally?

- The *functional description* contains the complete information of what needs to happen.
- The only problem with this description is that it cannot be safely simulated.
- However, since the description contains the complete information, what prevents us from formulating the model in this fashion and leave it up to the *model compiler* to decompose the functional description into a complete and consistent event description?
- This is precisely what **Dymola** does.

Object-oriented Descriptions of Discontinuities II

- We had already seen in the last few presentations that the **Dymola** model compiler performs a lot of *symbolic preprocessing*, before it generates the code that is to be numerically simulated.

Object-oriented Descriptions of Discontinuities II

- We had already seen in the last few presentations that the **Dymola** model compiler performs a lot of *symbolic preprocessing*, before it generates the code that is to be numerically simulated.

- For example, it performs symbolic index reduction by implementing the Pantelides algorithm.

Object-oriented Descriptions of Discontinuities II

- We had already seen in the last few presentations that the **Dymola** model compiler performs a lot of *symbolic preprocessing*, before it generates the code that is to be numerically simulated.
- For example, it performs symbolic index reduction by implementing the Pantelides algorithm.
- It also tackles algebraic loops by automatically placing a Newton iteration around each algebraic loop.

Object-oriented Descriptions of Discontinuities II

- We had already seen in the last few presentations that the **Dymola** model compiler performs a lot of *symbolic preprocessing*, before it generates the code that is to be numerically simulated.
- For example, it performs symbolic index reduction by implementing the Pantelides algorithm.
- It also tackles algebraic loops by automatically placing a Newton iteration around each algebraic loop.
- In some cases, the model compiler even generates multiple sets of simulation models with different state variables together with code to automatically toggle between them to avoid dynamic singularities (divisions by zero) in the model.

Object-oriented Descriptions of Discontinuities II

- We had already seen in the last few presentations that the **Dymola** model compiler performs a lot of *symbolic preprocessing*, before it generates the code that is to be numerically simulated.
- For example, it performs symbolic index reduction by implementing the Pantelides algorithm.
- It also tackles algebraic loops by automatically placing a Newton iteration around each algebraic loop.
- In some cases, the model compiler even generates multiple sets of simulation models with different state variables together with code to automatically toggle between them to avoid dynamic singularities (divisions by zero) in the model.
- We now realize that the model compiler does even considerably more work. It takes arbitrary object-oriented descriptions of discontinuous models and automatically decomposes them into series of event descriptions that can be safely and robustly simulated.

Object-oriented Descriptions of Discontinuities III

In **Dymola**, we code the discontinuous function using the following functional description:

$$
\begin{aligned}
y = \;& \textbf{if } x < x_1 \textbf{ then } a_1 \cdot x + b_1 \\
& \textbf{else if } x < x_2 \textbf{ then } a_2 \cdot x + b_2 \\
& \textbf{else } a_3 \cdot x + b_3;
\end{aligned}
$$

Object-oriented Descriptions of Discontinuities III

In **Dymola**, we code the discontinuous function using the following functional description:

$$
\begin{aligned}
y = \textbf{if } & x < x_1 \textbf{ then } a_1 \cdot x + b_1 \\
& \textbf{else if } x < x_2 \textbf{ then } a_2 \cdot x + b_2 \\
& \textbf{else } a_3 \cdot x + b_3;
\end{aligned}
$$

- The Dymola description is slightly different from the functional description proposed earlier.

Object-oriented Descriptions of Discontinuities III

In **Dymola**, we code the discontinuous function using the following functional description:

$$
\begin{aligned}
y = \textbf{if } & x < x_1 \textbf{ then } a_1 \cdot x + b_1 \\
& \textbf{else if } x < x_2 \textbf{ then } a_2 \cdot x + b_2 \\
& \textbf{else } a_3 \cdot x + b_3;
\end{aligned}
$$

- The Dymola description is slightly different from the functional description proposed earlier.
- Here, the dependent variable, y, is taken out of the **if**-clause, i.e., it applies to all branches of the **if**-clause.

Object-oriented Descriptions of Discontinuities III

In **Dymola**, we code the discontinuous function using the following functional description:

$$\begin{aligned} y = \textbf{if } & x < x_1 \textbf{ then } a_1 \cdot x + b_1 \\ & \textbf{else if } x < x_2 \textbf{ then } a_2 \cdot x + b_2 \\ & \textbf{else } a_3 \cdot x + b_3; \end{aligned}$$

- The Dymola description is slightly different from the functional description proposed earlier.
- Here, the dependent variable, y, is taken out of the **if**-clause, i.e., it applies to all branches of the **if**-clause.
- This is necessary, because otherwise, Dymola cannot *vertically sort* the **if**-statement together with the other model equations.

Object-oriented Descriptions of Discontinuities IV

Is the causality of the if-statement fixed?

Object-oriented Descriptions of Discontinuities IV

Is the causality of the if-statement fixed?

- Does the **if**-statement always compute the variable y, or can this statement also be solved for x?

Object-oriented Descriptions of Discontinuities IV

Is the causality of the if-statement fixed?

- Does the **if**-statement always compute the variable y, or can this statement also be solved for x?

- To answer this question, we must understand how the model compiler deals with this statement.

Object-oriented Descriptions of Discontinuities IV

Is the causality of the if-statement fixed?

