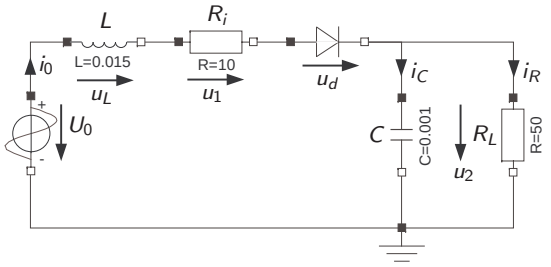
Numerical Simulation of Dynamic Systems XXIII

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

May 14, 2013

Variable Structure Models

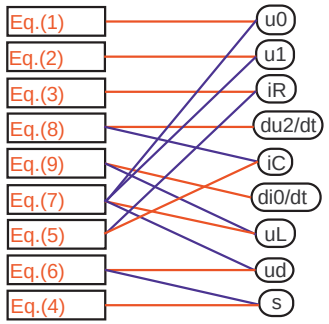
Let us try to simulate another almost identical circuit. All I did was to add an inductor in series with the source.



$$u_0 = f(t)$$
$$u_1 = R_i \cdot i_0$$
$$u_2 = R_L \cdot i_R$$
$$i_C = C \cdot \frac{du_2}{dt}$$
$$u_L = L \cdot \frac{di_0}{dt}$$
$$u_0 = u_L + u_1 + u_d + u_2$$
$$i_0 = i_C + i_R$$
$$u_d = m_o \cdot s$$
$$i_0 = (1 - m_o) \cdot s$$

Variable Structure Models II

This time, there is no algebraic loop. All equations can be causalized at once.



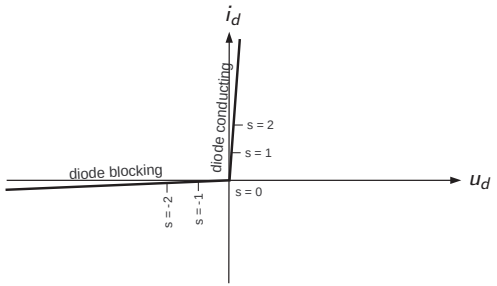
$$U_0 = f(t)$$
$$u_1 = R_i \cdot i_0$$
$$i_R = \frac{1}{R_L} \cdot u_2$$
$$s = \frac{1}{1 - m_o} \cdot i_0$$
$$i_C = i_0 - i_R$$
$$u_d = m_o \cdot s$$
$$u_L = u_0 - u_1 - u_d - u_2$$
$$\frac{du_2}{dt} = \frac{1}{C} \cdot i_C$$
$$\frac{di_0}{dt} = \frac{1}{L} \cdot u_L$$

Variable Structure Models III

- When we try to simulate this model, we get a division by zero as soon as the switch opens for the first time.
- The reason is simple: we failed to include the switch equation inside an algebraic loop. Thus, the causality on the switch was fixed, which invariably leads to a division by zero in one of the two switch positions.
- The cause of the problem is quite obvious. The inductor defines the inductive current as a state variable, i.e., i_0 is a known variable. Since this current flows also through the diode, we have no choice but to always compute the voltage across the diode, u_d, from the diode characteristic. Hence the causality of the diode is fixed, which foretells disaster.

Variable Structure Models IV

The **Modelica Standard Library** tackles this problem by replacing the *ideal diode* by a *leaky diode*:



Logical model equations:

$u_d = \textbf{if } \mathit{OpenSwitch} \textbf{ then } s \textbf{ else } R_0 \cdot s;$
$i_d = \textbf{if } \mathit{OpenSwitch} \textbf{ then } G_0 \cdot s \textbf{ else } s;$
$\mathit{OpenSwitch} = s < 0;$

Algebraic model equations:

$u_d = [m_o + (1 - m_o) \cdot R_0] \cdot s;$
$i_d = [m_o \cdot G_0 + (1 - m_o)] \cdot s;$
$m_o = \textbf{if } s < 0 \textbf{ then } 1 \textbf{ else } 0;$

Variable Structure Models V

The leaky diode model doesn't change the causalities of the equation system, i.e., the structure digraph of the model using the leaky diode is exactly the same as that using the ideal diode. However, the leaky diode avoids the division by zero.

