Equation-Based Modeling of Variable-Structure Systems

ETH Zürich, March 15, 2010

Presenting

SOL

An Object-Oriented Modeling Language
for Variable-Structure Systems.

PhD Examination of
Dirk Zimmer

ETH Zürich, Department of Computer Science

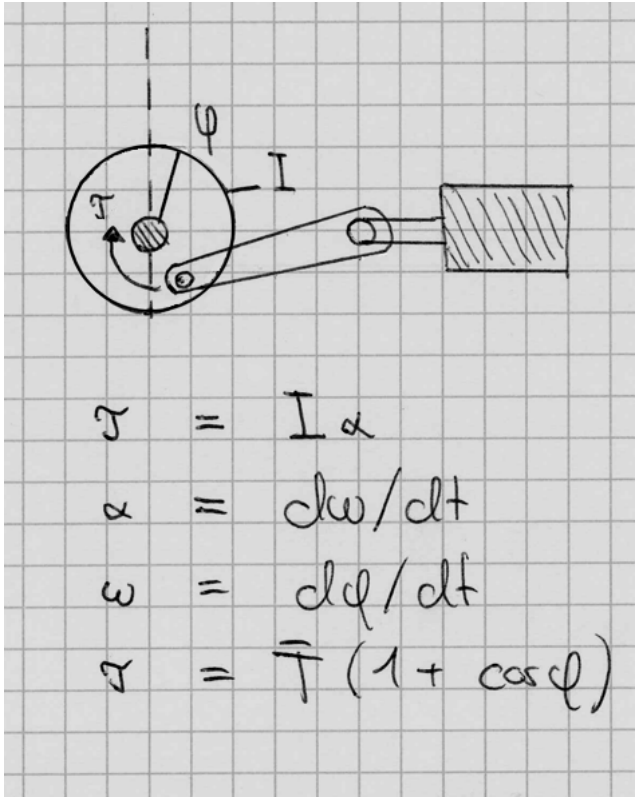
Outline

- Equation-Based Modeling in Sol
- Variable-Structure Systems
- Dynamic Processing of DAEs
- Example: The Trebuchet
- Conclusions

Equation-Based Modeling

- Physical Models are most generally and conveniently expressed by using differential-algebraic equations (DAEs).

- Equation-based computer languages like Sol enable the modeler to formulate his models directly by terms of equations.



Equation-Based Modeling

ETH Zürich
Department of Computer Science

- Physical Models are most generally and conveniently expressed by using differential-algebraic equations (DAEs).
- Equation-based computer languages like Sol enable the modeler to formulate his models directly by terms of equations.
- The Sol model describes an engine driving a flywheel.
- The resulting modeling file can then be used for simulation.

```
model SimpleMachine

define inertia as 1.0;

interface:
  parameter Real meanT;
  static Real w;

implementation:
  static Real phi;
  static Real t;
  static Real a;

  t = inertia*z;
  z = der(x=w);
  w = der(x=phi);
  t = (1+cos(x=phi))*meanT;

end SimpleMachine;
```

Object Orientation

- For large systems of equations, it is inconvenient and error-prone to write them down as a whole.
- Instead, the system shall be composed from reusable sub-models.
- To this end, each model in Sol consists in three parts that enable its generic usage:
  - The header
  - The interface
  - The implementation

```
model SimpleMachine

define inertia as 1.0;

interface:
  parameter Real meanT;
  static Real w;

implementation:
  static Real phi;
  static Real t;
  static Real a;

  t = inertia*z;
  z = der(x=w);
  w = der(x=phi);
  t = (1+cos(x=phi))*meanT;

end SimpleMachine;
```

Object Orientation

- Every organizational entity in Sol represents a model (One-Component approach).
- For instance, packages are models that consist solely of a header
- In this way, we can build a complete model library using Sol models.
- The example on the right shows excerpts from a library for 1D-rotational mechanics

```
package Mechanics […]
  model Engine2
  extends Interfaces.OneFlange;
  interface:
    parameter Real meanTorque;
  implementation:
    static Real transm;
    transm = 1+cos(x = f.phi);
    f.t = meanTorque*transm;
  end Engine2;

  model FlyWheel
  extends Interfaces.OneFlange;
  interface:
    parameter Real inertia;
    static Real w;
  implementation:
    static Real z;
    w = der(x=f.phi);
    z = der(x=w);
    -f.t = z*inertia;
  end FlyWheel;
[…]
end Mechanics;
```

Object Orientation

- Using these library models, we can now compose our top-model without the use of a single equation.

- This highlights the object-oriented aspects of modern equation-based languages.

```
model Machine
implementation:

  static Mechanics.FlyWheel F{inertia<<1};
  static Mechanics.Gear G{ratio << 1.8};
  static Mechanics.Engine2 E{meanTorque<<10};

  connection{a<<G.f2, b<<F.f};
  connection{a<<E.f, b<<G.f1};

end Machine;
```

Advantages of EOO-Languages

- Equation-based modeling languages are **declarative languages**.
- The modeler can focus on ***what*** he wants rather than spending his time on ***how*** to achieve a computational realization.
- This makes the models more **self-contained** and the knowledge can be conveniently organized in an **object-oriented** form.
- Although Sol has been designed from scratch, it builds upon its predecessor, namely **Modelica**. It represents the state of art with respect to the modeling of physical systems in industry.
- So what is new about Sol?

Variable-Structure Systems

- Sol supports the modeling of variable-structure systems.
- **Variable-structure systems** forms a collective term for models, where equations change during the time of simulation.
- The motivation for such models are manifold:
  - Ideal switching processes.
  - Variable number of entities or agents.
  - Variable level of detail.
  - User interaction.

Variable-Structure Systems

- A structural change may cause severe changes in the model structure. Even the exchange of a single equation may concern the whole system.

- This is why hardly any simulation environment supports this class of models in a truly general way.

- Only a few attempts have been made:
  MOSILAB, Chi, HYBRSIM, or recently Hydra

Advantages of Sol

- Most equation-based languages lack the required expressiveness to support variable-structure systems. Sol overcomes this deficiency by generalizing the prevalent language constructs.

- Thus, Sol enables **the creation and removal of equations** or even complete objects anytime during the simulation.

- To this end, the modeler describes the system in a **constructive way**, where the structural changes are expressed by conditionalized declarations.

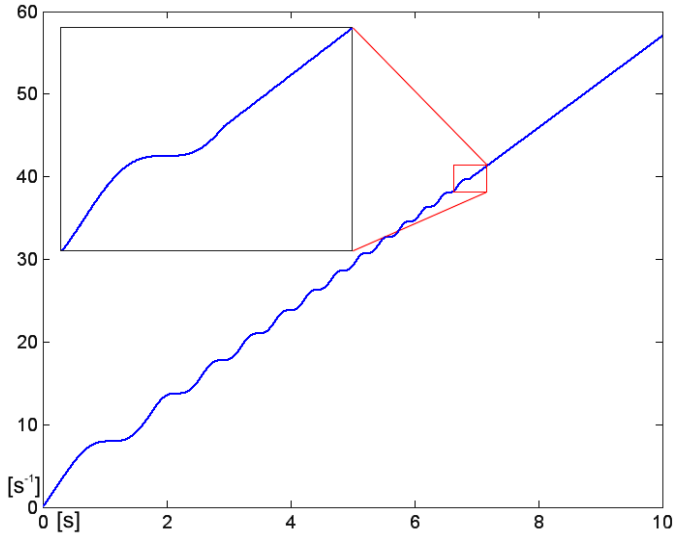
Structural Change: Example

- Let us look at a very simple example:
- The library features two different engine models. One with a constant torque and one with a fluctuating torque, roughly emulating a piston engine.
- Our intention is to use the latter, more detailed model at start and to **switch to the simpler model** as soon as the wheel's inertia starts to flatten out the fluctuation of the torque.

```
package Mechanics […]

  model Engine1
  extends Interfaces.OneFlange;
  interface:
    parameter Real meanTorque;
  implementation:
      f.t = meanTorque;
  end Engine1;

  model Engine2
  extends Interfaces.OneFlange;
  interface:
    parameter Real meanTorque;
  implementation:
    static Real transm;
    transm = 1+cos(x = f.phi);
    f.t = meanTorque*transm;
  end Engine2;

[…]
end Mechanics;
```

Structural Change: Example

- The simulation of the system is trivial.
- The plot depicts the angular velocity overtime.



Structural Change: Example

ETH Zürich
Department of Computer Science

```
model Machine
implementation:
  static Mechanics.FlyWheel F{inertia<<1};
  static Mechanics.Gear G{ratio << 1.8};
  connection{a<<F.f, b<<G.f2}
  static Boolean fast;

  if initial() then
    fast << false;
  end;

  if fast then
    static Mechanics.Engine1 E{meanT<<10};
    connection{a<<E.f, b<<G.f1};
  else then
    static Mechanics.Engine2 E{meanT<<10};
    connection{a<<E.f, b<<G.f1};
  end;

  when F.w > 40 then
    fast << true;
  end;

end Machine;
```

- The Boolean variable fast represents the current mode.

Structural Change: Example

ETH Zürich
Department of Computer Science

```
model Machine
implementation:
  static Mechanics.FlyWheel F{inertia<<1};
  static Mechanics.Gear G{ratio << 1.8};
  connection{a<<F.f, b<<G.f2}
  static Boolean fast;

  if initial() then
    fast << false;
  end;

  if fast then
    static Mechanics.Engine1 E{meanT<<10};
    connection{a<<E.f, b<<G.f1};
  else then
    static Mechanics.Engine2 E{meanT<<10};
    connection{a<<E.f, b<<G.f1};
  end;

  when F.w > 40 then
    fast << true;
  end;

end Machine;
```

- The Boolean variable fast represents the current mode.
- It is initially set to false

Structural Change: Example

ETH Zürich
Department of Computer Science

```
model Machine
implementation:
  static Mechanics.FlyWheel F{inertia<<1};
  static Mechanics.Gear G{ratio << 1.8};
  connection{a<<F.f, b<<G.f2}
  static Boolean fast;

  if initial() then
    fast << false;
  end;

  if fast then
    static Mechanics.Engine1 E{meanT<<10};
    connection{a<<E.f, b<<G.f1};
  else then
    static Mechanics.Engine2 E{meanT<<10};
    connection{a<<E.f, b<<G.f1};
  end;

  when F.w > 40 then
    fast << true;
  end;

end Machine;
```

- The Boolean variable fast represents the current mode.
- It is initially set to false
- An if-branch expresses the two modes.