- Does the **if**-statement always compute the variable y, or can this statement also be solved for x?
- To answer this question, we must understand how the model compiler deals with this statement.
- We introduce three integer variables, m_l, m_c, and m_r, whose values are linked to the linguistic discrete state variable, *region*, in the following way:

region	m_l	m_c	m_r
left	1	0	0
center	0	1	0
right	0	0	1

Object-oriented Descriptions of Discontinuities V

We can now reformulate the discontinuous function as follows:

Object-oriented Descriptions of Discontinuities V

We can now reformulate the discontinuous function as follows:

$y = m_l \cdot (a_1 \cdot x + b_1) + m_c \cdot (a_2 \cdot x + b_2) + m_r \cdot (a_3 \cdot x + b_3);$
case *region*
  left : **schedule** *Center* **when** $x - x_1 == 0$;
  center : **schedule** *Left* **when** $x - x_1 == 0$;
  schedule *Right* **when** $x - x_2 == 0$;
  right : **schedule** *Center* **when** $x - x_2 == 0$;
end;

Object-oriented Descriptions of Discontinuities V

We can now reformulate the discontinuous function as follows:

$y = m_l \cdot (a_1 \cdot x + b_1) + m_c \cdot (a_2 \cdot x + b_2) + m_r \cdot (a_3 \cdot x + b_3);$
case *region*
  left : **schedule** *Center* **when** $x - x_1 == 0;$
  center : **schedule** *Left* **when** $x - x_1 == 0;$
    schedule *Right* **when** $x - x_2 == 0;$
  right : **schedule** *Center* **when** $x - x_2 == 0;$
end;

together with the three discrete event descriptions:

event *Left*
  region := *left*;
  $m_l = 1; \quad m_c = 0; \quad m_r = 0;$
end *Left*;

event *Center*
  region := *center*;
  $m_l = 0; \quad m_c = 1; \quad m_r = 0;$
end *Center*;

event *Right*
  region := *right*;
  $m_l = 0; \quad m_c = 0; \quad m_r = 1;$
end *Right*;

Object-oriented Descriptions of Discontinuities VI

In this way, the former **if**-statement has been converted to the algebraic statement:

$$y = m_l \cdot (a_1 \cdot x + b_1) + m_c \cdot (a_2 \cdot x + b_2) + m_r \cdot (a_3 \cdot x + b_3)$$

Object-oriented Descriptions of Discontinuities VI

In this way, the former **if**-statement has been converted to the algebraic statement:

$$y = m_l \cdot (a_1 \cdot x + b_1) + m_c \cdot (a_2 \cdot x + b_2) + m_r \cdot (a_3 \cdot x + b_3)$$

which can be turned around in the usual way:

$$x = \frac{y - m_l \cdot b_1 - m_c \cdot b_2 - m_r \cdot b_3}{m_l \cdot a_1 - m_c \cdot a_2 - m_r \cdot a_3}$$

Object-oriented Descriptions of Discontinuities VI

In this way, the former **if**-statement has been converted to the algebraic statement:

$$y = m_l \cdot (a_1 \cdot x + b_1) + m_c \cdot (a_2 \cdot x + b_2) + m_r \cdot (a_3 \cdot x + b_3)$$

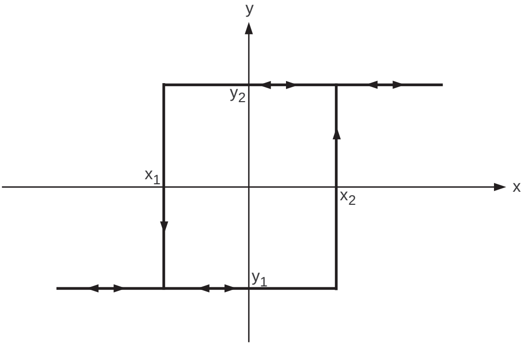
which can be turned around in the usual way:

$$x = \frac{y - m_l \cdot b_1 - m_c \cdot b_2 - m_r \cdot b_3}{m_l \cdot a_1 - m_c \cdot a_2 - m_r \cdot a_3}$$

Consequently, **if**-statements can also be *horizontally sorted*, just like other model equations.

Multi-valued Functions

The **if**-statements that we have introduced so far don't allow the description of multi-valued functions, such as the dry hysteresis function shown below:



Multi-valued Functions II

A possible event description for the dry hysteresis function could look as follows:

$y = y_{\text{last}};$
case *region*
  up : **schedule** *Down* **when** $x - x_1 < 0;$
  down : **schedule** *Up* **when** $x - x_2 > 0;$
end;

together with the two discrete event descriptions:

event *Up*
  region := *up*;
  $y_{\text{last}} := y_2;$
end *Left*;

event *Down*
  region := *down*;
  $y_{\text{last}} := y_1;$
end *Center*;

Multi-valued Functions II

A possible event description for the dry hysteresis function could look as follows:

$y = y_{\text{last}};$
case *region*
  up : **schedule** *Down* **when** $x - x_1 < 0$;
  down : **schedule** *Up* **when** $x - x_2 > 0$;
end;

together with the two discrete event descriptions:

event *Up*
  region := *up*;
  $y_{\text{last}} := y_2$;
end *Left*;

event *Down*
  region := *down*;
  $y_{\text{last}} := y_1$;
end *Center*;

In this code, y_{last} is a *discrete state variable* that needs to be initialized.

if and when

Since the **if**-statement cannot describe a multi-valued function, **Dymola** offers also a **when**-statement that can be used for such purpose.

if and when

Since the **if**-statement cannot describe a multi-valued function, **Dymola** offers also a **when**-statement that can be used for such purpose.