The causal equations of the model using the leaky diode are:

$$
\begin{aligned}
u_0 &= f(t) \\
u_1 &= R_i \cdot i_0 \\
i_R &= \frac{1}{R_L} \cdot u_2 \\
s &= \frac{1}{m_o \cdot G_0 + (1 - m_o)} \cdot i_0 \\
i_C &= i_0 - i_R \\
u_d &= [m_o + (1 - m_o) \cdot R_0] \cdot s \\
u_L &= u_0 - u_1 - u_d - u_2 \\
\frac{du_2}{dt} &= \frac{1}{C} \cdot i_C \\
\frac{di_0}{dt} &= \frac{1}{L} \cdot u_L
\end{aligned}
$$

Variable Structure Models VI

- The *leaky diode approximation* is not unrealistic. In fact, the *real diode characteristic* resembles more a leaky diode than an ideal diode.
- Unfortunately, whenever the ideal diode simulation model dies with a division by zero, the leaky diode model exhibits a division by a very small fudge variable whose value is arbitrary. Thus, the model becomes stiff, and the smaller we make the fudge variables, R_0 and G_0, the stiffer the model will become.
- Furthermore, the simulation results often depend significantly on the value of these non-physical fudge parameters.
- For these reasons, we would prefer to use an *ideal diode model* if we could.

Variable Structure Models VII

What is so special about this model?

- When the switch is closed, i.e., while the diode is conducting, the model exhibits second-order dynamics. However, once the switch opens, i.e., while the diode blocks the current, we are faced with only first-order dynamics. The inductor does not contribute to the dynamics in that case.
- We call a model that exhibits different structural properties, such as a varying number of differential equations depending on the position of some switches, a *variable structure model*.
- Variable structure systems are very common, e.g. in mechanical engineering. All systems involving clutches are by their very nature variable structure systems. In electrical engineering, most switching power converters are variable structure systems.
- The way, the equations of our system were formulated, it doesn't look like these equations contain a *structural singularity*. There is no constraint equation to be found. The singularity looks to be *parametric* in nature, thus the *Pantelides algorithm* cannot solve it.

Mixed-mode Integration

Let us look a little more closely at the inductor equation:

$$u_L = L \cdot \frac{di_0}{dt}$$

The reason why the causality on this equation is fixed is because of our desire to use i_0 as a state variable.

Let us check what happens when we inline the inductor, e.g. using a BDF algorithm:

$$u_L = L \cdot di_0$$
$$i_0 = \mathrm{pre}(i_0) + \bar{h} \cdot di_0$$

Substituting the model equation into the solver equation, we obtain:

$$i_0 = \mathrm{pre}(i_0) + \frac{\bar{h}}{L} \cdot u_L$$

By now, the current, i_0, is algebraic and linear in the voltage, u_L, i.e., **the causality of the inlined inductor is free**.

Mixed-mode Integration II

A simulation scheme, in which different integrators use different algorithms, is called a *mixed-mode integration* scheme.

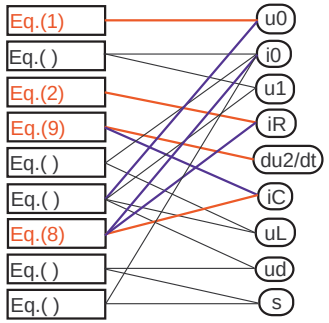
In the given example, we use a mixed-mode integration scheme by inlining the inductor, but not the capacitor.

The model equations can now be written as follows:

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

Mixed-mode Integration III

We start to causalize the equations:

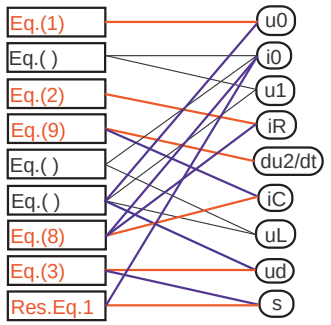


$$u_0 = f(t)$$
$$u_1 = R_i \cdot i_0$$
$$u_2 = R_L \cdot i_R$$
$$i_C = C \cdot \frac{du_2}{dt}$$
$$i_0 = \mathrm{pre}(i_0) + \frac{\bar{h}}{L} \cdot u_L$$
$$u_0 = u_L + u_1 + u_d + u_2$$
$$i_0 = i_C + i_R$$
$$u_d = m_o \cdot s$$
$$i_0 = (1 - m_o) \cdot s$$

We encounter an algebraic loop in five equations and five unknowns.

Mixed-mode Integration IV

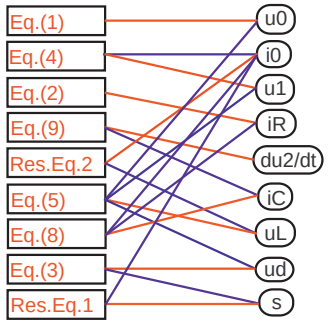
We must select s as our first tearing variable to avoid a division by zero:



Unfortunately, there is still a second embedded algebraic loop in three equations and three unknowns.

