Language extensions to Simulate Variable Structured Systems in Modelica

John Tinnerholm[1] & Adrian Pop[1]

1) Department of Computer and Information Science, Linköping University, Sweden

LINKÖPING UNIVERSITY

EMBrACE OpenModelica LARGEDYN ELLIIT

Agenda

- Background and challenges
- Language extension for explicit variable structured systems
- Language extensions for implicit variable structured systems
- Performance and scaling
- Current proposal, draft + demo

Modeling Highly Dynamic systems using Modelica

- Handling of models that dramatically change during simulation
- Number of equations and variables changes
- Needs efficient
  - ✓Just-in-time compilation
  - ✓Symbolic manipulation
  - ✓Interpretation
  - ✓Caching

Previous approaches

- Recent theses
  - Equation-based modeling of variable-structure systems[1]
  - First-class models: On a noncausal language for higher-order and structurally dynamic modelling and simulation, Höger[2]
  - Compiling Modelica : about the separate translation of models from Modelica to OCaml and its impact on variable-structure modeling[3]

- Techniques
  - ✓ Interpretation
  - ✓ DSL, embedded language
  - ✓ Focus on demonstrating techniques
  - ✓ Formal language specification & focus on formal semantics, not performance
  - ✓ Useful theoretical contributions
- ➢ Not standard compliant
- ➢ Not Modelica "Compilers"
- ➢ Small models

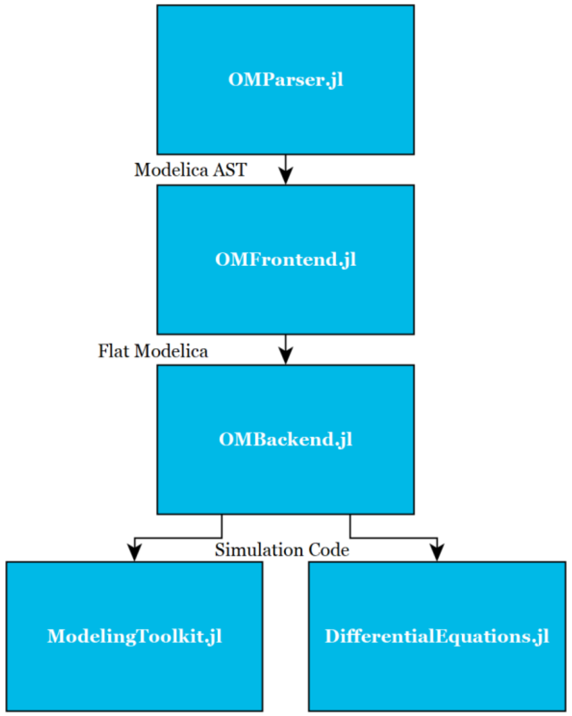
[1]Zimmer, Dirk (2010). "Equation-based modeling of variable-structure systems." PhD thesis. ETH Zürich.

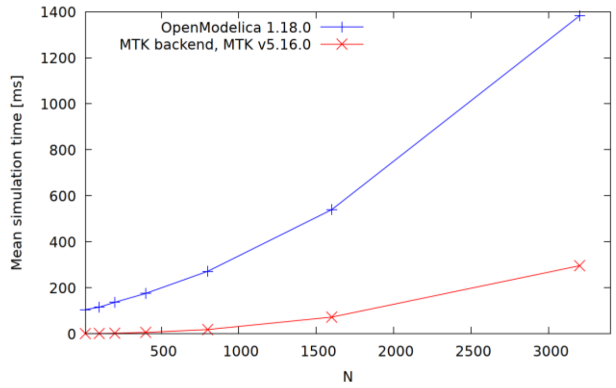
[2]Giorgidze, George (2012). "First-class models: On a noncausal language for higher-order and structurally dynamic modelling and simulation." PhD thesis. University of Nottingham.

[3]Höger, Christoph (2019). "Compiling Modelica : about the separate translation of models from Modelica to OCaml and its impact on variable-structure modeling." Doctoral Thesis. Berlin: Technische Universität Berlin. doi: 10.14279/depositonce-8354. url: http://dx.doi.org/

OpenModelica.jl

- How do we achieve standard compatibility?
    - Translating the High performance OpenModelica frontend into Julia
- A Modelica compiler in Julia
- SciML ecosystem
    - ModelingToolkit.jl (MTK)
    - DifferentialEquations.jl
    - Scientific machine learning (SCiML)
- Composable framework
    - Library interchange
    - Easily extendable
    - …





Mean simulation time when using the Tsit5 solver option of ModelingToolkit. The MTK backend manages to beat the OMC when using the TSIT5 solver[1]

[1]For OpenModelica IDA with DAE-mode was used. At the time of the experiment TSIT5 was not available in the OpenModelica backend and similar Runge-Kutta method was not supported for this particular problem.

