Efficient Handling of Arrays in the New Backend

Karim Abdelhak, Bernhard Bachmann

University of Applied Sciences Bielefeld
Bielefeld, Germany

FH Bielefeld
University of Applied Sciences
1 Introduction

2 New Backend

3 Set-Based Graphs

4 Summary
1. Introduction
## Collaborators

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karim Abdelhak</td>
<td>FH Bielefeld</td>
<td>Implementation, Design, Graph Theory</td>
</tr>
<tr>
<td>Ernesto Kofman</td>
<td>CIFASIS Rosario</td>
<td>Graph Theory</td>
</tr>
<tr>
<td>Philip Hannebohm</td>
<td>FH Bielefeld</td>
<td>Implementation, Design</td>
</tr>
<tr>
<td>Andreas Heuermann</td>
<td>Lingköping University</td>
<td>Implementation, Design</td>
</tr>
<tr>
<td>Bernhard Bachmann</td>
<td>FH Bielefeld</td>
<td>Design, Graph Theory</td>
</tr>
<tr>
<td>Per Östlund</td>
<td>Lingköping University</td>
<td>Util, New Frontend</td>
</tr>
</tbody>
</table>
Collaborators

Karim Abdelhak  
*FH Bielefeld* 
Implementation, Design, Graph Theory

Ernesto Kofman  
*CIFASIS Rosario* 
Graph Theory

Philip Hannebohm  
*FH Bielefeld* 
Implementation, Design

Andreas Heuermann  
*Lingköping University* 
Implementation, Design

Bernhard Bachmann  
*FH Bielefeld* 
Design, Graph Theory

Per Östlund  
*Lingköping University* 
Util, New Frontend
## Collaborators

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karim Abdelhak</td>
<td>FH Bielefeld</td>
<td>Implementation, Design, Graph Theory</td>
</tr>
<tr>
<td>Ernesto Kofman</td>
<td>CIFASIS Rosario</td>
<td>Graph Theory</td>
</tr>
<tr>
<td>Philip Hannebohm</td>
<td>FH Bielefeld</td>
<td>Implementation, Design</td>
</tr>
<tr>
<td>Andreas Heuermann</td>
<td>Lingköping University</td>
<td>Implementation, Design</td>
</tr>
<tr>
<td>Bernhard Bachmann</td>
<td>FH Bielefeld</td>
<td>Design, Graph Theory</td>
</tr>
<tr>
<td>Per Östlund</td>
<td>Lingköping University</td>
<td>Util, New Frontend</td>
</tr>
</tbody>
</table>
Collaborators

Karim Abdelhak, Bernhard Bachmann

Implementation, Design, Graph Theory

Ernesto Kofman

Graph Theory

Philip Hannebohm

Implementation, Design

Andreas Heuermann

Implementation, Design

Per Östlund

Util, New Frontend
Collaborators

Karim Abdelhak  
*FH Bielefeld*  
Implementation, Design, Graph Theory

Ernesto Kofman  
*CIFASIS Rosario*  
Graph Theory

Philip Hannebohm  
*FH Bielefeld*  
Implementation, Design

Andreas Heuermann  
*Lingköping University*  
Implementation, Design

Bernhard Bachmann  
*FH Bielefeld*  
Design, Graph Theory

Per Östlund  
*Lingköping University*  
Util, New Frontend
## Collaborators

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karim Abdelhak</td>
<td>FH Bielefeld</td>
<td>Implementation, Design, Graph Theory</td>
</tr>
<tr>
<td>Ernesto Kofman</td>
<td>CIFASIS Rosario</td>
<td>Graph Theory</td>
</tr>
<tr>
<td>Philip Hannebohm</td>
<td>FH Bielefeld</td>
<td>Implementation, Design</td>
</tr>
<tr>
<td>Andreas Heuermann</td>
<td>Lingköping University</td>
<td>Implementation, Design</td>
</tr>
<tr>
<td>Bernhard Bachmann</td>
<td>FH Bielefeld</td>
<td>Design, Graph Theory</td>
</tr>
<tr>
<td>Per Östlund</td>
<td>Lingköping University</td>
<td>Util, New Frontend</td>
</tr>
</tbody>
</table>

February 2, 2021
Primary Goal
Support unscalarized array processing.