Mixed-mode Integration V

We choose a second residual equation and complete the causalization:



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

Mixed-mode Integration VI

Using the variable substitution technique, we find replacement equations for the two residual equations. The final set of horizontally and vertically sorted equations presents itself as follows:

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

Mixed-mode Integration VII

- This model can be simulated in both the open and closed switch positions without any division by zero.
- There is no stiffness problem in the model.
- The switch variable m_0 is still a discrete state variable that must be initialized in the initialization section of the model.

Mixed-mode Integration VIII

Can mixed-mode integration be used to simulate all variable structure systems?

To answer this question, let us look once more what happens in the two switch positions.

- When the switch is closed, we are dealing with an *index-0 model*.
- When the switch is open, we have two storage elements, but only one degree of freedom. Hence are dealing with an *index-2 model*.
- Thus, *variable structure systems* lead to *conditional index changes*.
- We already knew that, using inline integration, we can simulate some index-2 models directly. This was also the case here.
- However, we mentioned that it is better to perform index reduction first, e.g. using the Pantelides algorithm.
- Unfortunately, the Pantelides algorithm usually doesn't work when dealing with conditional index changes.

Mixed-mode Integration IX

- Let us consider the case of two rotational bodies on two separate axles connected by a clutch.
- When the clutch is free, the two bodies move independently, each being modeled by a second-order differential equation. Thus, we are dealing with an *index-0 model*.
- When the clutch is locked, the two bodies move together, causing a constraint among the rotational positions of the two bodies. Thus, we are now dealing with an *index-3 model*.
- In this situation, mixed-mode integration will probably not be able to save our neck.

Mixed-mode Integration X

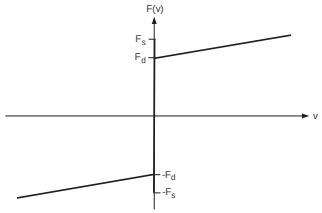
- **Sol** solves this problem in a different way.
- Sol uses *dynamic causalization*, i.e., whenever a *structural change* occurs in a *variable-structure system*, Sol interrupts the simulation, and calls upon an *incremental model compiler* to re-causalize those equations, the causality of which has changed.
- Once the new causalization has been obtained, the simulation is resumed, interpolating between the final values of the previous simulation segment to obtain the initial conditions of the new segment. Since the set of state variables has changed, we need to determine new initial conditions from the information available just prior to the structure change.

Mixed-mode Integration XI

- **Mosilab**, another dialect of the **Modelica** language, chose yet another approach.
- The Mosilab model compiler uses *static causalization*, but generates separate simulation models for all combinations of switches at compile time, together with conditions of when to change from one set of simulation equations to another.
- Unfortunately, the Mosilab approach doesn't scale well, because the number of simulation models grows exponentially with the number of switches, and because the set of initial conditions of each of these models grows quadratically in the number of simulation models.

Mechanical Friction

A typical friction force characteristic is shown below:



- There are three different regions (modes) of this nonlinear model: a *backward* mode, a *sticking* mode, and a *forward* mode.
- While the velocity of the articulation is zero, the articulation operates in its sticking mode. It will remain in this mode, until the sum of forces applied to this articulation becomes either larger than the positive sticking friction force, F_s, or smaller than the negative sticking friction force, $-F_s$.
- When this happens, the articulation comes out of sticking friction, and changes its operational mode to moving either forward or backward. It will remain in that mode, until the velocity of the articulation crosses through zero, at which time the model will return to its sticking mode.

Mechanical Friction II

- The friction force is a known applied force if the velocity v is different from zero. In that situation, the computational causality of the friction model is such that the velocity is an input to the model, whereas the friction force is its output.
- When the velocity becomes zero, the two bodies, between which the friction force is acting, become stuck. In this situation, *the model changes its structure*: A new equation, $v = 0.0$, and a new unknown force, F_c, are added to the model. The constraint force F_c is determined such that the new constraint on the velocity, $v = 0.0$, is always met.
- This is a new situation as compared to the electrical switch, because the electrical switch toggles between two different equations one of which is always active. Thus, the number of equations remains the same. In contrast, the friction element adds one equation and one variable to the model, when v becomes 0, and removes them again, when $\mathrm{abs}(F_c)$ becomes larger than the threshold value F_s.