Generating Flat Modelica

- Possible to generate flat Modelica
- Efficient implementation via MTK

```
connector Conn
  Real p "potential Variable";
  flow Real f "flow Variable";
end Conn;

partial model A
  Conn port;
end A;

partial model B
  extends A;
end B;

partial model C
  extends A;
end C;

model D
  extends B;
  extends C;
equation
port.f = port.p;
end D;

model E
  Conn port;
  D d;
equation
  connect(d.port, port);
end E;

model MultipleInheritanceConnect
  E e;
end MultipleInheritanceConnect;
```

Scripting in OpenModelica.jl

```
multipleinheritanceconnect = (ConnectTests.MultipleInheritanceConnect
                              , "MultipleInheritanceConnect"
                              , "./Connectors/MutipleInheritanceConnect.mo")
flatModelica = flattenFM("MultipleInheritanceConnect",
                         "./Connectors/MutipleInheritanceConnect.mo")
res = OMFrontend.toString(first(flatModelica))
@test res == ConnectTests.MultipleInheritanceConnect
```

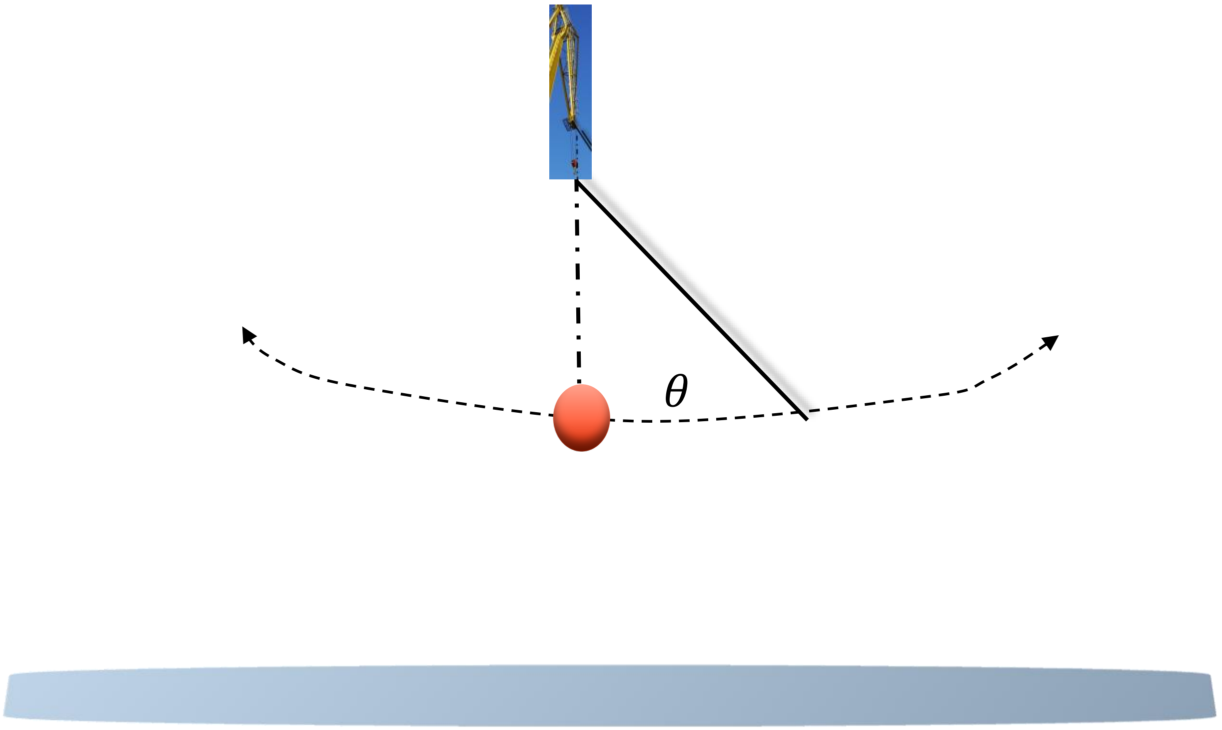
Result

```
class MultipleInheritanceConnect
  Real e.port.p;
  flow Real e.port.f;
  Real e.d.port.p;
  flow Real e.d.port.f;
equation
  e.port.p = e.d.port.p;
  e.port.f = 0.0;
  e.d.port.f - e.port.f = 0.0;
  e.d.port.f = e.d.port.p;
end MultipleInheritanceConnect;
```


[1]This example is based on the following test in the OpenModelica testsuite https://github.com/OpenModelica/OpenModelica/blob/master/testsuite/flattening/modelica/connectors/MultipleInheritanceConnect.mo

Modelling highly dynamic systems

- Handling of models that dramatically change during simulation
- Number of equations and variables changes
- Needs efficient
  - Just-in-time compilation
  - Symbolic manipulation
  - Interpretation
  - Caching