The semantics of the **if**-statement is as follows:

$$y = \textbf{if } x > 0 \textbf{ then } \ldots$$

means: *if x is larger than zero, then . . .* .

if and when

Since the **if**-statement cannot describe a multi-valued function, **Dymola** offers also a **when**-statement that can be used for such purpose.

The semantics of the **if**-statement is as follows:

$$y = \textbf{if } x > 0 \textbf{ then } \ldots$$

means: *if x is larger than zero, then . . .* .

In contrast, the semantics of the **when**-statement is as follows:

$$\textbf{when } x > 0 \ \ldots$$

means: *when x becomes larger than zero, then . . .* , or in other words, *when x crosses zero in the positive direction, then . . .* .

if and when II

Compare also:

$y =$ **if** $x == 0$ **then** ...

which means: *if x is exactly equal to zero, then ...* (not a very meaningful condition for a real-valued variable x).

if and when II

Compare also:

$y =$ **if** $x == 0$ **then** ...

which means: *if x is exactly equal to zero, then ...* (not a very meaningful condition for a real-valued variable x).

In contrast:

when $x == 0$ ...

means: *when x becomes equal to zero, then ...*, or in other words, *when x crosses zero in either direction, then ...* (very meaningful and frequently used).

if and when III

We might thus be inclined to code the *dry hysteresis function* in the following way:

```
when x < x1
   y = y1;
end when;

when x > x2
   y = y2;
end when;
```

if and when III

We might thus be inclined to code the *dry hysteresis function* in the following way:

when $x < x_1$
  $y = y_1$;
end when;

when $x > x_2$
  $y = y_2$;
end when;

Unfortunately, this won't work, because **Dymola** doesn't check that the conditions of all **when**-statements are mutually exclusive. The Dymola model comiler associates each equation inside a **when**-statement with its condition, and *sorts all of these equations both vertically and horizontally* together with all other model equations.

if and when III

We might thus be inclined to code the *dry hysteresis function* in the following way:

```
when x < x1
   y = y1;
end when;

when x > x2
   y = y2;
end when;
```

Unfortunately, this won't work, because **Dymola** doesn't check that the conditions of all **when**-statements are mutually exclusive. The Dymola model comiler associates each equation inside a **when**-statement with its condition, and *sorts all of these equations both vertically and horizontally* together with all other model equations.

Consequently, we cannot specify two separate equations to compute the variable y.

if and when IV

One way to avoid this pitfall would be to code:

when $x < x_1$ **or** $x > x_2$
  $y =$ **if** $x < 0$ **then** y_1 **else** y_2;
end when;

which will work fine, except that y is still a *discrete state variable* that must be initialized in the *initial equation* section of the model.

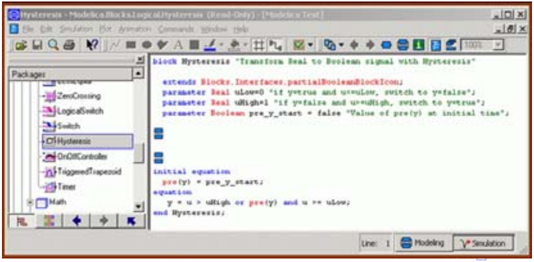
if and when IV

One way to avoid this pitfall would be to code:

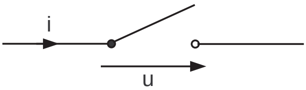
when $x < x_1$ **or** $x > x_2$
  $y =$ **if** $x < 0$ **then** y_1 **else** y_2;
end when;

which will work fine, except that y is still a *discrete state variable* that must be initialized in the *initial equation* section of the model.

The current version of the **Modelica Standard Library** codes this particular function even without use of a **when**-statement using a simple *Boolean expression*:

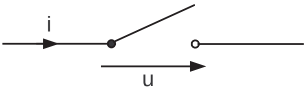


The Switch Equation



- The electrical switch is characterized by two variables, the voltage u and the current i.

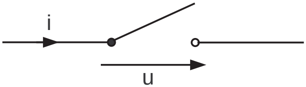
The Switch Equation



- The electrical switch is characterized by two variables, the voltage u and the current i.

- We already know that the equations that we obtain from an object-oriented description of physical systems are initially *acausal*.

The Switch Equation



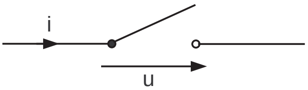
- The electrical switch is characterized by two variables, the voltage u and the current i.

- We already know that the equations that we obtain from an object-oriented description of physical systems are initially *acausal*.

- In **Dymola**, the electrical switch can be modeled using the following implicit equation:

$$0 = \textbf{if } switch == open \textbf{ then } i \textbf{ else } u;$$

The Switch Equation



- The electrical switch is characterized by two variables, the voltage u and the current i.
- We already know that the equations that we obtain from an object-oriented description of physical systems are initially *acausal*.
- In **Dymola**, the electrical switch can be modeled using the following implicit equation:

$$0 = \textbf{ if } switch == open \textbf{ then } i \textbf{ else } u;$$

- The electrical switch can, however, also be described by an algebraic equation:

switch	m_o
open	1
closed	0

$$0 = m_o \cdot i + (1 - m_o) \cdot u$$

The Switch Equation II

For the *algebraic switch equation*:

$$0 = m_o \cdot i + (1 - m_o) \cdot u$$

there exist two possible causalizations:

$$i = \frac{m_o - 1}{m_o} \cdot u$$
$$u = \frac{m_o}{m_o - 1} \cdot i$$

The Switch Equation II

For the *algebraic switch equation*:

$$0 = m_o \cdot i + (1 - m_o) \cdot u$$

there exist two possible causalizations:

$$i = \frac{m_o - 1}{m_o} \cdot u$$
$$u = \frac{m_o}{m_o - 1} \cdot i$$

Unfortunately, both of them result in a *division by zero* in one of the two switch positions.