Mechanical Friction III

- **Modelica** does not allow to add/remove variables and/or equations during the simulation. Therefore, a dummy equation is added that becomes active, when the constraint equation, $v = 0.0$, is removed. The dummy equation is used to provide a unique –but arbitrary– value for F_c during sliding motion. For example, $F_c = 0.0$ is as good a value as any.
- **Sol** and **Mosilab** would have allowed us to add/remove equations and/or variables, but we wish to discuss the more compact, albeit less readable, **Modelica** solution for now.
- The friction force F can thus be defined using the following **Modelica** code:

$$\begin{aligned} F &= \textbf{if } v > 0 \textbf{ then } c_f \cdot v + F_d \textbf{ else} \\ &\quad\ \textbf{if } v < 0 \textbf{ then } c_f \cdot v - F_d \textbf{ else } F_c \\ 0 &= \textbf{if } \mathit{Sticking} \textbf{ then } v \textbf{ else } F_c \end{aligned}$$

- The third equation is our meanwhile well-known *switch equation*.

Mechanical Friction IV

- This very elegant and compact friction model has only one disadvantage: it won't work.
- As the velocity, v, is usually a state variable, it is considered known, and consequently, the switch equation must always compute F_c.
- Since the causality of the switch equation is fixed, we will end up with a division by zero.
- I told you that the *Pantelides algorithm* usually doesn't work for *conditional index changes*. However, it sometimes does, and this is one of the cases, where this happens.
- If the velocity, v, is zero in the sticking mode, then also $a = 0$.
- Hence we can replace the velocity by the acceleration in the switch equation:

$$\begin{aligned} F &= \textbf{if } v > 0 \textbf{ then } c_f \cdot v + F_d \textbf{ else} \\ &\quad\ \textbf{if } v < 0 \textbf{ then } c_f \cdot v - F_d \textbf{ else } F_c \\ 0 &= \textbf{if } \mathit{Sticking} \textbf{ then } a \textbf{ else } F_c \end{aligned}$$

Mechanical Friction V

- This model works, but since we no longer state explicitly that $v = 0.0$, while in sticking mode, there may be a small remaining velocity from the iteration on the state event that subsequently leads to a slow drift.
- We can avoid that problem by augmenting the model:

$$\begin{aligned} F &= \textbf{if } v > 0 \textbf{ then } c_f \cdot v + F_d \textbf{ else} \\ &\quad\ \textbf{if } v < 0 \textbf{ then } c_f \cdot v - F_d \textbf{ else } F_c \\ 0 &= \textbf{if } \mathit{Sticking} \textbf{ then } a \textbf{ else } F_c \\ &\textbf{when } \mathit{Sticking} \textbf{ then} \\ &\quad \mathrm{reinit}(v, 0); \\ &\textbf{end when}; \end{aligned}$$

Mechanical Friction VI

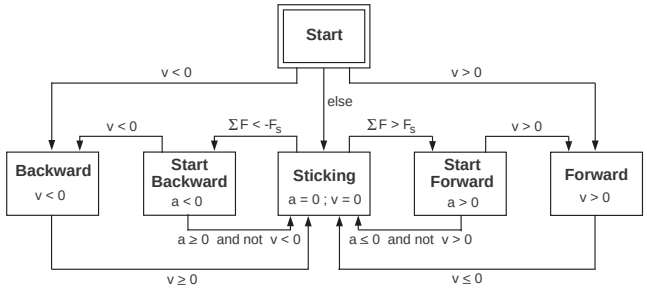
- We haven't defined yet, how the *discrete state variable*, *Sticking*, is computed by the model. To this end, we need to define, how the switching between the sliding and the sticking phases takes place.
- It is advantageous to split the friction force law into the following five different regions:

region:		region conditions:	
Forward	:	$v > 0$	and $F = c_f \cdot v + F_d$
StartForward	:	$v = 0$ and $a > 0$	and $F = +F_d$
Sticking	:	$v = 0$ and $a = 0$	and $F \in [-F_s, +F_s]$
StartBackward	:	$v = 0$ and $a < 0$	and $F = -F_d$
Backward	:	$v < 0$	and $F = c_f \cdot v - F_d$

- Unfortunately, the above five regions cannot be encoded directly, because the equality relation "=" appears in the conditional expressions of **if**-statements. It is not meaningful to test computed real-valued variables for being equal to zero.