Language extensions for explicit Variable Structured Systems

Explicit variable structured systems

- Bounded number of variables/equations

Modelica needs:

- Syntax and semantics to capture changes in the equations and variables during simulation

Solution

- Inspiration from existing state machine syntax
  - *“Continuous state machines”*

New keywords

- **initialStructuralState**
- **structuralTransistion**

Restriction

- **The set of common variables**

```
model SimpleTwoModes
  model Single
    parameter Real a = 1.0;
    Real x (start = 1.0);
  equation
    der(x) = 2 * x + a;
  end Single;
  model HybridSingle
    parameter Real a = 1.0;
    Real x (start = 0.0);
  equation
    der(x) = x  - a;
  end HybridSingle;
structuralmode Single firstMode;
structuralmode HybridSingle secondMode;
equation
  /* We start in this first mode */
  initialStructuralState(firstMode);
  structuralTransistion(firstMode, secondMode, time
 >= 0.7);
end SimpleTwoModes;
```

Representing the breaking pendulum

- *Possible* in standard Modelica
  - Requires manual intervention by the modeler
  - Complex models…
- Extensions using statemachines
- Advantages
  - **Visual representation is obvious**
    - Statecharts…
  - Minor extension to the flat Modelica representation
  - Compilation, can be done ahead of time
- Disadvantages
  - The total number of variables and equations is bounded
  - Boilerplate
  - Causal representation

```
package BreakingPendulumExperiment
model FreeFall
  Real x;
  Real y;
  Real vx;
  Real vy;
  parameter Real g = 9.81;
  parameter Real vx0 = 0.0;
equation
  der(x) = vx;
  der(y) = vy;
  der(vx) = vx0;
  der(vy) = -g;
end FreeFall;

model Pendulum
  parameter Real x0 = 10;
  parameter Real y0 = 10;
  parameter Real g = 9.81;
  parameter Real L = sqrt(x0^2 + y0^2);
/* Common variables */
  Real x(start = x0);
  Real y(start = y0);
  Real vx;
  Real vy;
/* Model specific variables */
  Real phi(start = 1., fixed = true);
  Real phid;
equation
  der(phi) = phid;
  der(x) = vx;
  der(y) = vy;
  x = L * sin(phi);
  y = -L * cos(phi);
  der(phid) = -g / L * sin(phi);
end Pendulum;

model BreakingPendulum
  parameter Boolean breaks=false;
  FreeFall ff if breaks;
  Pendulum p if not breaks;
end BreakingPendulum;
end BreakingPendulumExperiment;
```

```
model BreakingPendulum
model FreeFall
  Real x;
  Real y;
  Real vx;
  Real vy;
  parameter Real g = 9.81;
  parameter Real vx0 = 0.0;
equation
  der(x) = vx;
  der(y) = vy;
  der(vx) = vx0;
  der(vy) = -g;
end FreeFall;
model Pendulum
  parameter Real x0 = 10;
  parameter Real y0 = 10;
  parameter Real g = 9.81;
  parameter Real L = sqrt(x0^2 + y0^2);
  /* Common variables */
  Real x(start = x0);
  Real y(start = y0);
  Real vx;
  Real vy;
 /* Model specific variables */
  Real phi(start = 1., fixed = true);
  Real phid;
equation
  der(phi) = phid;
  der(x) = vx;
  der(y) = vy;
  x = L * sin(phi);
  y = -L * cos(phi);
  der(phid) =  -g / L * sin(phi);
end Pendulum;
structuralmode Pendulum pendulum;
structuralmode FreeFall freeFall;
equation
  initialStructuralState(pendulum);
  structuralTransistion(pendulum,
                        freeFall,
                        time - 5.0 <= 0);
end BreakingPendulum;
```