Secondary Goals
- better high level performance due to reworked graph theory,
- better low level performance by avoiding fail-based processing,
- higher information consistency by only creating one instance of each variable and equation,
- higher maintainability due to stricter module interfaces and pseudo member functions.
Introduction

Overview

Primary Goal
Support unscalarized array processing.

Secondary Goals
- Better high level performance due to reworked graph theory,
- Better low level performance by avoiding fail-based processing,
- Higher information consistency by only creating one instance of each variable and equation,
- Higher maintainability due to stricter module interfaces and pseudo member functions.
**Primary Goal**

Support unscalarized array processing.

**Secondary Goals**

- better high level performance due to reworked graph theory,
- better low level performance by avoiding fail-based processing,
- higher information consistency by only creating one instance of each variable and equation,
- higher maintainability due to stricter module interfaces and pseudo member functions.
Overview

Primary Goal
Support unscalarized array processing.

Secondary Goals
- better high level performance due to reworked graph theory,
- better low level performance by avoiding fail-based processing,
- higher information consistency by only creating one instance of each variable and equation,
- higher maintainability due to stricter module interfaces and pseudo member functions.
Overview

Primary Goal
Support unscalarized array processing.

Secondary Goals
- better high level performance due to reworked graph theory,
- better low level performance by avoiding fail-based processing,
- higher information consistency by only creating one instance of each variable and equation,
- higher maintainability due to stricter module interfaces and pseudo member functions.
Overview

Primary Goal
Support unscalarized array processing.

Secondary Goals
- better high level performance due to reworked graph theory,
- better low level performance by avoiding fail-based processing,
- higher information consistency by only creating one instance of each variable and equation,
- higher maintainability due to stricter module interfaces and pseudo member functions.
2. New Backend
General Changes
Coding Ethics

1. **encapsulated uniontypes** with pseudo member functions,

2. use `match` instead of `matchoncontinue`,

3. traverser functions with a maximum of one extra argument,

4. use `for`-loops instead of recursive traversing more often,

5. major modules (that inherit from `NBModule.mo`) need to have a full description or reference to source publication,

6. ...
General Changes

Coding Ethics

1. **encapsulated uniontypes** with **pseudo member functions**,  

2. **use** `match` **instead of** `matchcontine`,  

3. **traverser functions** with a **maximum of one extra argument**,  

4. **use** `for`-loops **instead of** recursive traversing more often,  

5. **major modules (that inherit from** `NBModule.mo`) **need to have a full description or reference to source publication**,  

6. ...
General Changes
Coding Ethics

1. encapsulated uniontypes with pseudo member functions,

2. use \texttt{match} instead of \texttt{match\_continue},

3. traverser functions with a maximum of one extra argument,

4. use \texttt{for}-loops instead of recursive traversing more often,

5. major modules (that inherit from \texttt{NBModule.mo}) need to have a full description or reference to source publication,

6. ...
General Changes
Coding Ethics

1. **encapsulated uniontypes** with pseudo member functions,

2. use **match** instead of **matchontinue**,

3. traverser functions with a maximum of one extra argument,

4. use **for-loops** instead of recursive traversing more often,

5. major modules (that inherit from **NBModule.mo**) need to have a full description or reference to source publication,

6. . . .
General Changes
Coding Ethics

1. **encapsulated uniontypes** with pseudo member functions,
2. use **match** instead of **matchcontinue**, 
3. **traverser** functions with a maximum of one extra argument, 
4. use **for**-loops instead of recursive traversing more often, 
5. major modules (that inherit from **NBModule.mo**) need to have a full description or reference to source publication, 

...
General Changes
Coding Ethics

1. **encapsulated union types** with pseudo member functions,
2. use `match` instead of `matchoncontinue`,
3. traverser functions with a maximum of one extra argument,
4. use `for`-loops instead of recursive traversing more often,
5. major modules (that inherit from `NBModule.mo`) need to have a full description or reference to source publication,
6. ...