The Switch Equation II

For the *algebraic switch equation*:

$$0 = m_o \cdot i + (1 - m_o) \cdot u$$

there exist two possible causalizations:

$$i = \frac{m_o - 1}{m_o} \cdot u$$
$$u = \frac{m_o}{m_o - 1} \cdot i$$

Unfortunately, both of them result in a *division by zero* in one of the two switch positions.

The computational causality of the switch equation depends on the switch position.

The Switch Equation II

For the *algebraic switch equation*:

$$0 = m_o \cdot i + (1 - m_o) \cdot u$$

there exist two possible causalizations:

$$i = \frac{m_o - 1}{m_o} \cdot u$$
$$u = \frac{m_o}{m_o - 1} \cdot i$$

Unfortunately, both of them result in a *division by zero* in one of the two switch positions.

The computational causality of the switch equation depends on the switch position.

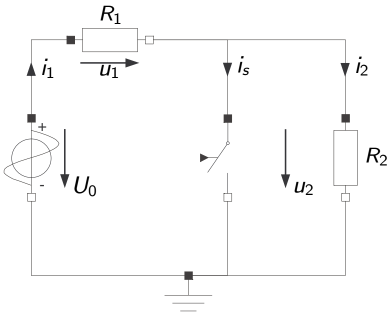
The only way to get a *free computational causality* is to include the switch equation inside an *algebraic loop*.

The Switch Equation III

Let us start with an example. We shall simulate a simple electrical circuit containing a switch.

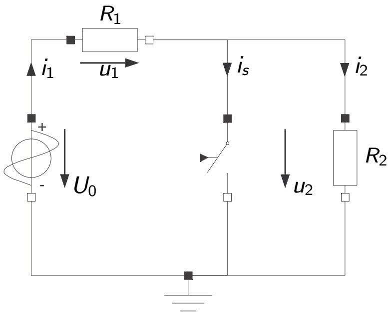
The Switch Equation III

Let us start with an example. We shall simulate a simple electrical circuit containing a switch.



The Switch Equation III

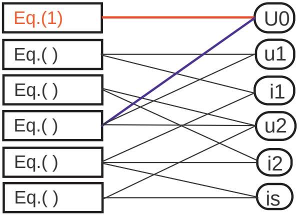
Let us start with an example. We shall simulate a simple electrical circuit containing a switch.



$$\begin{aligned}
U_0 &= f(t) \\
u_1 &= R_1 \cdot i_1 \\
u_2 &= R_2 \cdot i_2 \\
U_0 &= u_1 + u_2 \\
i_1 &= i_s + i_2 \\
0 &= m_o \cdot i_s + (1 - m_o) \cdot u_2
\end{aligned}$$

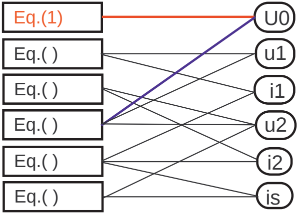
The Switch Equation IV

$$U_0 = f(t)$$
$$u_1 = R_1 \cdot i_1$$
$$u_2 = R_2 \cdot i_2$$
$$U_0 = u_1 + u_2$$
$$i_1 = i_s + i_2$$
$$0 = m_o \cdot i_s + (1 - m_o) \cdot u_2$$

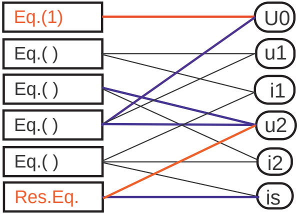


The Switch Equation IV

$$
\begin{aligned}
U_0 &= f(t) \\
u_1 &= R_1 \cdot i_1 \\
u_2 &= R_2 \cdot i_2 \\
U_0 &= u_1 + u_2 \\
i_1 &= i_s + i_2 \\
0 &= m_o \cdot i_s + (1 - m_o) \cdot u_2
\end{aligned}
$$

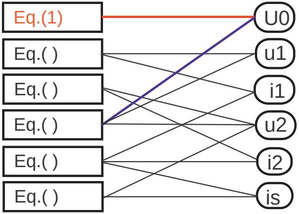


All *switch equations* must be included in the list of the *residual equations*.

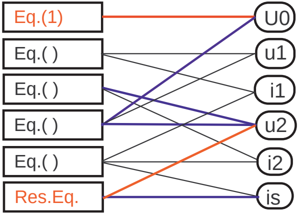


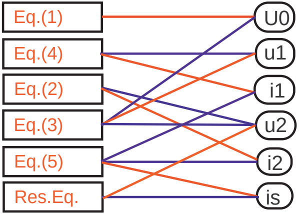
The Switch Equation IV

$$U_0 = f(t)$$
$$u_1 = R_1 \cdot i_1$$
$$u_2 = R_2 \cdot i_2$$
$$U_0 = u_1 + u_2$$
$$i_1 = i_s + i_2$$
$$0 = m_o \cdot i_s + (1 - m_o) \cdot u_2$$

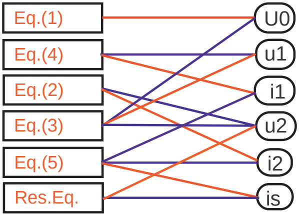


All *switch equations* must be included in the list of the *residual equations*.