Mechanical Friction VII

We shall use an indirect approach. The switching between the five regions is described by a *deterministic finite state machine (DFSM)* represented by the following *state transition diagram*:



Mechanical Friction VIII

We are now ready to encode to complete friction model:

```
Forward       = pre(Start)                 and v > 0 or
                pre(StartForward)          and v > 0 or
                pre(Forward)               and not v <= 0;

Backward      = pre(Start)                 and v < 0 or
                pre(StartBackward)         and v < 0 or
                pre(Backward)              and not v >= 0;

StartForward  = pre(Sticking)              and F_c > +F_s or
                pre(StartForward)          and not
                (v > 0 or a <= 0 and not v > 0);

StartBackward = pre(Sticking)              and F_c < -F_s or
                pre(StartBackward)         and not
                (v < 0 or a >= 0 and not v < 0);

Sticking      = not (Start or
                Forward or StartForward or
                Backward or StartBackward);
```

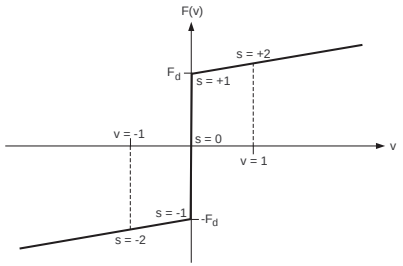
```
F = if Forward       then c_f * v + F_d else
    if Backward      then c_f * v - F_d else
    if StartForward  then +F_d          else
    if StartBackward then -F_d          else F_c;

0 = if Sticking or Start then a else F_c;

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


Mechanical Friction IX

- This model can be simulated. Unfortunately, the conditional expression accompanying the switch equation is anything but smooth. Consequently, we are forced to use golden section to iterate on the switch equation.
- Let us see whether we can improve on the situation by using the *parameterized curve description* method.

Mechanical Friction X

A slightly simplified parameterized friction force characteristic is shown below:



The curve parameter is defined as follows:

region	s
forward	$v + 1$
sticking	$\frac{F}{F_d}$
backward	$v - 1$

Mechanical Friction XI

- Curve parameters can be defined in any way that is most suitable.
- They don't have to be equidistantly spaced, and they can even adopt different units in different regions, as the example demonstrates.
- Using the new variable, s, we can define the simplified friction model as follows:

$$
\begin{array}{lll}
\mathit{Forward} & = s > +1; & \\
\mathit{Backward} & = s < -1; & \\
v = \textbf{if } \mathit{Forward} & \textbf{then } s - 1 & \textbf{else} \\
\quad \textbf{if } \mathit{Backward} & \textbf{then } s + 1 & \textbf{else } 0; \\
F = \textbf{if } \mathit{Forward} & \textbf{then } c_f \cdot (s - 1) + F_d & \textbf{else} \\
\quad \textbf{if } \mathit{Backward} & \textbf{then } c_f \cdot (s + 1) - F_d & \textbf{else } F_d \cdot s;
\end{array}
$$

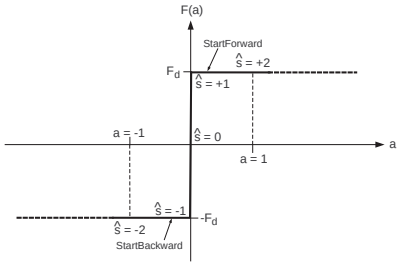
- This model is correct in the sense that it describes unambiguously our intentions of what the model is supposed to accomplish.

Mechanical Friction XII

- Unfortunately, the **Dymola** model compiler, as it is currently implemented, is unable to generate executable simulation code from that model.
- While the model operates in its *Forward* region, the velocity, v, is a state variable, thus can be assumed known. Hence the curve parameter, s, can be computed from the equation $s = v + 1$, and the friction force can be obtained using the equation $F = c_f \cdot (s - 1) + F_d$.
- When s becomes smaller than $+1$, the model enters its *Sticking* region. In this region, we have the equation; $v = 0$. Thus, the velocity, v, can no longer be treated as a known state variable, and we are confronted with a *conditional index change*.
- We can tackle the problem using the same argumentation that had been used already in the previous model: If the velocity, v, is constantly equal to zero over a period of time, then also the acceleration, a, must be constantly equal to zero during that time period.

Mechanical Friction XIII

Hence we parameterize the friction force characteristic differently:



The curve parameter is defined as follows:

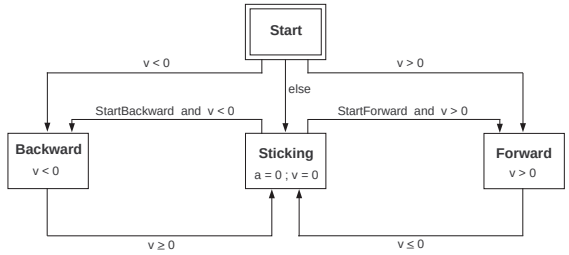
region	$\hat{s}$
forward	$a + 1$
sticking	$\frac{F}{F_d}$
backward	$a - 1$

Mechanical Friction XIV

We code the so parameterized force characteristic as follows:

$$
\begin{aligned}
&\mathit{StartForward} &&= \hat{s} > +1;\\
&\mathit{StartBackward} &&= \hat{s} < -1;\\
&a = \textbf{if } \mathit{StartForward} \textbf{ then } \hat{s} - 1 \textbf{ else}\\
&\quad \textbf{if } \mathit{StartBackward} \textbf{ then } \hat{s} + 1 \textbf{ else } 0;\\
&F = \textbf{if } \mathit{StartForward} \textbf{ then } +F_d \textbf{ else}\\
&\quad \textbf{if } \mathit{StartBackward} \textbf{ then } -F_d \textbf{ else } F_d \cdot \hat{s};
\end{aligned}
$$

The following *state transition diagram* describes the transitions between the different modes:



Mechanical Friction XV