Structural Change: Example


```
model Machine
implementation:
  static Mechanics.FlyWheel F{inertia<<1};
  static Mechanics.Gear G{ratio << 1.8};
  connection{a<<F.f, b<<G.f2}
  static Boolean fast;

  if initial() then
    fast << false;
  end;

  if fast then
    static Mechanics.Engine1 E{meanT<<10};
    connection{a<<E.f, b<<G.f1};
  else then
    static Mechanics.Engine2 E{meanT<<10};
    connection{a<<E.f, b<<G.f1};
  end;

  when F.w > 40 then
    fast << true;
  end;

end Machine;
```

- The Boolean variable fast represents the current mode.
- It is initially set to false
- An if-branch expresses the two modes.
- Each mode can declare its own variables or sub-models and provide equations.

Structural Change: Example

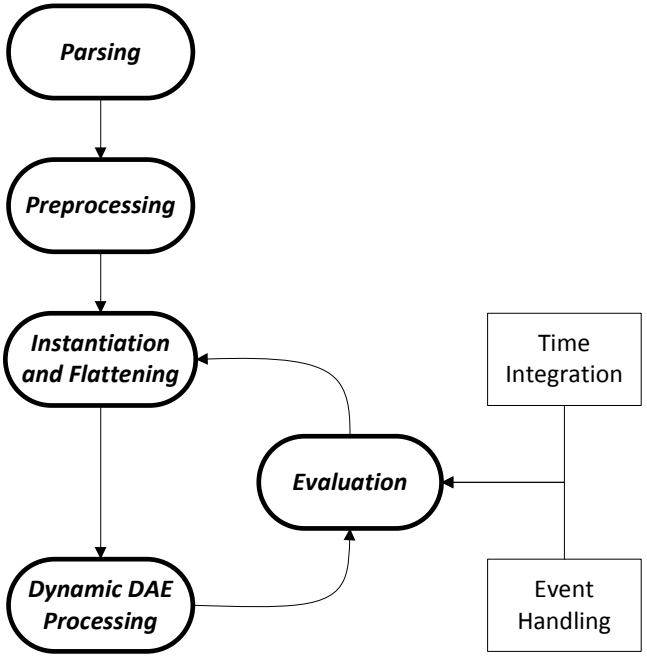
ETH Zürich
Department of Computer Science

```
model Machine
implementation:
  static Mechanics.FlyWheel F{inertia<<1};
  static Mechanics.Gear G{ratio << 1.8};
  connection{a<<F.f, b<<G.f2}
  static Boolean fast;

  if initial() then
    fast << false;
  end;

  if fast then
    static Mechanics.Engine1 E{meanT<<10};
    connection{a<<E.f, b<<G.f1};
  else then
    static Mechanics.Engine2 E{meanT<<10};
    connection{a<<E.f, b<<G.f1};
  end;

  when F.w > 40 then
    fast << true;
  end;

end Machine;
```

- The Boolean variable fast represents the current mode.
- It is initially set to false
- An if-branch expresses the two modes.
- Each mode can declare its own variables or sub-models and provide equations.
- An Event models the switch between the modes.

Solsim: Processing Scheme

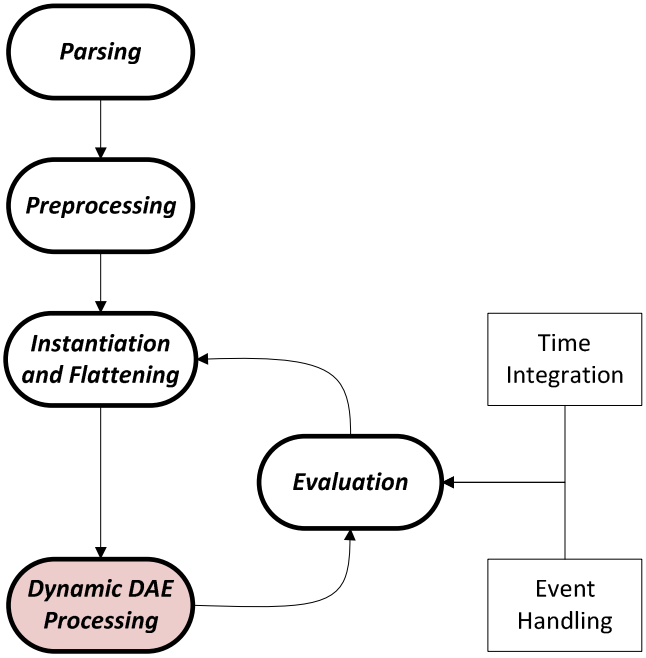
ETH Zürich
Department of Computer Science



- To understand, how such a system is simulated, let us take a look at the simulator Solsim.
- It represents an interpreter.
- The main processing loop of the interpreter contains 3 stages:
  - Instantiation and flattening
  - Dynamic causalization
  - Evaluation
- In a classic Modelica translator these stages are executed once in sequential order. In Solsim, they form a loop.

Solsim: Processing Scheme

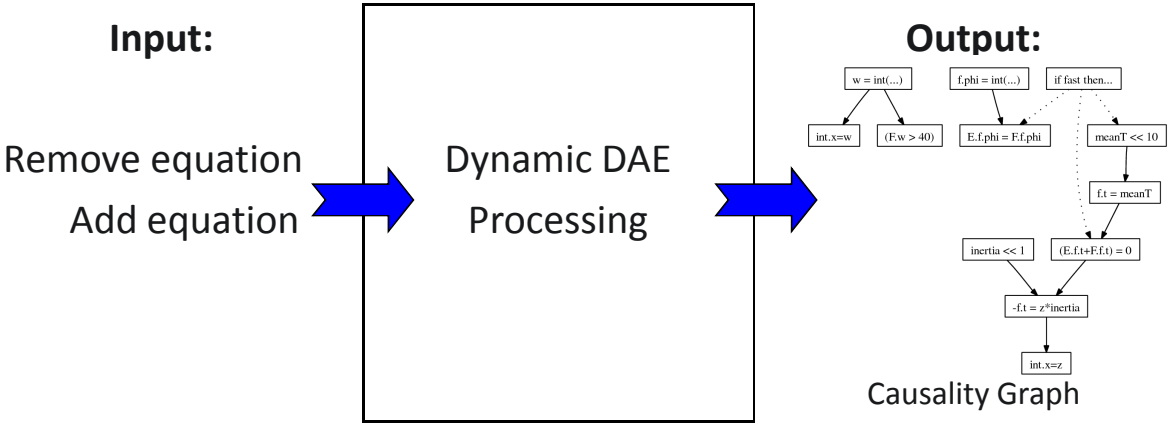
ETH Zürich
Department of Computer Science



- To understand, how such a system is simulated, let us take a look at the simulator Solsim.
- It represents an interpreter.
- The main processing loop of the interpreter contains 3 stages:
  - Instantiation and flattening
  - Dynamic causalization
  - Evaluation
- In a classic Modelica translator these stages are executed once in sequential order. In Solsim, they form a loop.

Dynamic DAE Processing

- The heart of the processing loop is the dynamic DAE processing (DDP).



Index Reduction

- The target of this processing stage is to transform the DAE form

$$F(\dot{\boldsymbol{x}}_p, \boldsymbol{x}_p, \boldsymbol{u}(t), t) = \mathbf{0}$$

into the following state-space form that suits a numerical ODE solution

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

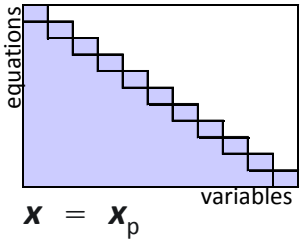
where the set of state-vector $\boldsymbol{x}$ is a sub-vector of $\boldsymbol{x}_p$.

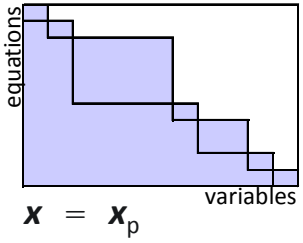
- The difficulty of this transformation is commonly described by the perturbation index of the DAE system.

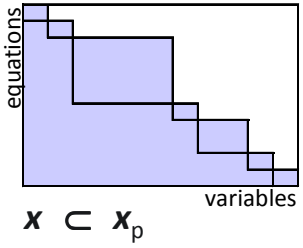
Index Reduction

There are three major cases:

- *Index-0 systems*: The system can be solved by forward substitution. Hence it is sufficient to order the equations.
  ➔ **Causalization, Topological Sorting**
- *Index-1 systems with algebraic loops*: The system cannot be solved by forward substitution anymore. It is required to solve one or several systems of equations.
  ➔ **Tearing of Loops**
- *Index-2 systems and higher-index systems*: Again forward substitution is insufficient but there are algebraic constraints between the elements of $\boldsymbol{x}_p$. It is required to differentiate a subset of equations.
  ➔ **Algorithmic Differentiation**

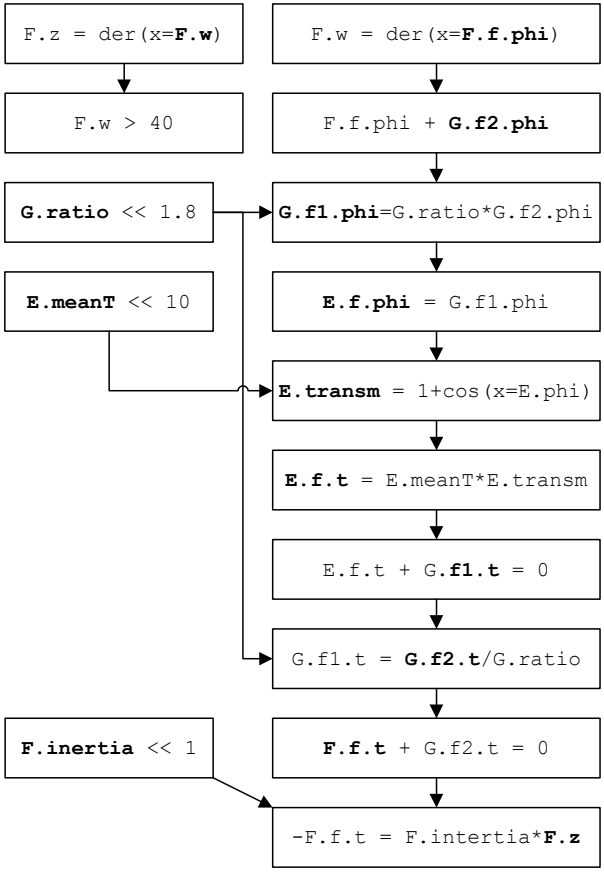






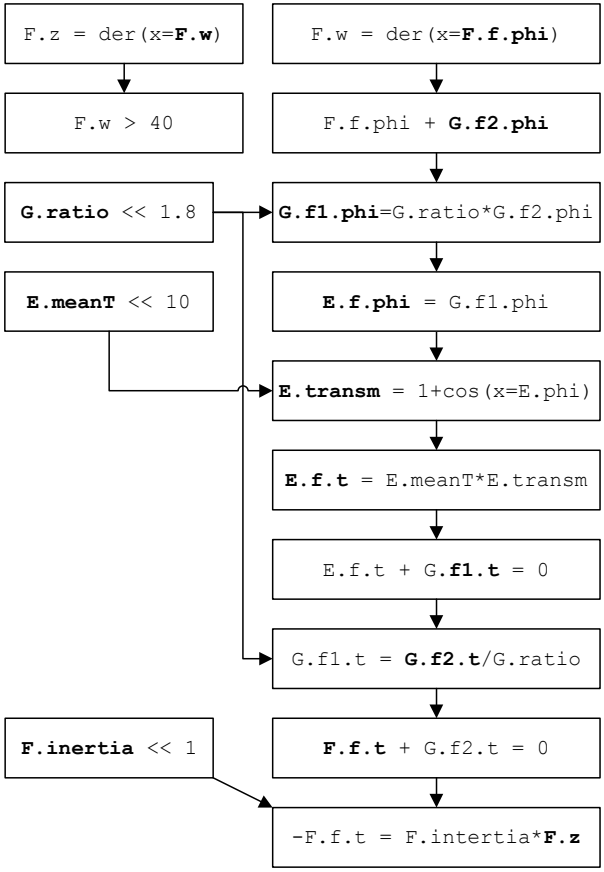
Causality Graph

- The reduced system can be represented as causality graph
- Each vertex represents an equation (or relation).
- If the equation is causalized, it points to all those equations that are dependent on the variable it determines.
- The causality graph is cycle-free and gives rise to a partial order.