Karim Abdelhak, Bernhard Bachmann
Structures in the New Backend

Module Dependencies

Lowering → Simplify → Events → Partitioning → Causalize → Initialization → Categorize → Tearing → Jacobian → SimCode

DetectStates → Alias

DAE-Mode

Main

Mandatory

Optional
Structures in the New Backend
Module Dependencies
Structures in the New Backend
Module Dependencies
Structures in the New Backend

Module Dependencies
Structures in the New Backend
Module Dependencies
Structures in the New Backend
Main Structure

```plaintext
record MAIN
    list <System> ode
    list <System> alg
    list <System> ode_event
    list <System> alg_event
    list <System> init
    Option<list <System>> init_0
    Option<list <System>> dae
    BVariable.VarData varData
    BEquation.EqData eqData
    Events.EventInfo eventInfo
    FunctionTree funcTree
end MAIN;
```

"Systems for ode equations";
"Systems for algebraic equations";
"Systems for ode event iteration";
"Systems for alg. event iteration";
"Systems for initialization";
"Systems for homotopy initialization";
"Systems for dae mode";
"Variable data.";
"Equation data.";
"contains time and state events";
"Function bodies.";
Structures in the New Backend

Variables

All variables only exist once and are called by reference using the `Pointer<>` structure.

```plaintext
record VAR_DATA_SIM
  VariablePointers variables;

  // subset of full variable array
  VariablePointers unknowns;
  VariablePointers knowns;
  VariablePointers initials;
  VariablePointers auxiliaries;
  VariablePointers aliasVars;

end VAR_DATA_SIM;
```
Structures in the New Backend

Variables

All variables only exist once and are called by reference using the `Pointer<>` structure.

```plaintext
record VAR_DATA_SIM
  VariablePointers variables;
  // subset of full variable array
  VariablePointers unknowns;
  VariablePointers knowns;
  VariablePointers initials;
  VariablePointers auxiliaries;
  VariablePointers aliasVars;

/ *** /
end VAR_DATA_SIM;
```

Karim Abdelhak, Bernhard Bachmann
Structures in the New Backend

Equation

All equations only exist once and are called by reference using the Pointer<> structure. Furthermore each equation has a unique identifier variable which is the former residual variable. $RES_i$

```plaintext
record EQ_DATA_SIM
    Pointer<Integer> uniqueIndex;
    EquationPointers equations;
    EquationPointers simulation;
    EquationPointers continuous;
    EquationPointers discretes;
    EquationPointers initials;
    EquationPointers auxiliaries;
    EquationPointers removed;
end EQ_DATA_SIM;
```
Equation

All equations only exist once and are called by reference using the `Pointer<>` structure. Furthermore each equation has a unique identifier variable which is the former residual variable. $RES_i$

```plaintext
record EQ_DATA_SIM
  Pointer<Integer> uniqueIndex;
  EquationPointers equations;
  EquationPointers simulation;
  EquationPointers continuous;
  EquationPointers discrete;
  EquationPointers initials;
  EquationPointers auxiliaries;
  EquationPointers removed;
end EQ_DATA_SIM;
```
Structures in the New Backend

Pointer Arrays

The arrays contain pointers to variables or equations instead of the instances themselves. An additional unordered map is provided to also always find the index for any cref (variable name or equation residual name).

```plaintext
record VARIABLE_POINTERS
    UnorderedMap map;
    ExpandableArray<Pointer<Variable>> varArr;
end VARIABLE_POINTERS;

record EQUATION_POINTERS
    UnorderedMap map;
    ExpandableArray<Pointer<Equation>> eqArr;
end EQUATION_POINTERS;
```
Structures in the New Backend