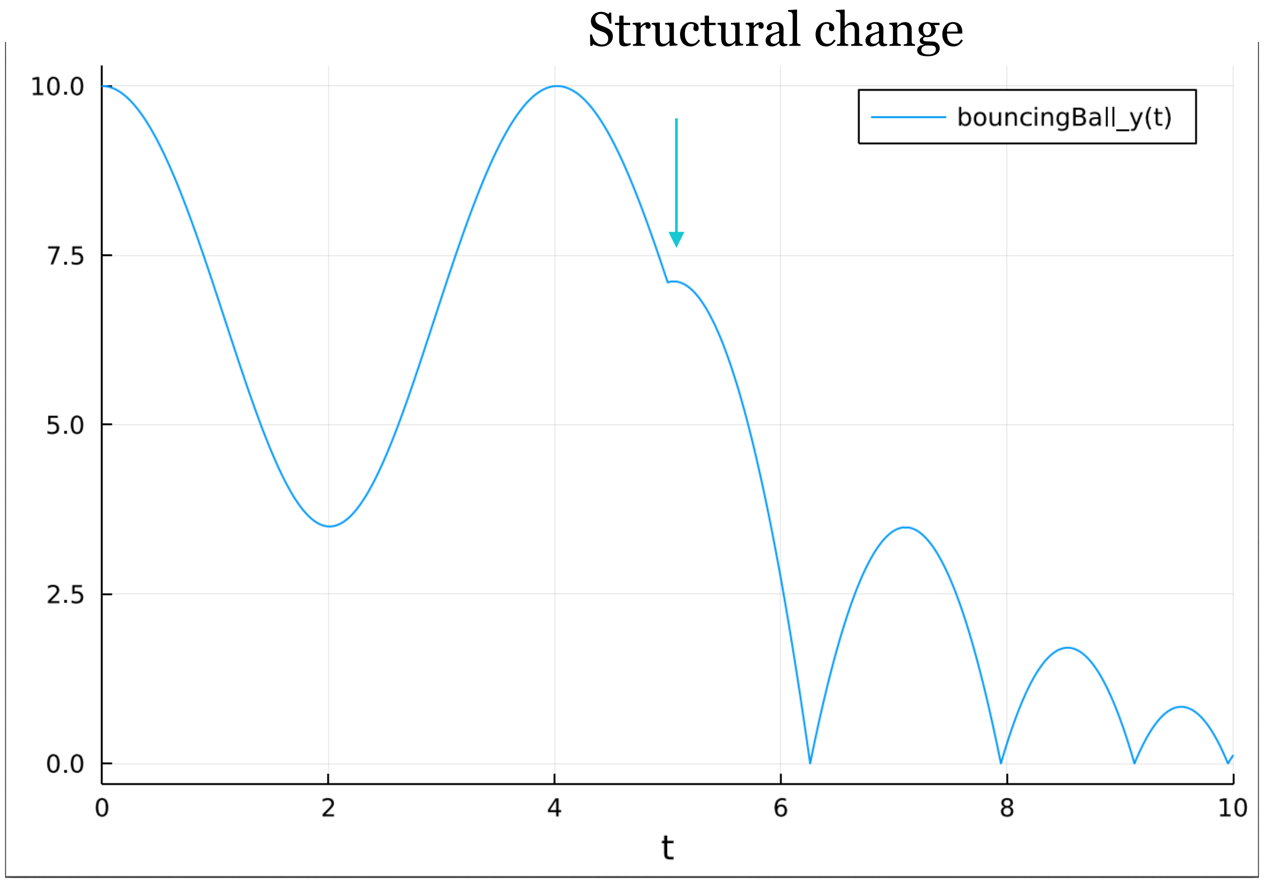
Modifying the flat Modelica representation

- The flat model is extended with a list of flat models.
- That is the flat model may itself contain other flat models and so on…
- Each flat model is compiled in separation before code generation

```
struct FLAT_MODEL <: FlatModel
  name::String
  variables::List{Variable}
  equations::List{Equation}
  initialEquations::List{Equation}
  algorithms::List{Algorithm}
  initialAlgorithms::List{Algorithm}
  #= VSS Modelica extension =#
  structuralSubmodels::List{FlatModel}
  scodeProgram::Option{SCode.CLASS}
  #= End VSS Modelica extension =#
  comment::Option{SCode.Comment}
end
```