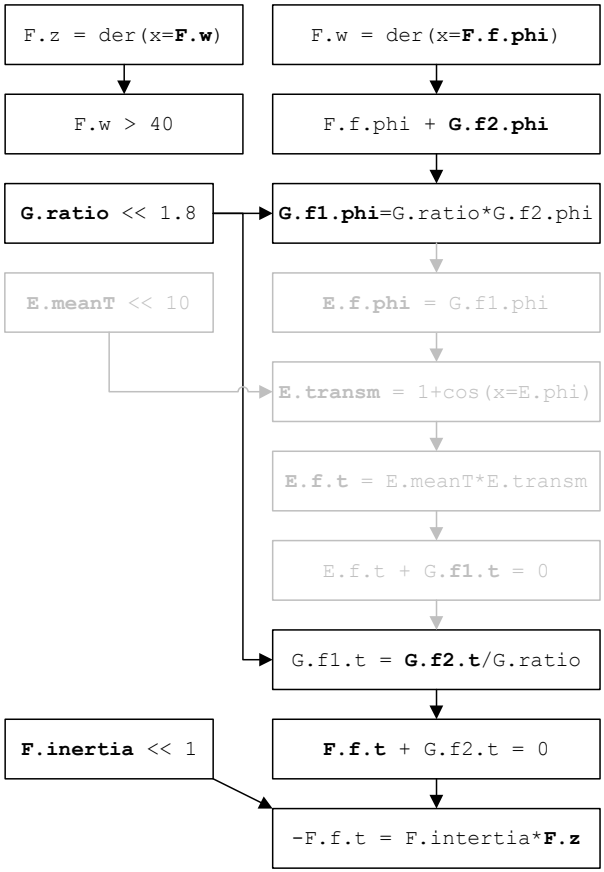
Causality Graph

- Each update in the set of equations is tracked in the graph.



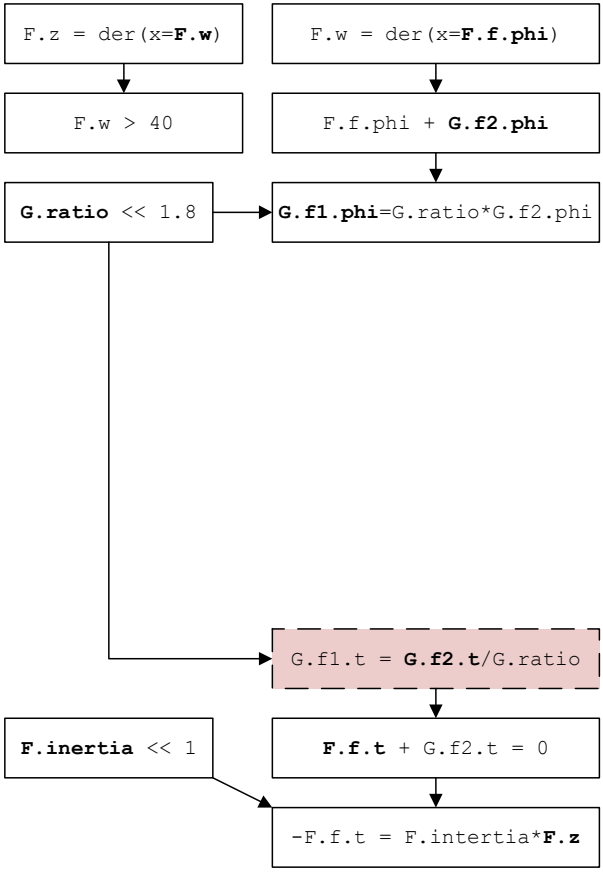
Causality Graph

- Each update in the set of equations is tracked in the graph.
- When we replace the engine model, we first remove the old equations.



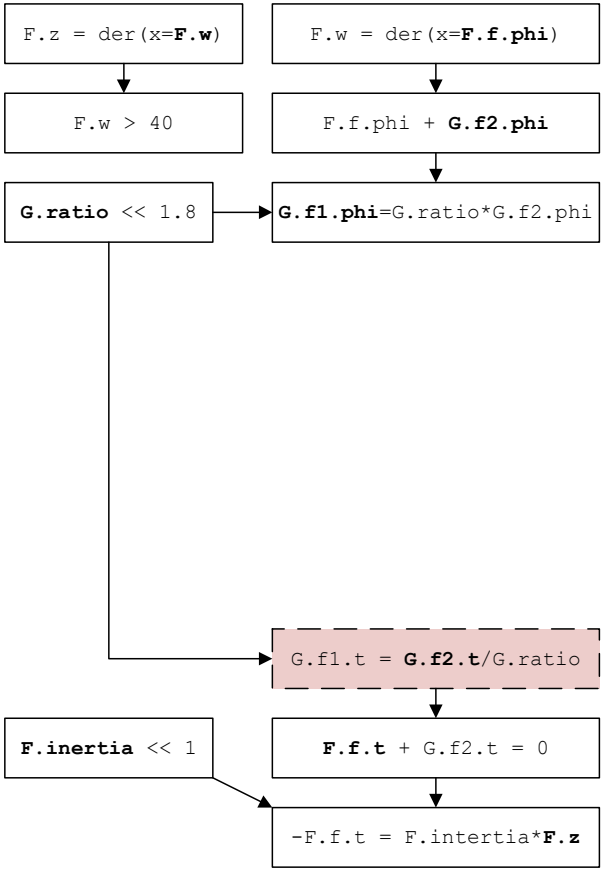
Causality Graph

- Each update in the set of equations is tracked in the graph.
- When we replace the engine model, we first remove the old equations.
- The dependent equations remain then *potentially causalized*.



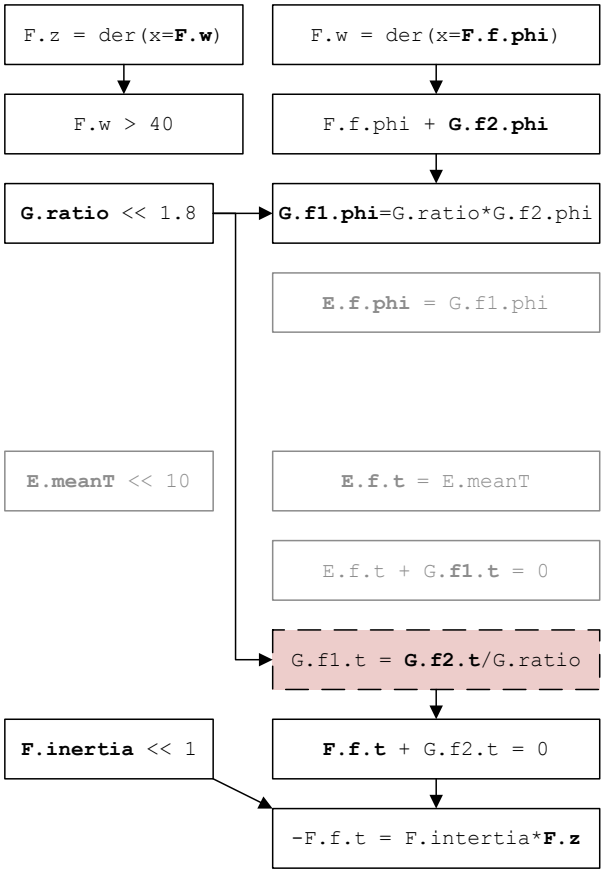
Causality Graph

- Each update in the set of equations is tracked in the graph.
- When we replace the engine model, we first remove the old equations.
- The dependent equations remain then *potentially causalized*.
- By doing so, we attempt to preserve the existing graph from overhasty changes.



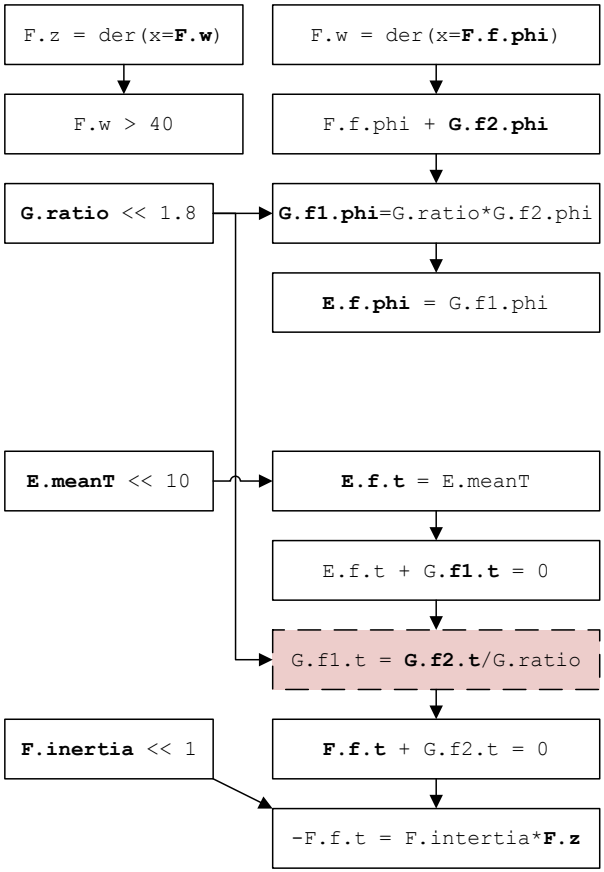
Causality Graph

- When we add the equations of the new engine model...