Pointer Arrays

The arrays contain pointers to variables or equations instead of the instances themselves. An additional unordered map is provided to also always find the index for any cref (variable name or equation residual name).

```plaintext
record VARIABLE_POINTERS
    UnorderedMap map;
    ExpandableArray<Pointer<Variable>> varArr;
end VARIABLE_POINTERS;

record EQUATION_POINTERS
    UnorderedMap map;
    ExpandableArray<Pointer<Equation>> eqArr;
end EQUATION_POINTERS;
```
Structures in the New Backend

Modules

Modules have strict interfaces they have to follow. Inside every module there is a wrapper function which takes the full system and applies this restricted body function to it.

```plaintext
partial function wrapper
  input output BackendDAE bdae;
end wrapper;

partial function tearingInterface
  input output StrongComponent comp
  input output FunctionTree funcTree
  input output Integer index
  input System.SystemType systemType
end tearingInterface;
```

"the suspected algebraic loop";
"Function call bodies";
"current unique loop index";
"system type";
Structures in the New Backend

Modules

Modules have strict interfaces they have to follow. Inside every module there is a wrapper function which takes the full system and applies this restricted body function to it.

```plaintext
partial function wrapper
  input output BackendDAE bdae;
end wrapper;

partial function tearingInterface
  input output StrongComponent comp
  input output FunctionTree funcTree
  input output Integer index
  input System.SystemType systemType
end tearingInterface;
```

"the suspected algebraic loop";
"Function call bodies";
"current unique loop index";
"system type";
3. Set-Based Graphs
Array Processing
Set-Based Graphs

Theory by Ernesto Kofman (CIFASIS, Rosario Argentina)
Theory by Ernesto Kofman (CIFASIS, Rosario Argentina)

Definition 1 (Set–Vertex). A set–vertex is a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$.
Theory by Ernesto Kofman (CIFASIS, Rosario Argentina)

Definition 1 (Set–Vertex). A set–vertex is a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$.

Definition 2 (Set–Edge). Given two set–vertices, $V^a$ and $V^b$, with $V^a \cap V^b = \emptyset$, a set–edge connecting $V^a$ and $V^b$ is a set of non repeated edges $E[\{V^a, V^b\}] = \{e_1, e_2, \ldots, e_n\}$ where each edge is a tuple containing two vertices $e_i = \{v^a_k \in V^a, v^b_l \in V^b\}$. 
Array Processing
Set-Based Graphs

*Theory by Ernesto Kofman (CIFASIS, Rosario Argentina)*

**Definition 3 (Set–Based Graph).** A set–based graph is a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where

- $\mathcal{V} = \{V^1, \ldots, V^n\}$ is a set of disjoint set–vertices.
- $\mathcal{E} = \{E^1, \ldots, E^m\}$ is a set of set–edges connecting set–vertices of $\mathcal{V}$. In addition two different set–edges in $\mathcal{E}$ cannot connect the same set–vertices.
Definition 3 (Set–Based Graph). A set–based graph is a pair \( \mathcal{G} = (\mathcal{V}, \mathcal{E}) \) where
- \( \mathcal{V} = \{V^1, \ldots, V^n\} \) is a set of disjoint set–vertices.
- \( \mathcal{E} = \{E^1, \ldots, E^m\} \) is a set of set–edges connecting set–vertices of \( \mathcal{V} \). In addition two different set–edges in \( \mathcal{E} \) cannot connect the same set–vertices.
**Theory by Ernesto Kofman (CIFASIS, Rosario Argentina)**