- This is a simplified version of the state transition diagram used by the earlier model. The new *finite state machine* has only four instead of six discrete states (regions).
- The *StartForward* and *StartBackward* modes of operation are no longer considered separate regions. Instead, they are contained within the *Sticking* region model. They only represent different aspects of the *Sticking* region.
- This is the friction model that is currently being offered as part of the *mechanics sub-library* of the **Modelica Standard Library**.

Petri Nets

We shall now demonstrate that it is always possible to decompose complex (combined) event conditions into sets of simple event conditions that consist of a single relational operator only.

Thus, all zero-crossing functions can be made smooth.

To this end, we shall introduce a new model description tool: the *Petri net*.

Petri Nets II

- Petri nets consist of two modeling elements: *places* and *transitions*.
- Places are holders of *tokens*.
- Each place maintains a discrete state variable that counts the number of tokens currently held by the place.
- Transitions connect places.
- When a transition *fires*, it takes some tokens out of places connected at its inputs, and places some new tokens at places connected at its outputs in accordance with some logic to be defined.
- A transition may fire, when an external firing condition is true, if the conditions concerning the necessary numbers of tokens held by its input places are also true.
- If one place feeds several transitions, additional logic may be required to determine firing preferences in the case of simultaneous events, i.e., in the case where the external firing conditions of several transitions become true simultaneously, because there may be enough tokens in the input place to fire one or the other of these transitions, but not all of them.

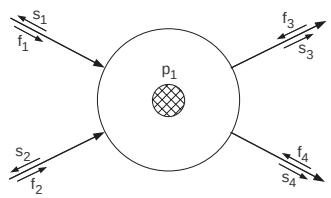
Petri Nets III

Many different dialects of Petri nets have been described in the literature.

- *Bounded Petri nets* are Petri nets with capacity limitations imposed on their places.
- *Normal Petri nets* are Petri nets with a capacity limit of one imposed on each place. In a normal Petri net, the discrete state counting the number of tokens contained in a place can thus be represented as a Boolean state. If the state has a value of *true*, there is a token located at the place. If the state has a value of *false*, there is no token at the place.
- *Priority Petri nets* resolve the ambiguity associated with multiple transitions being able to fire simultaneously by associating a prioritization scheme to these transitions.
- *Normal priority Petri nets (NPPNs)* are normal Petri nets employing prioritization schemes in all of their transitions.

Petri Nets IV

A NPPN place with two inputs and two outputs is shown below:



The place passes state information, s_i, to all neighboring transitions, and in turn receives firing information, f_i, back from these transitions.

The NPPN place could be governed by the following equations:

$s_1 = \text{pre}(p_1)$
$s_2 = \text{pre}(p_1)$ **or** f_1
$s_3 = \text{pre}(p_1)$
$s_4 = \text{pre}(p_1)$ **and not** f_3
$p_1 = [\text{pre}(p_1)$ **and not** $(f_3$ **or** $f_4)]$ **or** f_1 **or** f_2

Petri Nets V

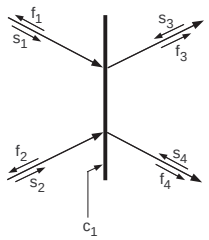
- The place first provides the first input transition, t_1, with its state information. Transition t_1 needs to know this information, because, due to the single-token capacity limitation, it cannot fire, unless the place, p_1, is currently unoccupied. The place receives the firing information, f_1, back from transition t_1. If t_1 fires, it means that it is going to place a new token at p_1.
- The place then provides the appropriate state information, s_2, to the second input transition, t_2. Transition t_2 is assigned a lower priority than transition t_1. Transition t_2 is not allowed to fire if either there is already a token at place p_1, or if the other input transition, t_1, decided to fire, because if both transitions were to fire simultaneously, they both would try to place a token at p_1, which would violate the imposed capacity limit of one.
- The place then provides its state information to the first output transition, t_3. Transition t_3 is allowed to fire if a token is currently at p_1. If it fires, it will take the token away from place p_1.

Petri Nets VI

- The place then provides its state information to the second output transition, t_4. Transition t_4 is assigned a lower priority than transition t_3. Transition t_4 is not allowed to fire, unless there is currently a token at place p_1 and transition t_3 has not decided to fire, because if both transitions were to fire simultaneously, they both would fight over who gets to remove the token from p_1.