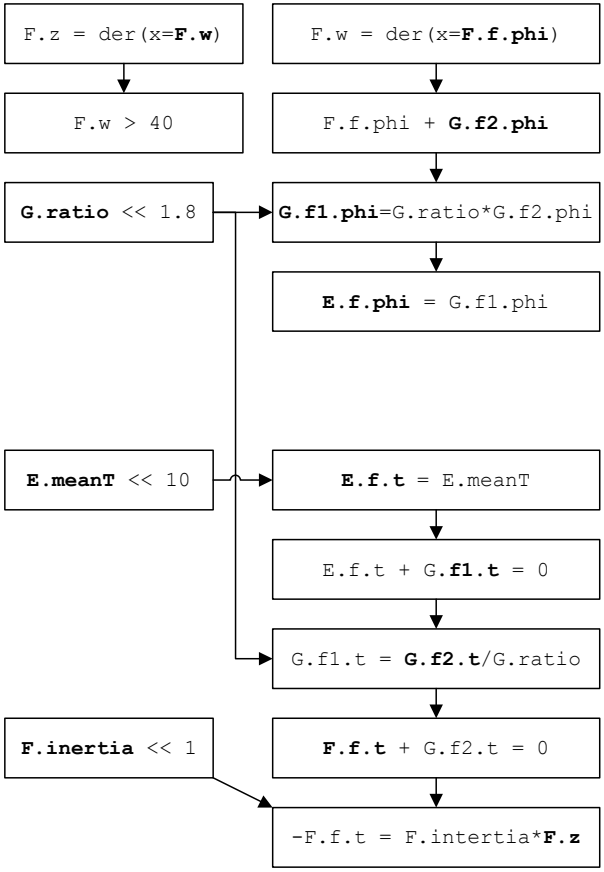
Causality Graph

- When we add the equations of the new engine model...
- … they get causalized …



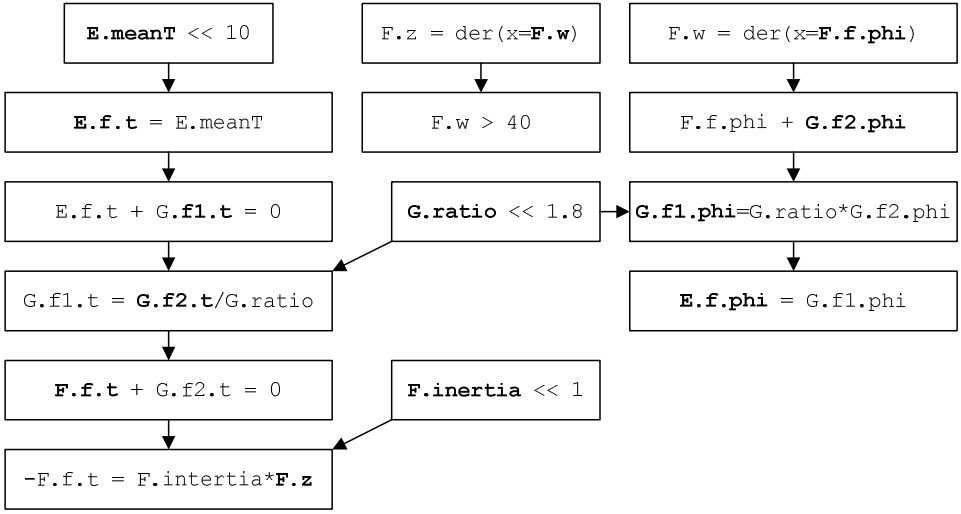
Causality Graph

- When we add the equations of the new engine model...
- … they get causalized …
- … and the potential causalization gets reinstated.



Causality Graph

- This structural change could be handled with minimal effort.
- We can redraw the graph. The left part is solely based on constant values and needs to be computed only once.

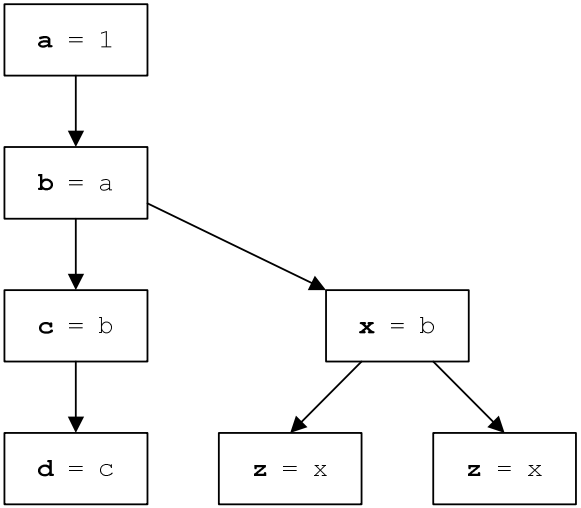


Causality Conflicts

- Unfortunately, not all structural changes are so nice….

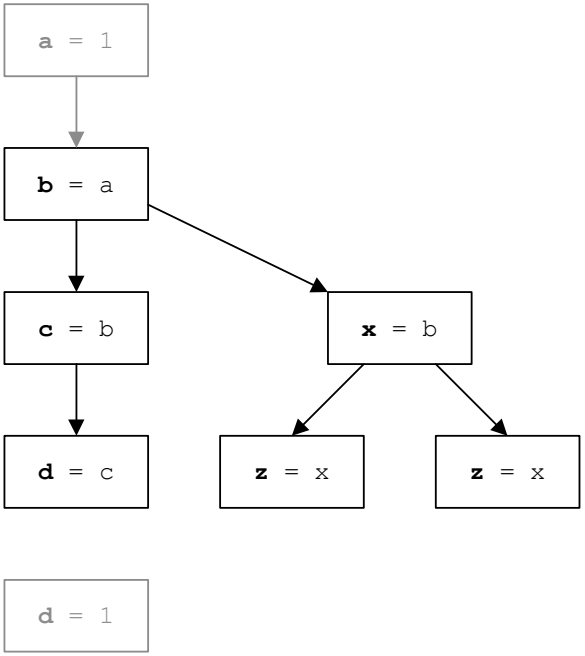
Causality Conflicts: Example

- Start



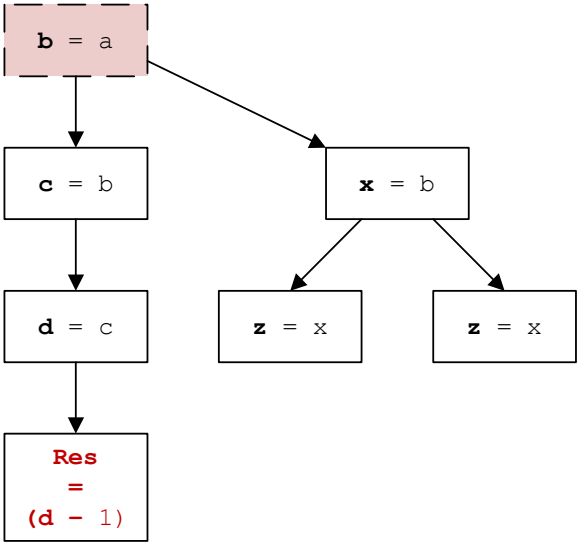
Causality Conflicts: Example

- Remove “a=1”; Add “d=1”



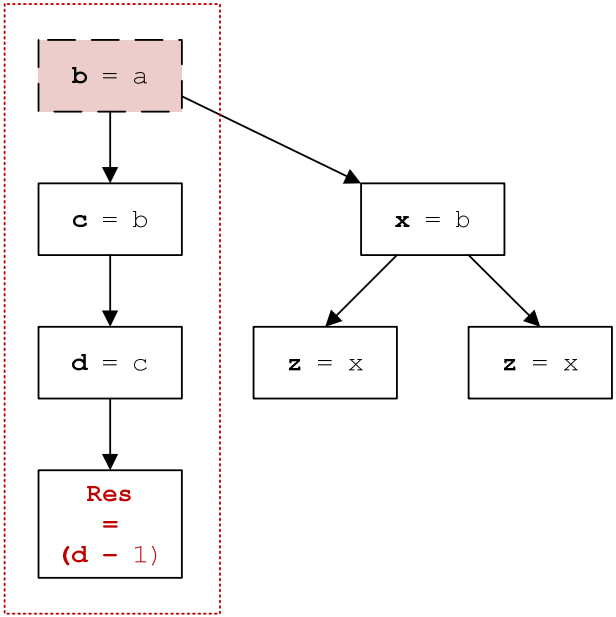
Causality Conflicts: Example

- d is now overdetermined. “d=1” is put into residual form.
- “b=a” remains potentially causalized.



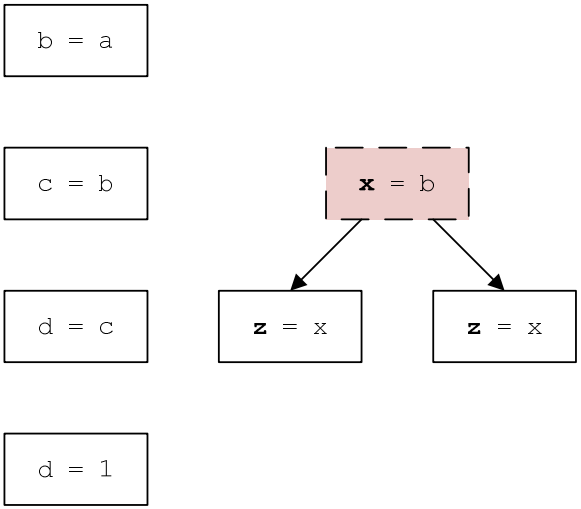
Causality Conflicts: Example

- We look up the source of overdetermination.



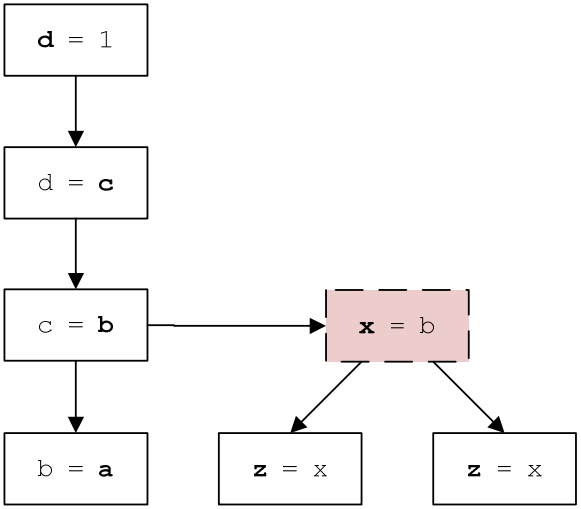
Causality Conflicts: Example

- Remove existing causalities.