Simulating the breaking pendulum

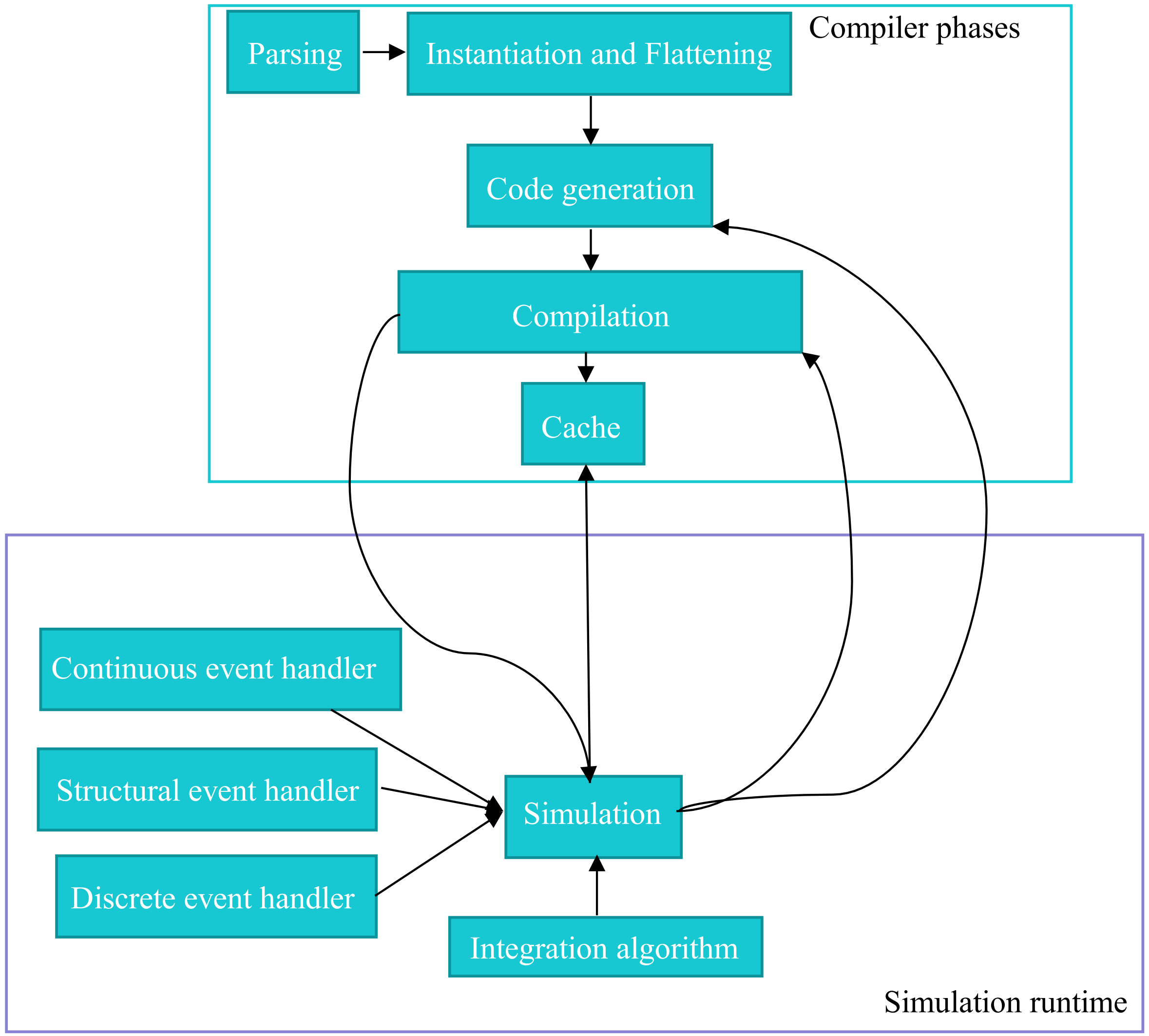
```
model BreakingPendulum

model BouncingBall
  parameter Real e=0.7;
  parameter Real g=9.81;
  Real x;
  Real y(start = 1.0);
  Real vx;
  Real vy;
equation
  der(x) = vx;
  der(y) = vy;
  der(vy) = -g;
  der(vx) = 0.0;
  when y <= 0 then
    reinit(vy, -e*pre(vy));
  end when;
end BouncingBall;

model Pendulum
  parameter Real x0 = 10;
  parameter Real y0 = 10;
  parameter Real g = 9.81;
  parameter Real L = sqrt(x0^2 + y0^2);
  /* Common variables */
  Real x(start = x0);
  Real y(start = y0);
  Real vx;
  Real vy;
 /* Model specific variables */
  Real phi(start = 1., fixed = true);
  Real phid;
equation
  der(phi) = phid;
  der(x) = vx;
  der(y) = vy;
  x = L * sin(phi);
  y = -L * cos(phi);
  der(phid) =  -g / L * sin(phi);
end Pendulum;
structuralmode Pendulum pendulum;
structuralmode BouncingBall bouncingBall;
/* Required? */
equation
  initialStructuralState(pendulum);
  structuralTransistion(pendulum, bouncingBall, time - 5.0 <= 0);
end BreakingPendulum;
```



In this model the free fall model is replaced with a bouncing ball model instead. That is when the pendulum breaks the model behaves like a bouncing ball. The graph show the change in height (y).

Implicit Variable Structure Systems

- With the explicit approach the user need to specify each state/change explicitly
- Enable compiling during simulation
  - Just in time compilation
  - Simulation *might* trigger recompilation
- The ***recompilation*** keyword
  - Triggers a recompilation during an event
  - Allows adjustments of the parameters of the model when a structural change occurs