- Finally, the place must update its own state information. If there was a token at p_1 before, and neither of the two output transitions, t_3 or t_4, has taken it away, or, if one of the two input transitions, t_1 or t_2, has placed a new token at p_1, there will be a token at that place during the next cycle.

Petri Nets VII

Let us now look at a transition with two input places and two output places:



This logic can be described by the following set of equations:

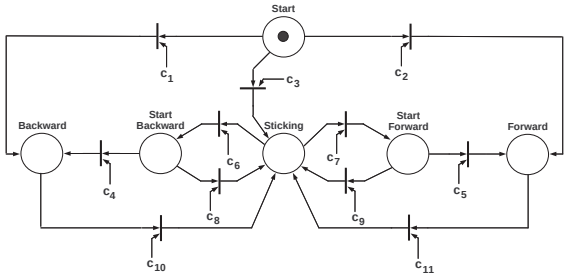
$$fire = c_1 \text{ and } s_1 \text{ and } s_2 \text{ and not } (s_3 \text{ or } s_4)$$
$$f_1 = fire$$
$$f_2 = fire$$
$$f_3 = fire$$
$$f_4 = fire$$

The transition is allowed to fire along all of its connections, when the external firing condition, c_1, is true, and if each of the input places holds a token (or more precisely, if the state information arriving from all of the input places is *true*), and if none of the output places holds a token (or more precisely, if the state information of none of the output places is *true*).

Petri Nets VIII

Deterministic finite state machines (DFSMs) can be modeled as *normal priority Petri nets* with the additional constraints that there is only one token in the system that is initially located at the *Start* place. Furthermore, DFSMs map to NPPNs, in which each transition is associated with exactly one input place and one output place.

Let us model the DFSM of the *original friction model* as a Petri net. The corresponding NPPN representation is shown below:



Petri Nets IX

What have we gained by this representation?

- We immediately recognize what the external firing conditions, c_i, represent. These are the conditions that are associated with state transitions in the DFSM. Hence those are the edge-triggered Boolean variables associated with the zero-crossing functions.
- In the past, we had many different discrete event blocks representing the actions to be taken, when one or the other of the zero-crossing functions triggered an event. This is no longer the case.
- All of the discrete equations governing both *places* and *transitions* are valid at every event, since they were formulated as functions of the current location of the tokens, i.e., they were functions of the discrete state that the system is currently operating in.

Petri Nets X

Thus, every discontinuous model, as complex as it may be, can be described by exactly three sets of equations:

- There are the *implicitly defined algebraic and differential equations* describing the continuous subsystem.
- There is the set of *zero-crossing functions* that are all evaluated in parallel, while the continuous subsystem is being simulated. If a state event is being triggered by one of them, an iteration (or interpolation) takes place to locate the event time as accurately as necessary.
- At that moment, the third set of simultaneous equations is being executed. These are the *(possibly implicitly defined) algebraic and difference equations* describing the discrete subsystem. The discrete equations are executed iteratively, until no discrete state changes occur any longer. When this happens, we have found our new initial state, from which we can start the continuous simulation afresh.

Petri Nets XI

- A simulation model that has been compiled into this form, can be simulated in an organized and systematic fashion based on a *synchronous data flow*.
- It may not be convenient for the end user of the modeling and simulation environment to describe his or her model in this fashion. Different application domains make use of different modeling formalisms that users are familiar with.
- It is the job of the *model compiler* to dissect the model description that the user supplies, and translate it down to sets of simultaneous equations that can be simulated without numerical difficulties.
- As this class concerns itself with the set of algorithms underlying a powerful modeling and simulation environment, such as **Dymola**, we had to show step by step, how model equations need to be preconditioned, until they are finally in a form such that they can be simulated without difficulties. Yet, it is inconvenient to translate every model that we came across manually down to such a form.

Petri Nets XII

Will the iteration on the simultaneous discrete equations always converge?

- If the model of a physical system is formulated correctly, the iteration should always converge, as our Newtonian world is deterministic in nature.
- Yet, it is easy to make mistakes, and formulate a set of discrete equations that will not converge. It is very easy to specify logical conditions that are contradicting themselves.
- In the Petri-net implementation, this leads to oscillations of discrete state variables with infinite frequency, a so-called *illegitimate model*, i.e., it prevents the algorithm from finding a consistent initial state, from which the continuous simulation can be started.