The Switch Equation V



$$U_0 = f(t)$$

$$i_2 = \frac{1}{R_2} \cdot u_2$$

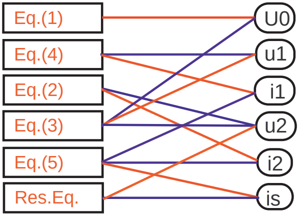
$$u_1 = U_0 - u_2$$

$$i_1 = \frac{1}{R_1} \cdot u_1$$

$$i_s = i_1 - i_2$$

$$u_2 = \frac{m_o}{m_o - 1} \cdot i_s$$

The Switch Equation V



$$U_0 = f(t)$$
$$i_2 = \frac{1}{R_2} \cdot u_2$$
$$u_1 = U_0 - u_2$$
$$i_1 = \frac{1}{R_1} \cdot u_1$$
$$i_s = i_1 - i_2$$
$$u_2 = \frac{m_o}{m_o - 1} \cdot i_s$$

We use substitution:

$$
\begin{aligned}
u_2 &= \frac{m_o}{m_o - 1} \cdot i_s \\
&= \frac{m_o}{m_o - 1} \cdot (i_1 - i_2) \\
&= \frac{m_o}{(m_o - 1) \cdot R_1} \cdot u_1 - \frac{m_o}{(m_o - 1) \cdot R_2} \cdot u_2 \\
&= \frac{m_o}{(m_o - 1) \cdot R_1} \cdot U_0 - \frac{m_o}{(m_o - 1) \cdot R_1} \cdot u_2 - \frac{m_o}{(m_o - 1) \cdot R_2} \cdot u_2 \\
&= \frac{m_o}{(m_o - 1) \cdot R_1} \cdot U_0 - \frac{m_o \cdot (R_1 + R_2)}{(m_o - 1) \cdot R_1 \cdot R_2} \cdot u_2
\end{aligned}
$$

The Switch Equation VI

Solving for u_2, we obtain:

$$u_2 = \frac{m_o \cdot R_2}{m_o \cdot (R_1 + R_2) + (m_o - 1) \cdot R_1 \cdot R_2} \cdot U_0$$

The Switch Equation VI

Solving for u_2, we obtain:

$$u_2 = \frac{m_o \cdot R_2}{m_o \cdot (R_1 + R_2) + (m_o - 1) \cdot R_1 \cdot R_2} \cdot U_0$$

This equation doesn't lead to a division by zero in either of the two switch positions.

The Switch Equation VI

Solving for u_2, we obtain:

$$u_2 = \frac{m_o \cdot R_2}{m_o \cdot (R_1 + R_2) + (m_o - 1) \cdot R_1 \cdot R_2} \cdot U_0$$

This equation doesn't lead to a division by zero in either of the two switch positions.

The model equations can thus be written in the following form:

$$\begin{aligned}
U_0 &= f(t) \\
u_2 &= \frac{m_o \cdot R_2}{m_o \cdot (R_1 + R_2) + (m_o - 1) \cdot R_1 \cdot R_2} \cdot U_0 \\
i_2 &= \frac{1}{R_2} \cdot u_2 \\
u_1 &= U_0 - u_2 \\
i_1 &= \frac{1}{R_1} \cdot u_1 \\
i_s &= i_1 - i_2
\end{aligned}$$

The Switch Equation VI

Solving for u_2, we obtain:

$$u_2 = \frac{m_o \cdot R_2}{m_o \cdot (R_1 + R_2) + (m_o - 1) \cdot R_1 \cdot R_2} \cdot U_0$$

This equation doesn't lead to a division by zero in either of the two switch positions.

The model equations can thus be written in the following form:

$$\begin{aligned}
U_0 &= f(t) \\
u_2 &= \frac{m_o \cdot R_2}{m_o \cdot (R_1 + R_2) + (m_o - 1) \cdot R_1 \cdot R_2} \cdot U_0 \\
i_2 &= \frac{1}{R_2} \cdot u_2 \\
u_1 &= U_0 - u_2 \\
i_1 &= \frac{1}{R_1} \cdot u_1 \\
i_s &= i_1 - i_2
\end{aligned}$$

These model equations don't contain either an algebraic loop or a division by zero. Thus, they can be simulated without any problems.

Ideal Diodes

Ideal diodes are ideal electrical switches complemented by an internal logic for determining the switch position.

Ideal Diodes

Ideal diodes are ideal electrical switches complemented by an internal logic for determining the switch position.

An ideal diode closes its switch, when the voltage across the diode from the anode to the cathode becomes positive, and it opens its switch again, when the current through the diode passes through zero, if at that time the voltage across the diode is negative.

Ideal Diodes

Ideal diodes are ideal electrical switches complemented by an internal logic for determining the switch position.

An ideal diode closes its switch, when the voltage across the diode from the anode to the cathode becomes positive, and it opens its switch again, when the current through the diode passes through zero, if at that time the voltage across the diode is negative.

An ideal diode can be modeled in **Dymola** as follows:

$$0 = \textbf{if } OpenSwitch \textbf{ then } i_d \textbf{ else } u_d;$$
$$OpenSwitch = u_d <= 0 \textbf{ and not } i_d > 0;$$

Ideal Diodes

Ideal diodes are ideal electrical switches complemented by an internal logic for determining the switch position.

An ideal diode closes its switch, when the voltage across the diode from the anode to the cathode becomes positive, and it opens its switch again, when the current through the diode passes through zero, if at that time the voltage across the diode is negative.