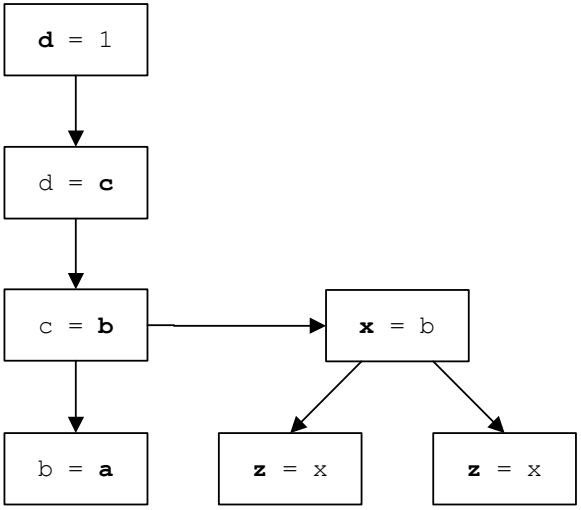
Causality Conflicts: Example

- Recausalize.



Causality Conflicts: Example

- Done.



Sources of Overdetermination

- Each equation can be in four different states:
  - Non-causalized
  - Causalized
  - Potentially causalized
  - Causalized in residual form

- Within index-0 systems, ***potentially causalized equations*** are the only source of overdetermination.

- In index-1 systems, ***tearing variables*** are another source of overdetermination.

- In higher-index systems, the ***selected state variables*** may also represent a source of overdetermination.

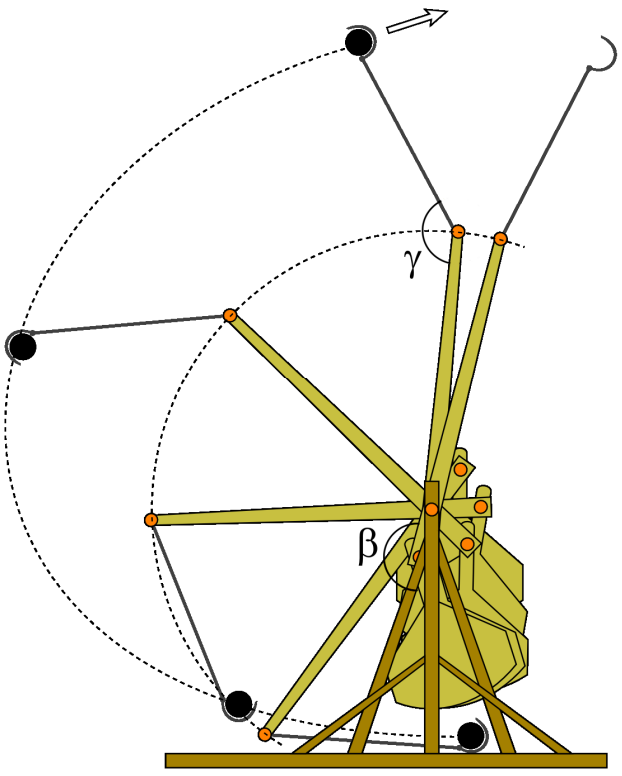
General Processing Scheme

- For all systems, there is a common approach:

  1. Perform forward causalization as much as possible.
  2. Forward causalization is supported by potential causalization, selection of tearing variables, and selection of state variables.
  3. In case of conflicts, generate residuals.
  4. Examine potential sources of overdetermination.
  5. When the source has been located, take corresponding action to resolve the conflict.

- Our approach works very well for index-0 system, it is also good for index-1 systems and represents a practicable solution for higher-index systems.

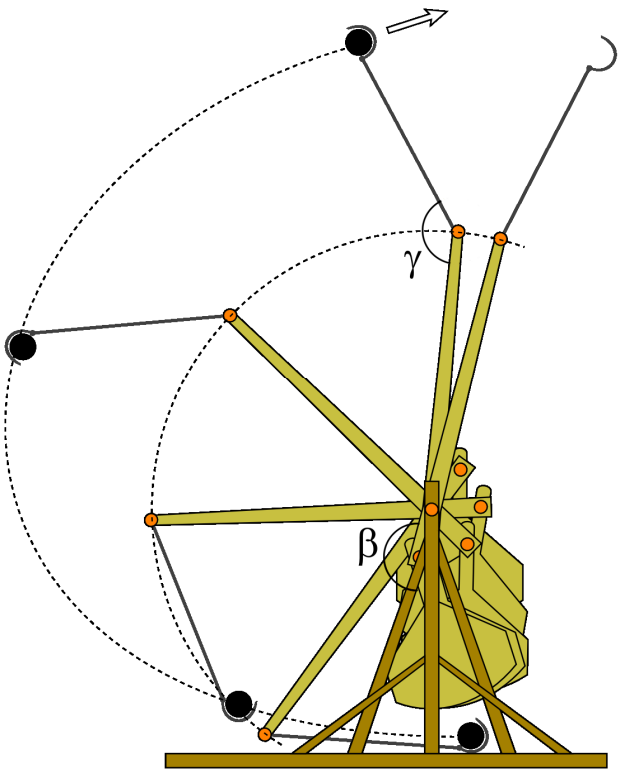
The Trebuchet



Let us demonstrate the power of Sol by means of another example:

- The trebuchet is an old catapult weapon developed in the Middle Ages.
- Technically, it is a double pendulum propelling a projectile in a sling.
- The rope of the sling is released on a predetermined angle *γ* when the projectile is about to overtake the lever arm.

The Trebuchet



Let us state a few assumptions for the model:

- All **mechanics are planar**. The positional states of any object are *x*, *y* and the orientation angle *φ*.
- All elements are **rigid**.
- The sling **rope is ideal and weightless**. It exhibits an inelastic impulse when being stretched to maximum length.
- The **revolute joint** of the counterweight **is limited** to a certain angle β. It also exhibits an inelastic impulse when reaching its limit.
- Air resistance or friction is neglected.

The Trebuchet

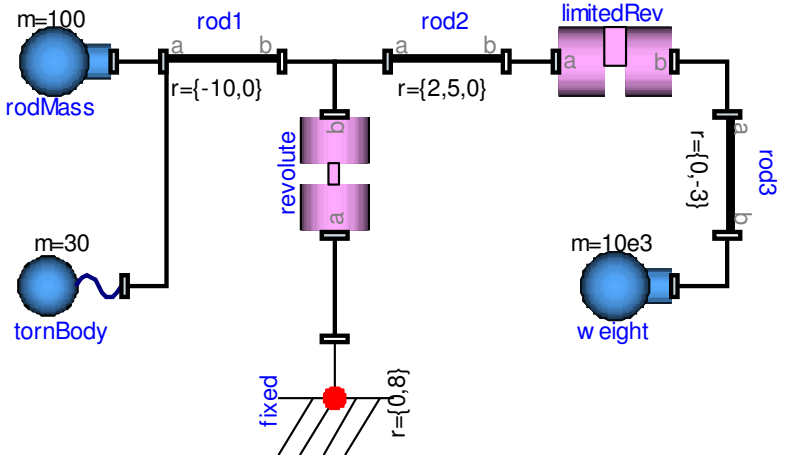
- Whereas these idealizations simplify the parameterization of the model, they pose serious difficulties for a general simulation environment.
- Although being fairly simple, the model can neither be modeled nor simulated with Modelica yet. At least not in a truly object-oriented manner.
- Indeed, the trebuchet represents a very suitable example for variable-structure systems since it puts up a broad set of requirements.

List of Requirements (incomplete):

- *The simulator must provide means for the numerical time integration.*
- *The simulator must provide means for the symbolical differentiation.*
- *The simulator must provide means for the numerical solution of linear and nonlinear systems of equations.*
- *The simulator must be able to trigger events.*
- *The simulator must be able to handle consecutive, discrete events and to synchronize them*
- *The simulator shall support the runtime instantiation and deallocation of arbitrary components.*
- *Changes in the set of DAEs shall be handled in an efficient manner that provides a general solution.*
- *…*

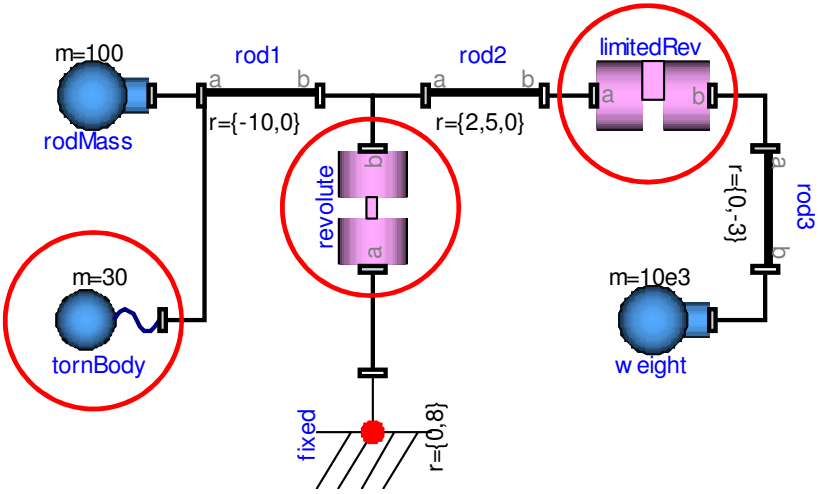
ETH Zürich
Department of Computer Science

- For the object-oriented modeling, we decompose the model into components from a planar mechanical library.



- The total model contains from 246 to 256 variables. The corresponding systems of DAEs have the perturbation index 3. They need to be differentiated twice and they contain linear equation systems.

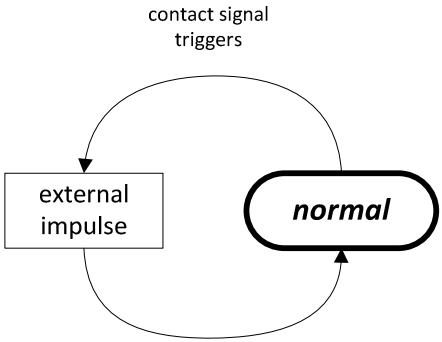
- Three of these components exhibit structural changes:



- Whereas the top-model can be neatly decomposed into generally applicable components, the modeling of these components requires a skilled modeler.

Revolute Joint: Modes

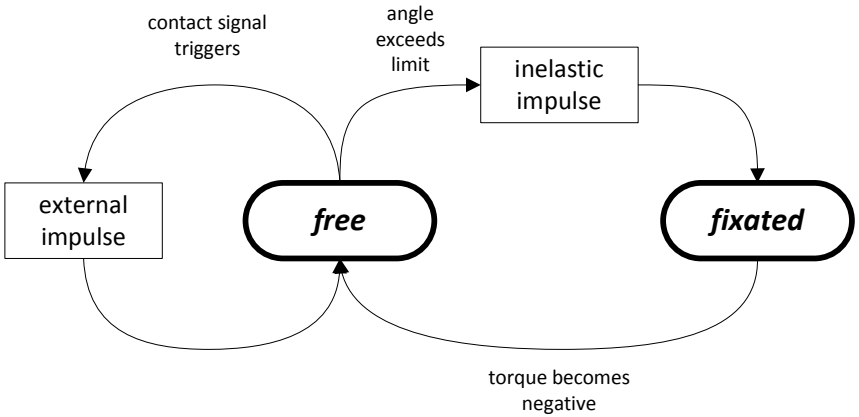
- The classic revolute joint just has one continuous-time mode.