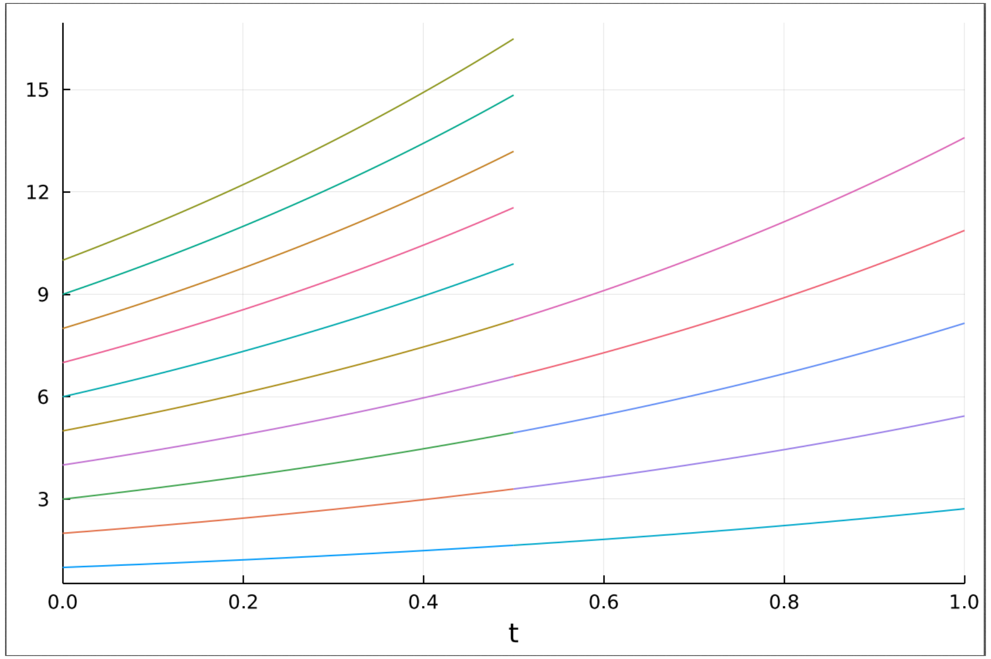


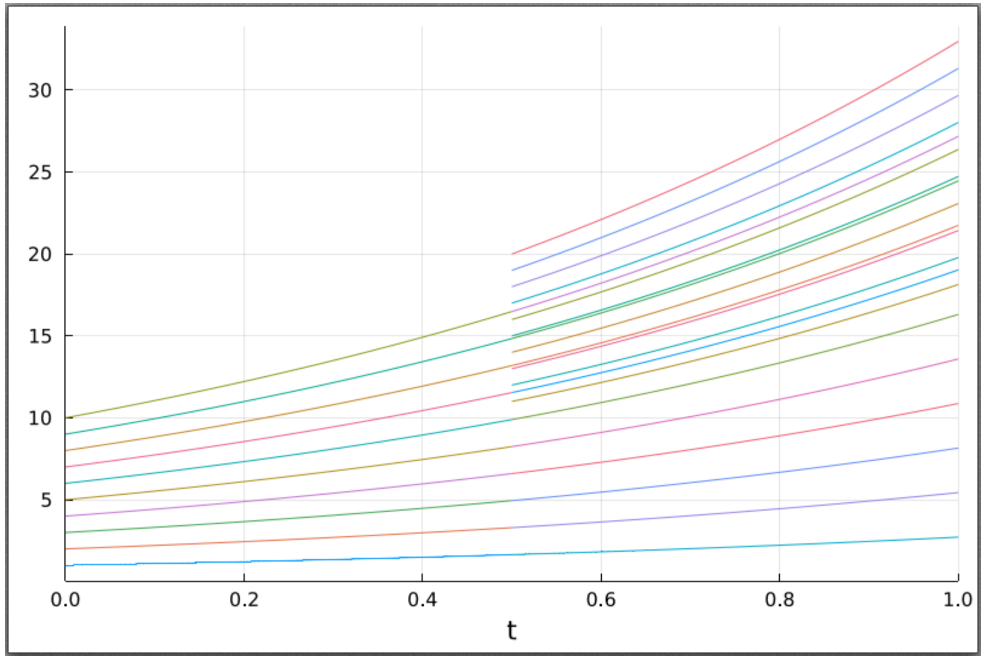
Example: ArrayGrow and ArrayShrink

```
/*
  This is an example of a model with structural variability
  We initially start with 10 equations, however during the simulation
  the ammount of equations are increased by 10.
*/
model ArrayGrow
   parameter Integer N = 10;
   Real x[N](start = {i for i in 1:N});
equation
  for i in 1:N loop
    x[i] = der(x[i]);
  end for;
  when time > 0.5 then
    /*
      Recompilation with change of parameters.
      the name of this function is the subject of change.
      What is changed depends on the argument passed to this function.
    */
    recompilation(N /*What we are changing*/, 20 /*The Value of the change*/);
  end when;
end ArrayGrow;
```

```
/*
  This is an example of a model with structural variability
  We initially start with 10 equations, however during the simulation
  the ammount of equations are increased by 10.
*/
model ArrayShrink
   parameter Integer N = 10;
   Real x[N](start = {i for i in 1:N});
equation
  for i in 1:N loop
    x[i] = der(x[i]);
  end for;
  when time > 0.5 then
    /*
      Recompilation with change of parameters.
      the name of this function is the subject of change.
      What is changed depends on the argument passed to this function.
    */
    recompilation(N /*What we are changing*/, 5 /*The Value of the change*/);
  end when;
end ArrayShrink;
```