An ideal diode can be modeled in **Dymola** as follows:

$$0 = \textbf{if } \textit{OpenSwitch} \textbf{ then } i_d \textbf{ else } u_d;$$
$$\textit{OpenSwitch} = u_d <= 0 \textbf{ and not } i_d > 0;$$

OpenSwitch is here a Boolean variable, the value of which is computed in the above Boolean expression. If *OpenSwitch* is *true*, the switch is considered *open*.

Ideal Diodes II

Unfortunately, the model:

$0 = \textbf{if } \mathit{OpenSwitch} \textbf{ then } i_d \textbf{ else } u_d;$
$\mathit{OpenSwitch} = u_d <= 0 \textbf{ and not } i_d > 0;$

while being very elegant, is problematic from a numerical point of view.

Ideal Diodes II

Unfortunately, the model:

$$0 = \textbf{if } \mathit{OpenSwitch} \textbf{ then } i_d \textbf{ else } u_d;$$
$$\mathit{OpenSwitch} = u_d <= 0 \textbf{ and not } i_d > 0;$$

while being very elegant, is problematic from a numerical point of view.

Remember that **if**-statements get translated into *event descriptions* by the model compiler. In the process, the conditional expression gets converted to a *zero-crossing function*.

Ideal Diodes II

Unfortunately, the model:

$0 = \textbf{if } \mathit{OpenSwitch} \textbf{ then } i_d \textbf{ else } u_d;$
$\mathit{OpenSwitch} = u_d <= 0 \textbf{ and not } i_d > 0;$

while being very elegant, is problematic from a numerical point of view.

Remember that **if**-statements get translated into *event descriptions* by the model compiler. In the process, the conditional expression gets converted to a *zero-crossing function*.

In the above example, we obtain the zero-crossing function:

$$f = \textbf{if } \mathit{OpenSwitch} \textbf{ then } 1 \textbf{ else } -1$$

Ideal Diodes III

- The function f indeed crosses through zero, whenever the switch changes its position, but it is anything but smooth. In fact, its derivative is zero everywhere except at the switching point itself, where it is infinite.

Ideal Diodes III

- The function f indeed crosses through zero, whenever the switch changes its position, but it is anything but smooth. In fact, its derivative is zero everywhere except at the switching point itself, where it is infinite.

- Hence we cannot use any higher-order iteration algorithm, such as *cubic interpolation*, to iterate on this zero-crossing function. In fact, the only one among the iteration methods introduced in the previous presentation that will work half-way efficiently on this zero-crossing function is *golden section*.

Ideal Diodes III

- The function f indeed crosses through zero, whenever the switch changes its position, but it is anything but smooth. In fact, its derivative is zero everywhere except at the switching point itself, where it is infinite.

- Hence we cannot use any higher-order iteration algorithm, such as *cubic interpolation*, to iterate on this zero-crossing function. In fact, the only one among the iteration methods introduced in the previous presentation that will work half-way efficiently on this zero-crossing function is *golden section*.

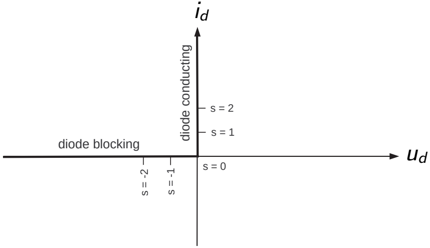
- We definitely need something better.

Parameterized Curve Descriptions

We parameterize the diode characteristic in a new variable s as shown below:

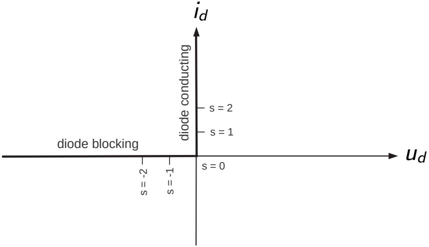
Parameterized Curve Descriptions

We parameterize the diode characteristic in a new variable s as shown below:



Parameterized Curve Descriptions

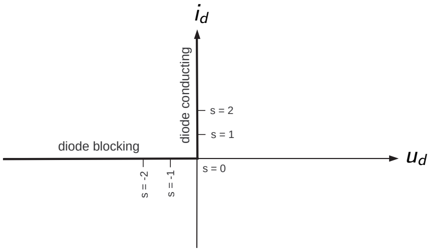
We parameterize the diode characteristic in a new variable s as shown below:



- In the blocking mode $s = u_d$.

Parameterized Curve Descriptions

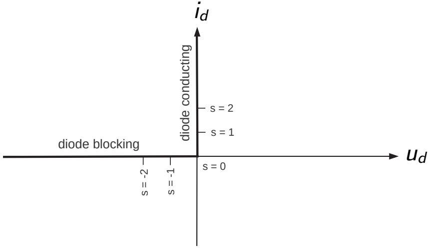
We parameterize the diode characteristic in a new variable s as shown below:



- In the blocking mode $s = u_d$.
- In the conducting mode $s = i_d$.

Parameterized Curve Descriptions

We parameterize the diode characteristic in a new variable s as shown below:



- In the blocking mode $s = u_d$.
- In the conducting mode $s = i_d$.

Thus, we can code the diode model as follows:

u_d = **if** *OpenSwitch* **then** s **else** 0;
i_d = **if** *OpenSwitch* **then** 0 **else** s;
OpenSwitch = $s < 0$;

Parameterized Curve Descriptions II

- The Dymola model compiler is smart enough to translate the Boolean expression to the zero-crossing function:

$$f = s$$

which is as smooth as smooth can be.