**Definition 3 (Set–Based Graph).** A set–based graph is a pair $G = (\mathcal{V}, \mathcal{E})$ where

- $\mathcal{V} = \{V^1, \ldots, V^n\}$ is a set of disjoint set–vertices.
- $\mathcal{E} = \{E^1, \ldots, E^m\}$ is a set of set–edges connecting set–vertices of $\mathcal{V}$. In addition two different set–edges in $\mathcal{E}$ cannot connect the same set–vertices.
**Definition 4 (Bipartite Set–Based Graph).** A bipartite set–based graph is a set–based graph \( \mathcal{G} = (\mathcal{V}, \mathcal{E}) \) where two disjoints sets of set–vertices \( \mathcal{F}, \mathcal{U} \) can be found verifying \( \mathcal{F} \cup \mathcal{U} = \mathcal{V} \) and \( \mathcal{F} \cap \mathcal{U} = \emptyset \). Set-edges in \( \mathcal{E} \) can only connect set-vertices from \( \mathcal{F} \) with \( \mathcal{U} \).
**Theory by Ernesto Kofman (CIFASIS, Rosario Argentina)**

**Definition 4 (Bipartite Set–Based Graph).** A bipartite set–based graph is a set–based graph $G = (V, E)$ where two disjoints sets of set–vertices $F, U$ can be found verifying $F \cup U = V$ and $F \cap U = \emptyset$. Set-edges in $E$ can only connect set-vertices from $F$ with $U$. 
**Definition 4 (Bipartite Set–Based Graph).** A bipartite set–based graph is a set–based graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where two disjoints sets of set–vertices $\mathcal{F}, \mathcal{U}$ can be found verifying $\mathcal{F} \cup \mathcal{U} = \mathcal{V}$ and $\mathcal{F} \cap \mathcal{U} = \emptyset$. Set-edges in $\mathcal{E}$ can only connect set-vertices from $\mathcal{F}$ with $\mathcal{U}$.
Array Processing
Set-Based Matching

\[ \mathcal{F} \quad \mathcal{E} \quad \mathcal{U} \]
Array Processing
Set-Based Matching
Array Processing
Set-Based Matching
Array Processing
Set-Based Matching
Array Processing
Set-Based Matching
Array Processing
Set-Based Matching

Description

- set-edges saved in the form of linear maps
- computationally easy to find connected subsets
- computational complexity independent of array sizes
Array Processing
Set-Based Matching

Description

- set-edges saved in the form of linear maps
- computationally easy to find connected subsets
- computational complexity independent of array sizes
Array Processing
Set-Based Matching

Description

- set-edges saved in the form of linear maps
- computationally easy to find connected subsets
- computational complexity independent of array sizes
Summary

Already Implemented

- Basic scalarized processing for all presented modules (besides Partitioning and Tearing),
- Runs test models for each module and simple models from the MSL.

Current Development

- Unscalarized causalization with set-based-graphs,
- Record handling and better simplification,
- minimal Tearing.

Further Plans

- Unscalarized processing for other modules,
- CommonSubExpression / WrapFunctionCalls,
- Dynamic state selection.
Summary

Already Implemented

- Basic scalarized processing for all presented modules (besides Partitioning and Tearing),
- Runs test models for each module and simple models from the MSL.

Current Development

- Unscalarized causalization with set-based-graphs,
- Record handling and better simplification,
- minimal Tearing.

Further Plans

- Unscalarized processing for other modules,
- CommonSubExpression / WrapFunctionCalls,
- Dynamic state selection.
Summary

Already Implemented
- Basic scalarized processing for all presented modules (besides Partitioning and Tearing),
- Runs test models for each module and simple models from the MSL.

Current Development
- Unscalarized causalization with set-based-graphs,
- Record handling and better simplification,
- minimal Tearing.

Further Plans
- Unscalarized processing for other modules,
- CommonSubExpression / WrapFunctionCalls,
- Dynamic state selection.
Summary

Already Implemented
- Basic scalarized processing for all presented modules (besides Partitioning and Tearing),
- Runs test models for each module and simple models from the MSL.

Current Development
- Unscalarized causalization with set-based-graphs,
- Record handling and better simplification,
- minimal Tearing.

Further Plans
- Unscalarized processing for other modules,
- CommonSubExpression / WrapFunctionCalls,
- Dynamic state selection.
Thank you for your attention!