Simulating ArrayGrow and ArrayShrink

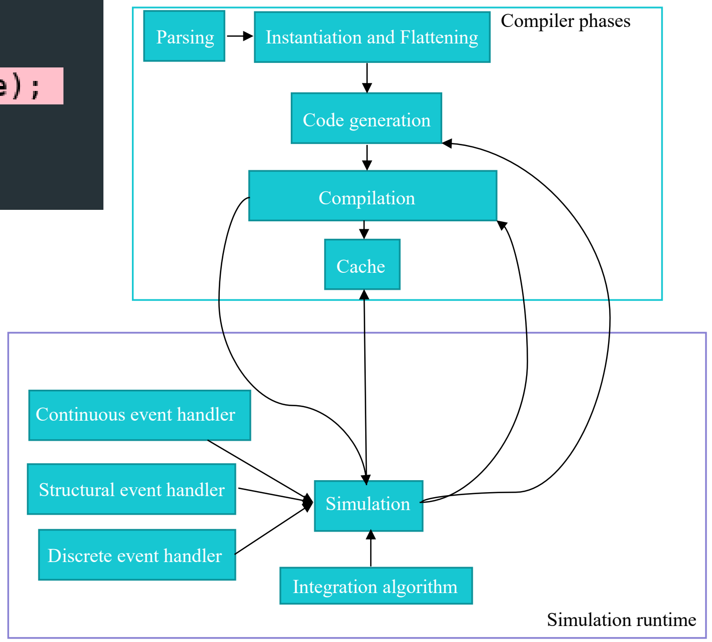




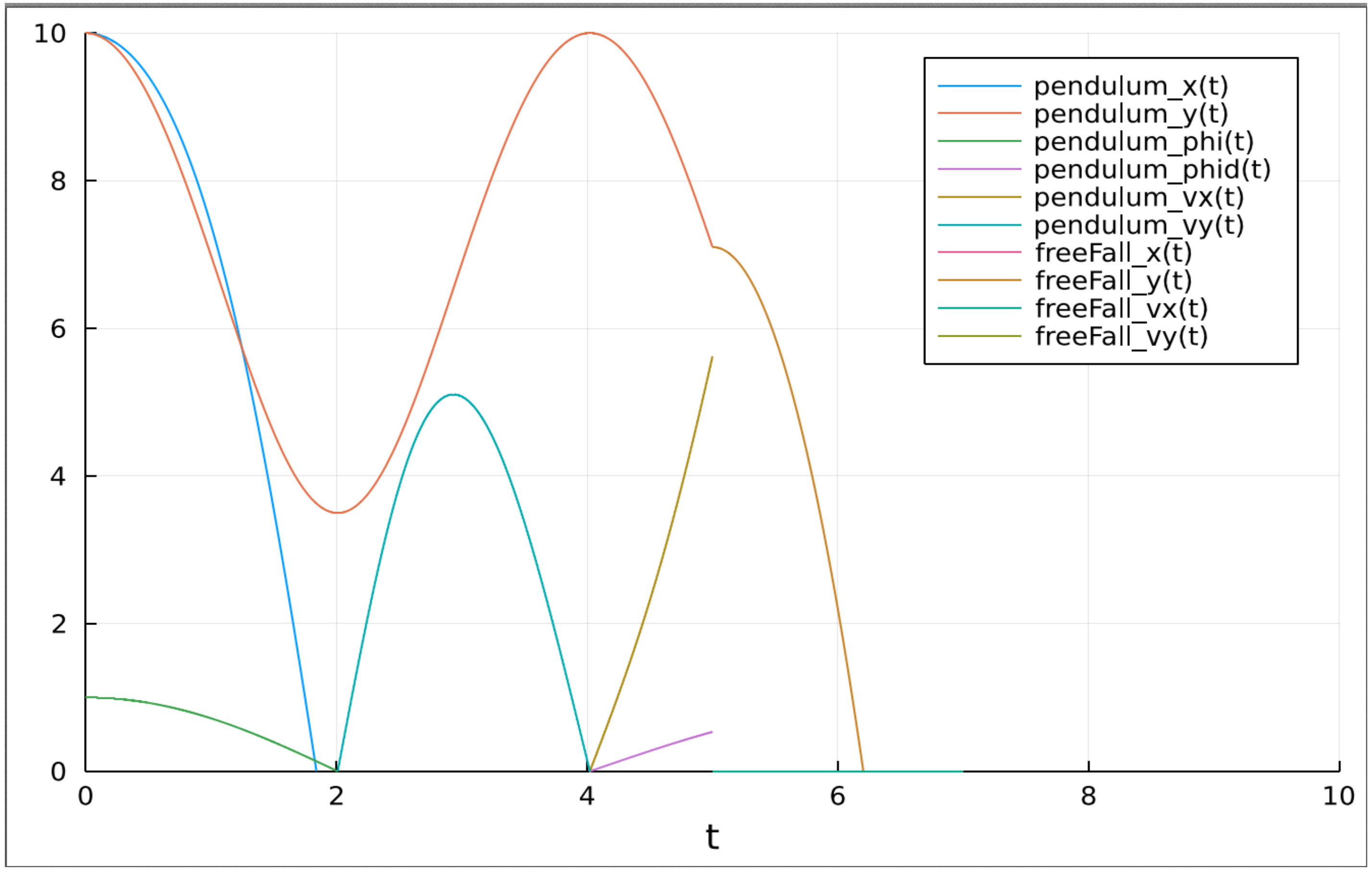
The breaking pendulum revisited compilation during simulation