Parameterized Curve Descriptions II

- The Dymola model compiler is smart enough to translate the Boolean expression to the zero-crossing function:

$$f = s$$

  which is as smooth as smooth can be.

- Consequently, we can apply any one of the iteration methods introduced in the previous presentation to this model.

Parameterized Curve Descriptions II

- The Dymola model compiler is smart enough to translate the Boolean expression to the zero-crossing function:

  $$f = s$$

  which is as smooth as smooth can be.

- Consequently, we can apply any one of the iteration methods introduced in the previous presentation to this model.

An algebraic version of that model can be written as:

$$u_d = m_o \cdot s;$$
$$i_d = (1 - m_o) \cdot s;$$
$$m_o = \textbf{if } s < 0 \textbf{ then } 1 \textbf{ else } 0;$$

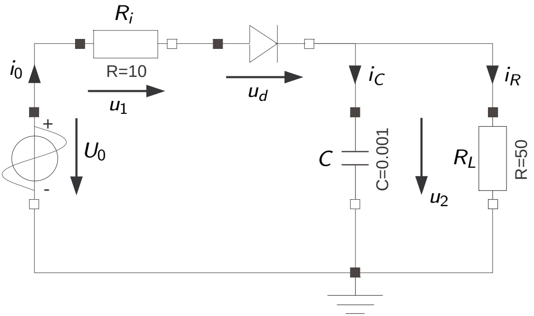
which is the version that we shall work with here, as these equations are easier to analyze.

Parameterized Curve Descriptions III

Let us illustrate the use of the ideal diode model by means of the simple half-way rectifier circuit shown below:

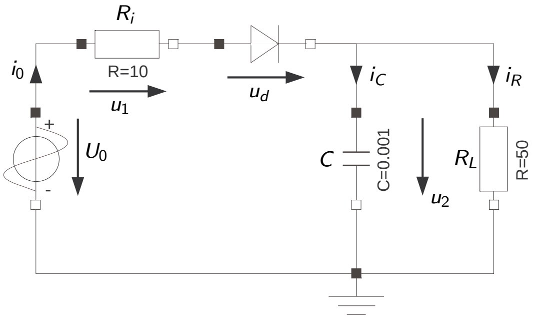
Parameterized Curve Descriptions III

Let us illustrate the use of the ideal diode model by means of the simple half-way rectifier circuit shown below:



Parameterized Curve Descriptions III

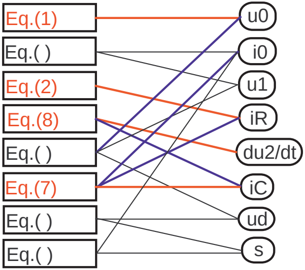
Let us illustrate the use of the ideal diode model by means of the simple half-way rectifier circuit shown below:



$$
\begin{aligned}
u_0 &= f(t) \\
u_1 &= R_i \cdot i_0 \\
u_2 &= R_L \cdot i_R \\
i_C &= C \cdot \frac{du_2}{dt} \\
u_0 &= u_1 + u_d + u_2 \\
i_0 &= i_C + i_R \\
u_d &= m_o \cdot s \\
i_0 &= (1 - m_o) \cdot s
\end{aligned}
$$

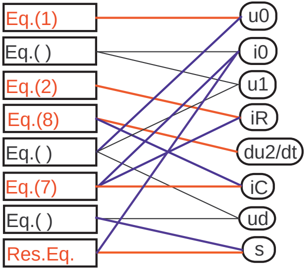
Parameterized Curve Descriptions IV

$$U_0 = f(t)$$
$$u_1 = R_i \cdot i_0$$
$$u_2 = R_L \cdot i_R$$
$$i_C = C \cdot \frac{du_2}{dt}$$
$$u_0 = u_1 + u_d + u_2$$
$$i_0 = i_C + i_R$$
$$u_d = m_o \cdot s$$
$$i_0 = (1 - m_o) \cdot s$$



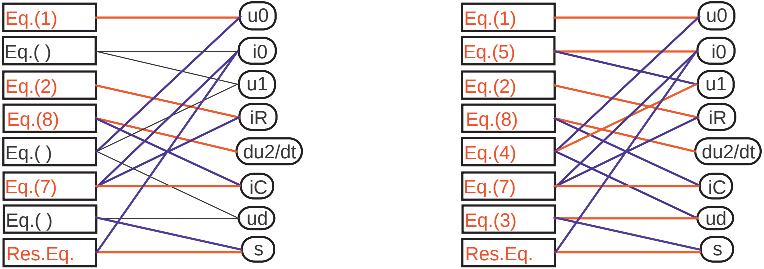
Parameterized Curve Descriptions V

In order to avoid divisions by zero, we need to choose *s* as our first *tearing variable*.

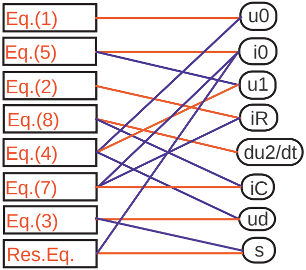


Parameterized Curve Descriptions V

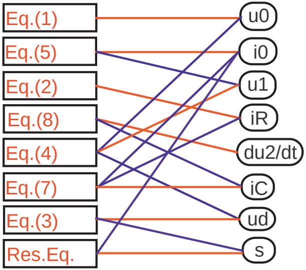
In order to avoid divisions by zero, we need to choose *s* as our first *tearing variable*.