- For example, the discrete "equation" $p_1 = \textbf{not}\ \mathrm{pre}(p_1)$; should not be contained in the set of discrete equations, as this will lead to an oscillation between the two states *true* and *false* that will never end.
- If we mean to toggle between two discrete states as a response to a state-event being triggered (a fairly common situation), we need to model this using two separate places with transitions back and forth that get fired by zero-crossing functions.

Petri Nets XIII

How can complex zero-crossing functions be reduced to simple ones?

- *or*-conditions can be mapped to a set of parallel transitions located between the same two places. They can thus be easily implemented.
- *and*-conditions are harder to implement, as they would require transitions to be placed in series with each other. Unfortunately, this cannot be done without introducing a new place between them. Thus, *and*-conditions invariably call for an increase in the number of discrete states.

Petri Nets XIV

We shall demonstrate this concept by means of the *modified friction DFSM*.

- We recognize that we wouldn't need the *and*-conditions on the zero-crossing functions in this example, if we were to have available separate discrete states called *StartForward* and *StartBackward*. Thus, we shall decompose the state *Sticking* again into three separate discrete states. Luckily, we know the conditions for switching between them.
- We don't need to draw the modified Petri net, as is looks exactly like the previous one. Only the interpretation of the zero-crossing functions is now different:

$c_1 = v < 0$
$c_2 = v > 0$
$c_3 = v == 0$
$c_4 = v < 0$
$c_5 = v > 0$
$c_6 = \hat{s} < -1$
$c_7 = \hat{s} > 1$
$c_8 = \hat{s} >= -1$
$c_9 = \hat{s} <= 1$
$c_{10} = v >= 0$
$c_{11} = v <= 0$

- As expected, all of the zero-crossing functions are now simple functions consisting of a single relational operation only.

Conclusions

- In this third presentation on the simulation of discontinuous systems, we looked at *variable structure models*, i.e., models with different sets of equations and/or variables during different segments of the simulation.
- We resumed the discussion of electrical switches, and demonstrated that switches that are not included in algebraic loops lead invariably to a division by zero in one of the two switch positions.
- We then showed that *inlining integrators* that constrain the causality of switches often helps to avoid divisions by zero. To this end, we introduced the concept of *mixed-mode integration*.
- Next, we looked at *mechanical friction characteristics* as a very important example of a complex variable structure system that exhibits all of the theoretical difficulties illuminated before. We made use of *state transition diagrams* and, more abstractly, *deterministic finite state machines (DFSMs)* to describe the different regions of operation of a friction characteristic and the conditions for switching from one region to another.

Conclusions II

- The presentation ended with the introduction of a new modeling tool, the *Petri net*, a low-level technique for describing discrete events, and demonstrated that DFSMs can always be represented by *normal priority Petri nets (NPPNs)*.
- This offered us a systematic way for grouping equations together so that every discontinuous system can now be described by three sets of declarative equations: the *set of continuous DAEs* describing the model equations, whereby the different segments are specified by discrete state variables that change their values only at event times; a *set of zero-crossing functions* that specify when events occur; and a *set of discrete DAEs* describing the evolvement of the discrete state variables over time.
- Hence all discontinuous systems can be described by models that can be simulated using a *synchronous data flow* approach.
- We showed that the Petri-net representation makes it furthermore possible to reduce all zero-crossing functions to simple Boolean expressions containing a single relational operator that can be implemented using smooth functions.

References

1. Cellier, F.E. and M. Krebs (2007), "Analysis and Simulation of Variable Structure Systems Using Bond Graphs and Inline Integration," *Proc. ICBGM07, 8th SCS Intl. Conf. on Bond Graph Modeling and Simulation*, San Diego, California, pp. 29-34.
2. Otter, M., H. Elmqvist, and F.E. Cellier (1996), "Modeling of Multibody Systems with the Object-Oriented Modeling Language Dymola," *J. Nonlinear Dynamics*, **9**(1), pp.91-112.
3. Krebs, Matthias (1997), *Modeling of Conditional Index Changes*, MS Thesis, Dept. of Electr. & Comp. Engr., University of Arizona, Tucson, AZ.
4. Zimmer, Dirk (2010), *Equation-based Modeling of Variable-structure Systems*, Ph.D. Dissertation, Dept. of Computer Science, ETH Zurich, Switzerland.