- Change the conditional component during simulation
- Enables a variable set of:
    - Number of variables
    - Number of equations
    - Number of components
    - ....
- Minor change in syntax
    - Combine with conditional components
- Compilation during simulation

```
  parameter Boolean breaks = false;
  FreeFall freeFall if breaks;
  Pendulum pendulum if not breaks;
equation
  when 5.0 <= time then
    recompilation(breaks, true);
  end when;
end BreakingPendulum;
```



The breaking pendulum revisited compilation during simulation



What does this cost?

- Explicit VSS
  - Minor costs
    - Restart integration
    - Mapping variables
    - …
- Implicit
  - Currently, requires recompiling the entire model
  - Optimization possible
- Compiling to machine code + machine code optimization by LLVM is expensive

Some initial numbers for the breaking pendulum

- Compiling to machine code + machine code optimization by LLVM is expensive
- Recompilation step expensive but only a small part

Generating FlatModelica

0.033579 seconds (55.00 k allocations: 3.002 MiB)

Generating backend code

0.010235 seconds (9.87 k allocations: 485.242 KiB, 0.00% compilation time)

Recompiling the model due to the structural change

0.163508 seconds (330.05 k allocations: 19.279 MiB, 75.93% compilation time)

Compiling to LLVM + Simulating the model

4.535383 seconds (11.39 M allocations: 747.197 MiB, 4.32% gc time, 98.48% compilation time)

[1]Numbers generated by Julias buildin profiler.

What about larger models?

```
/*
  This is an example of a model with structural variability
  We initially start with 10 equations, however during the simulation
  the ammount of equations are increased by 10.
*/
model ArrayGrow
   parameter Integer N = 10;
   Real x[N](start = {i for i in 1:N});
equation
  for i in 1:N loop
    x[i] = der(x[i]);
  end for;
  when time > 0.5 then
    /*
      Recompilation with change of parameters.
      the name of this function is the subject of change.
      What is changed depends on the argument passed to this function.
    */
    recompilation(N /*What we are changing*/,
                  20 /*The Value of the change*/);
  end when;
end ArrayGrow;
```

What about larger models?

- Increase the change from 100 to 200 variables + equations
  - **Generating FlatModelica**
    - **0.038404 seconds (87.40 k allocations: 3.540 MiB, 0.70% compilation time)**
  - Generating backend code
  - 0.092079 seconds (247.91 k allocations: 11.983 MiB, 0.01% compilation time)
  - Compiling to LLVM + Simulating the model
    - Recompiling the model due to the structural change
      - 3.390860 seconds (2.37 M allocations: 121.405 MiB, 90.43% compilation time)
  - 11.580225 seconds (14.56 M allocations: 976.074 MiB, 1.67% gc time, 94.11% compilation time)
- Increase the change from 200 to 250 variables + equations
  - Generating FlatModelica
    - **0.056116 seconds (154.05 k allocations: 5.902 MiB)**
  - Generating backend code
    - 0.212804 seconds (809.03 k allocations: 38.524 MiB, 0.01% compilation time)
  - Compiling to LLVM + Simulating the model
    - Recompiling the model due to the structural change
      - 13.841762 seconds (6.11 M allocations: 316.432 MiB, 0.41% gc time, 93.73% compilation time)
  - 29.652287 seconds (19.70 M allocations: 1.255 GiB, 1.25% gc time, 91.27% compilation time)

Not scalarizing arrays is the key.
Memory is a bottleneck!

Conclusion

- Support for bounded VSS does not require Just-in-time compilation
- VSS support can be added with minimal modifcation to existing syntax
- Requirement on tools
  - Explicit VSS requires separate flattening and tight solver integration
  - Implicit VSS requires Just-in-time compilation
    - The simulation need to call the compiler during simulation…
- Performance improvements are possible

Future work

- New translator written in Julia
- Support for more Modelica constructs in the backend
- Higher coverage for the MSL in the frontend
- Efficient Just-in-time compilation
  - Compilation to machine code is expensive
  - **Not scalarizing arrays is the key**
  - **Incremental/Separate compilation**
- Calculate the impact and minimize the ammount of new code generated for the structural change?
  - Abysmal improvements
  - …

This Photo by Unknown Author is licensed under CC BY

Thank you for your attention

Questions?