Parameterized Curve Descriptions VI



Parameterized Curve Descriptions VI



$$U_0 = f(t)$$

$$i_R = \frac{1}{R_L} \cdot u_2$$

$$u_d = m_o \cdot s$$

$$u_1 = u_0 - u_d - u_2$$

$$i_0 = \frac{1}{R_i} \cdot u_1$$

$$s = \frac{1}{1 - m_o} \cdot i_0$$

$$i_C = i_0 - i_R$$

$$\frac{du_2}{dt} = \frac{1}{C} \cdot i_C$$

Parameterized Curve Descriptions VII

Substitution yields:

$$s = \frac{1}{m_o + (1 - m_o) \cdot R_i} \cdot (u_0 - u_2)$$

which does not lead to a division by zero in either switch position.

Parameterized Curve Descriptions VII

Substitution yields:

$$s = \frac{1}{m_o + (1 - m_o) \cdot R_i} \cdot (u_0 - u_2)$$

which does not lead to a division by zero in either switch position.

Thus the model equations can be written in the following form:

$$\begin{aligned}
u_0 &= f(t) \\
i_R &= \frac{1}{R_L} \cdot u_2 \\
s &= \frac{1}{m_o + (1 - m_o) \cdot R_i} \cdot (u_0 - u_2) \\
u_d &= m_o \cdot s \\
u_1 &= u_0 - u_d - u_2 \\
i_0 &= \frac{1}{R_i} \cdot u_1 \\
i_C &= i_0 - i_R \\
\frac{du_2}{dt} &= \frac{1}{C} \cdot i_C
\end{aligned}$$

Parameterized Curve Descriptions VIII

A single *zero-crossing function* accompanies the model equations:

$$f = s$$

with the associated *event action*:

event *Toggle*
$m_o := 1 - m_o$;
end *Toggle*;

Parameterized Curve Descriptions VIII

A single *zero-crossing function* accompanies the model equations:

$$f = s$$

with the associated *event action*:

```
event Toggle
    m_o := 1 - m_o;
end Toggle;
```

The correct initial value of the *discrete state variable*, m_o, is assigned to that variable in an appropriate *initialization section* of the simulation program.

Parameterized Curve Descriptions VIII

A single *zero-crossing function* accompanies the model equations:

$$f = s$$

with the associated *event action*:

```
event Toggle
    m_o := 1 − m_o;
end Toggle;
```

The correct initial value of the *discrete state variable*, m_o, is assigned to that variable in an appropriate *initialization section* of the simulation program.

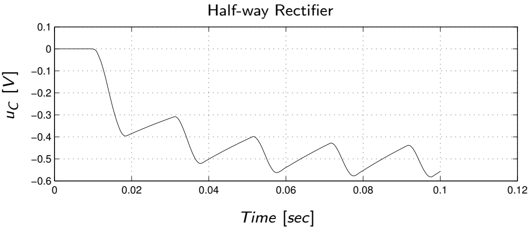
The model can now be simulated without any difficulties using any numerical integration algorithm with a root solver.

Parameterized Curve Descriptions IX

Simulation results:

Parameterized Curve Descriptions IX

Simulation results:



Conclusions

- In this presentation, we first discussed the *event description* of models containing discontinuities. We demonstrated that manually reducing a discontinuous model to a simulation code at the level of event descriptions accompanying model equations can be highly challenging and is, in fact, a hopeless undertaking except in the simplest of cases.

Conclusions

- In this presentation, we first discussed the *event description* of models containing discontinuities. We demonstrated that manually reducing a discontinuous model to a simulation code at the level of event descriptions accompanying model equations can be highly challenging and is, in fact, a hopeless undertaking except in the simplest of cases.

- We then showed how discontinuous models can be described in an object-oriented fashion, and how the model compiler can compile that *object-oriented description* down to an event description in an algorithmic and systematic fashion.

Conclusions

- In this presentation, we first discussed the *event description* of models containing discontinuities. We demonstrated that manually reducing a discontinuous model to a simulation code at the level of event descriptions accompanying model equations can be highly challenging and is, in fact, a hopeless undertaking except in the simplest of cases.

- We then showed how discontinuous models can be described in an object-oriented fashion, and how the model compiler can compile that *object-oriented description* down to an event description in an algorithmic and systematic fashion.

- We then looked at *multi-valued functions* and showed how these can be modeled.

Conclusions

- In this presentation, we first discussed the *event description* of models containing discontinuities. We demonstrated that manually reducing a discontinuous model to a simulation code at the level of event descriptions accompanying model equations can be highly challenging and is, in fact, a hopeless undertaking except in the simplest of cases.

- We then showed how discontinuous models can be described in an object-oriented fashion, and how the model compiler can compile that *object-oriented description* down to an event description in an algorithmic and systematic fashion.

- We then looked at *multi-valued functions* and showed how these can be modeled.

- The presentation ended with a description of the *switch equation*, i.e., an equation, the computational causality of which changes as a function of the switch position. *Ideal diodes* were discussed as an application of the switch equation, and we introduced the notion of *parameterized curve descriptions* as a means to obtain simulation code that is numerically better behaved during event iterations.

References

1. 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.

2. 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.

3. 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.

4. Shiva, Ali (2004), *Modeling Switching Networks Using Bond Graph Technique*, MS Thesis, Dept. of Aerospace & Mechanical Engr., University of Arizona, Tucson, AZ.