- An intermediate mode is required in order to handle external impulses that cause a discrete change in angular velocity.

Limited Revolute: Modes

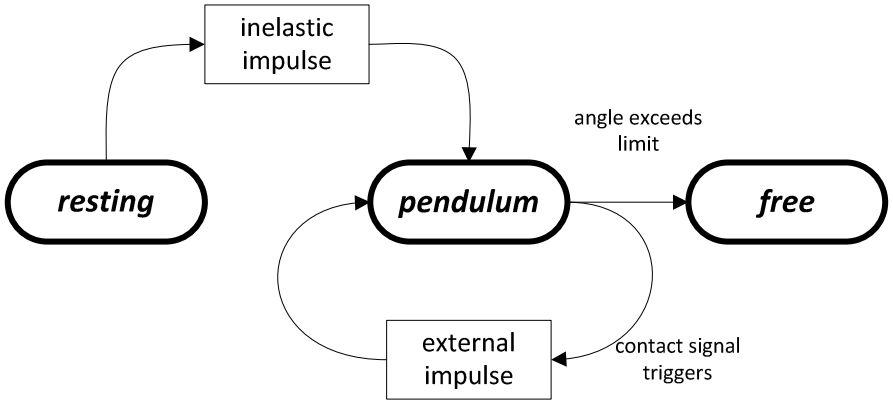
- An elbow is one possible representation of a limited revolute joint.
- The model has two major modes: *free* and *fixated*.



- Since the transition between these two states causes a discrete change in velocity, it involves an inelastic impulse that acts on the rigidly connected components.

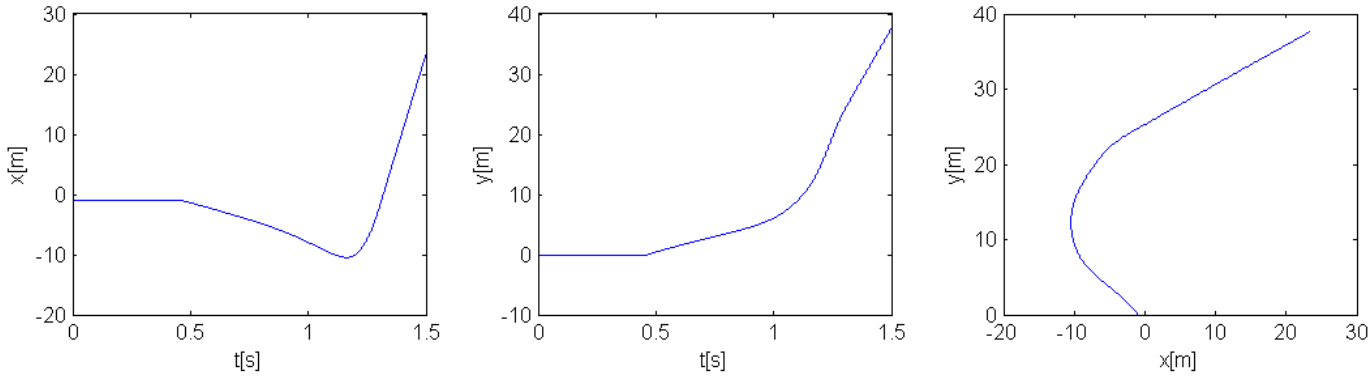
Torn Body: Modes

- The torn-body also exhibits structural changes. The model has three continuous-time modes: *resting, pendulum,* and *free*.



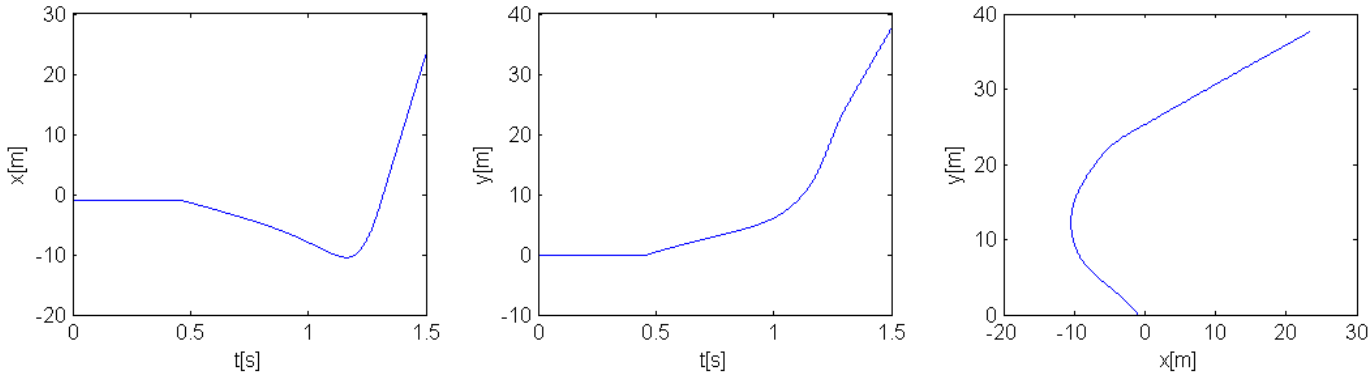
- Each mode has its own state variables:
  Resting: { }
  Pendulum: {φ, ω }
  Free: {x, y, φ, v_x, v_y, ω}

Simulation Results



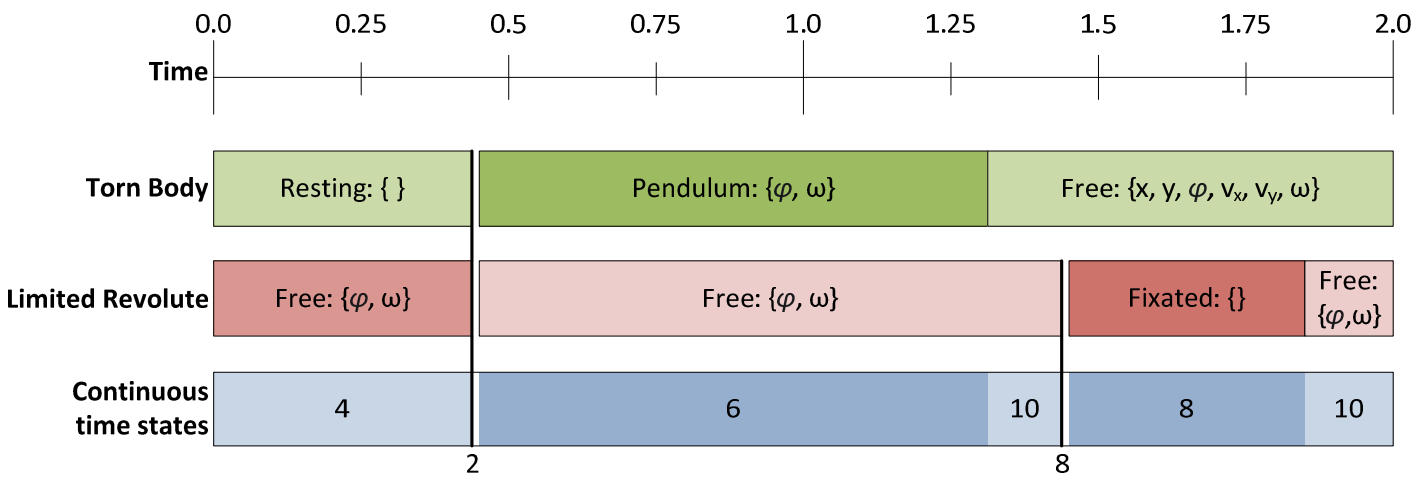
- The plots display the result of the simulation for the first 1.5 seconds.

Simulation Results



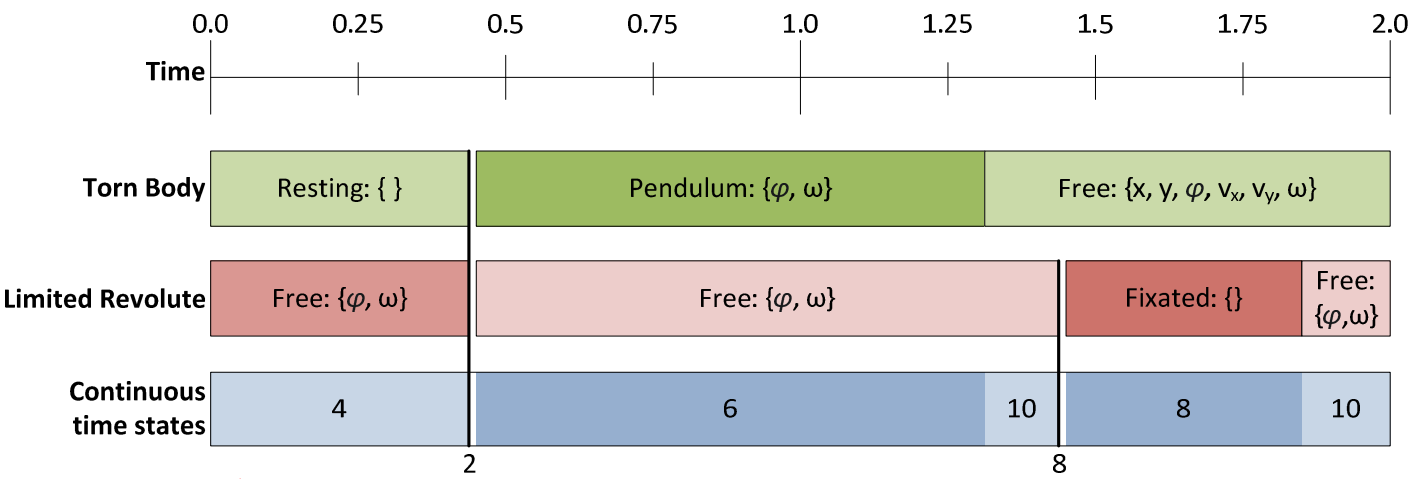
- The plots display the result of the simulation for the first 1.5 seconds.
- Forward Euler was used for time-integration (fixed step size: 1ms).
- The simulation of the whole system was performed within one second on a common PC.

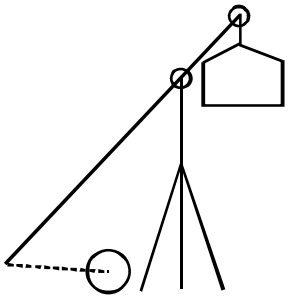
Simulation Result: Modes



Simulation Result: Modes

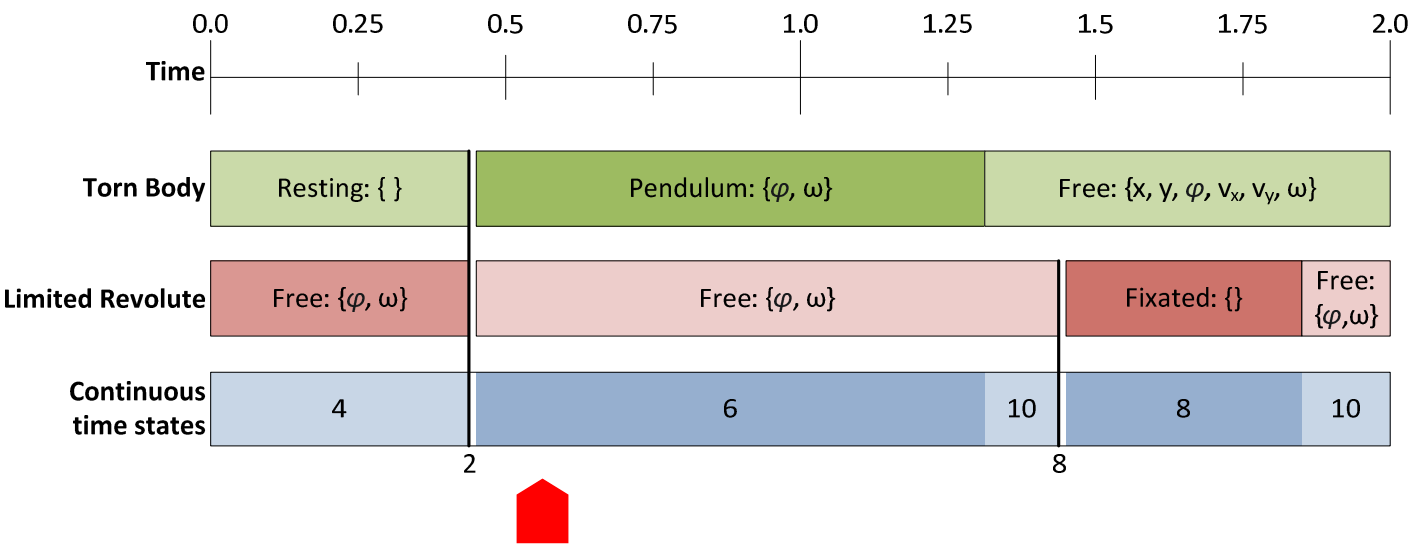
ETH Zürich
Department of Computer Science

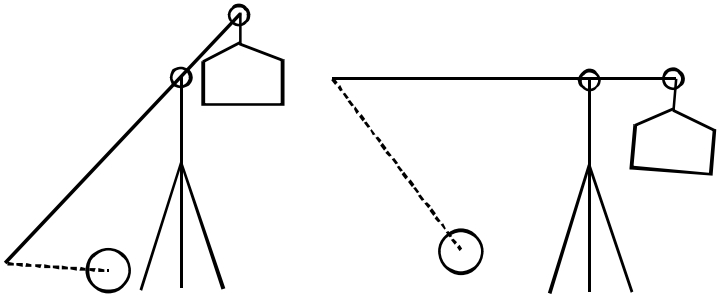




Simulation Result: Modes

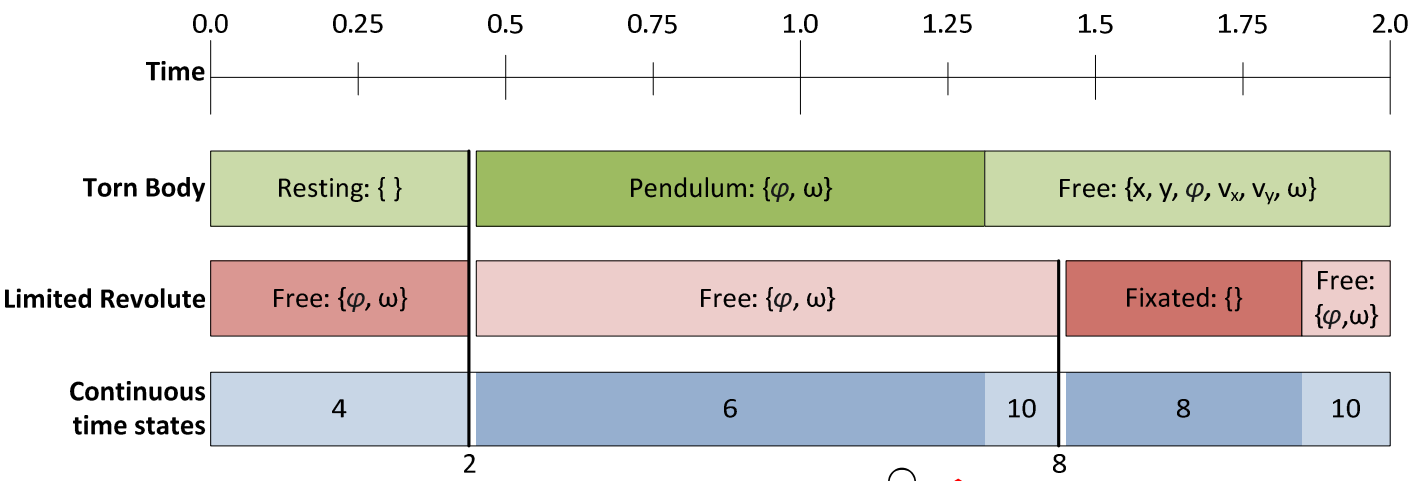
ETH Zürich
Department of Computer Science

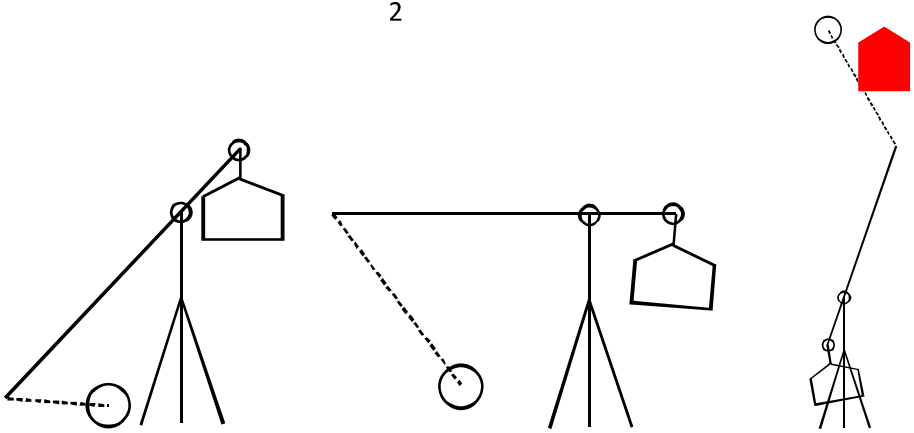




Simulation Result: Modes

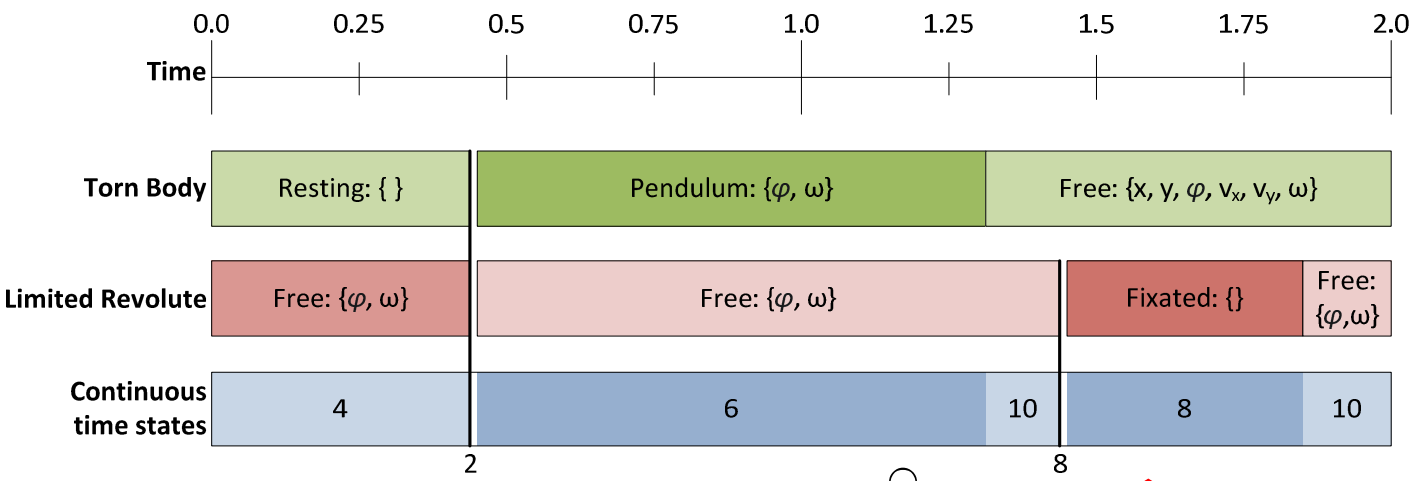
ETH Zürich
Department of Computer Science

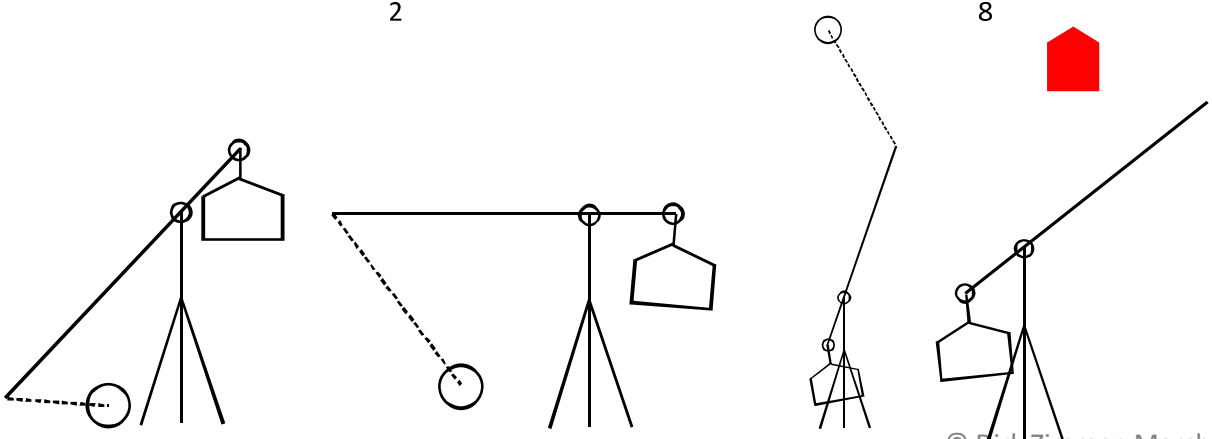




Simulation Result: Modes

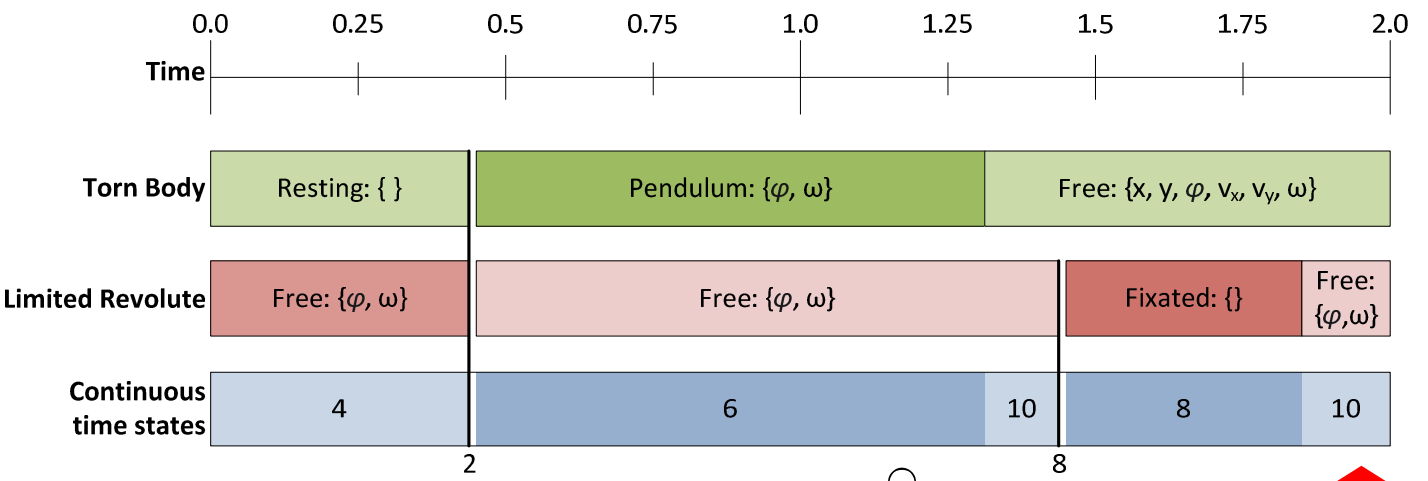
ETH Zürich
Department of Computer Science

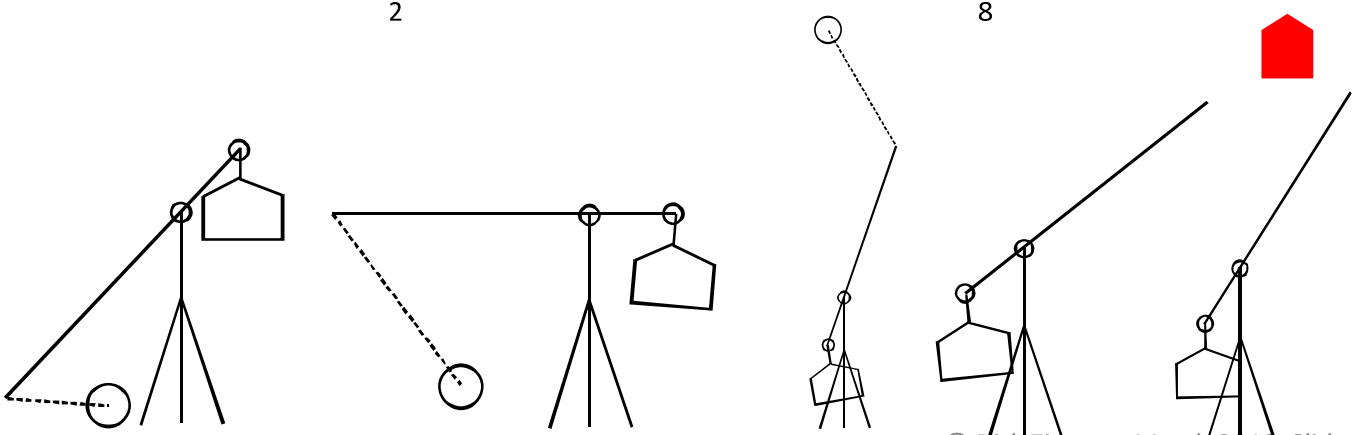




Simulation Result: Modes

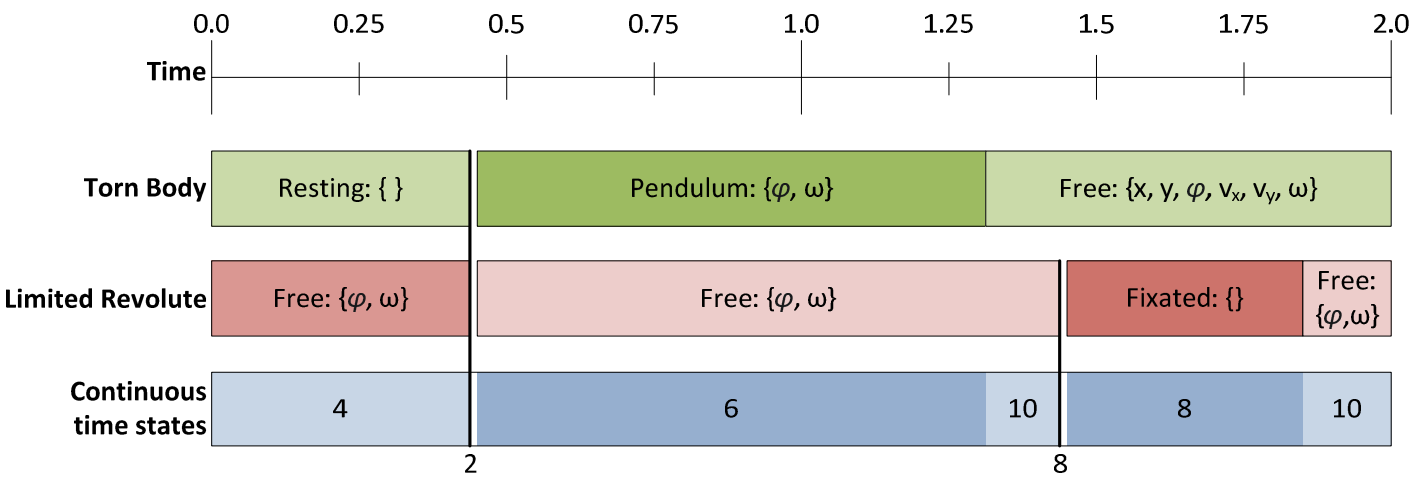
ETH Zürich
Department of Computer Science





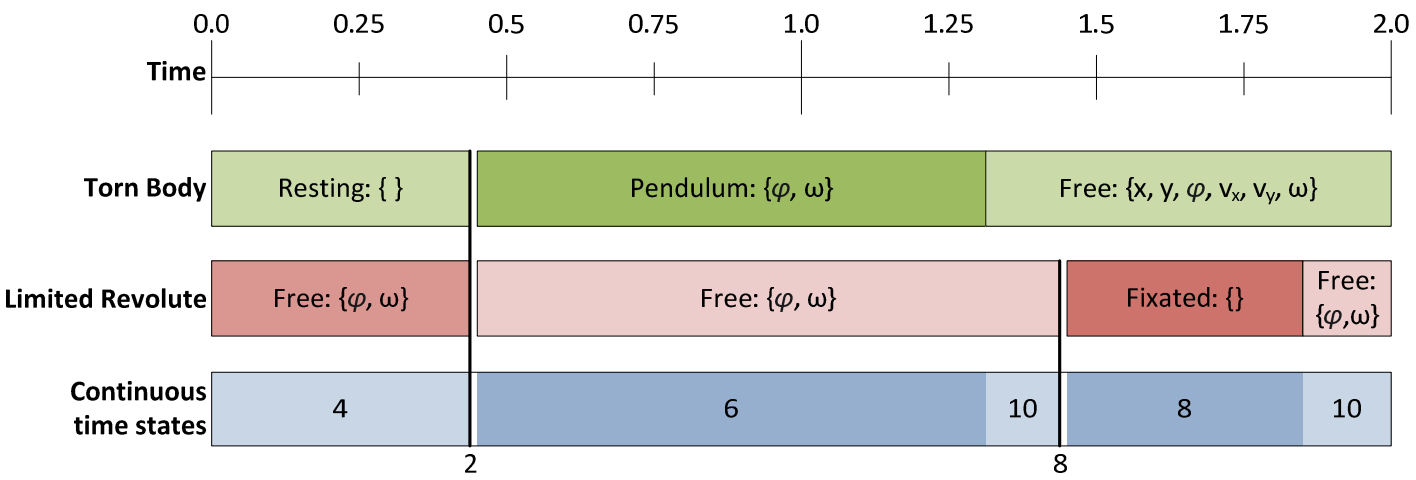
Simulation Result: Modes

ETH Zürich
Department of Computer Science



- The combination of modes of the components forms the modes of the complete system.
- In total, there occur 5 modes where only 2 of them are equivalent.
- Furthermore, there are 2 intermediate modes for the inelastic impulses.

Simulation Result: Modes



- To support object-oriented modeling, the simulation engine must derive the modes of the total system.

- If the modeler would be forced to model all modes and their transition on the top level, the modeling would become extremely laborious. Furthermore, the resulting solution would hardly be reusable.

The Sol Project: Review

- The motivation of the **Sol** project is **twofold**:
- One, Sol is a **language experiment**. We want to explore the full power of a declarative modeling approach.
- Two, Sol shall offer a platform for the development of corresponding **technical solutions**. This concerns…
  - dynamic (re-)causalization
  - dynamic treatment of higher index problems
  - etc…
- **Sol is not a product!** We don't intend to throw another modeling language or dialect on the market. Sol is primarily a research tool.

Conclusions

- There are two major contribution resulting from the Sol Project:

- **One**, the Sol language demonstrates that the object-oriented modeling paradigm of Modelica can be **successfully extended to variable-structure** systems. The power and expressiveness of Sol originates from the generalizations of successful Modelica concepts and not from the introduction of new paradigms.

  This may help future concerns in language design.

- **Two**, the simulator Solsim contains a **dynamic DAE processor** that can handle arbitrary changes in the set of equations and is able to cope with higher index systems.

  These techniques may be valuable for future, more dynamic simulation engines.

The End