# 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

# 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

| | | |
|---|---|---|
| **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

| | | |
|---|---|---|
| **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

| | | |
|---|---|---|
| **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

| | | |
|---|---|---|
| **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

| | | |
|---|---|---|
| **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 |

# 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.

# 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 `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 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 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 `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 `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. ...

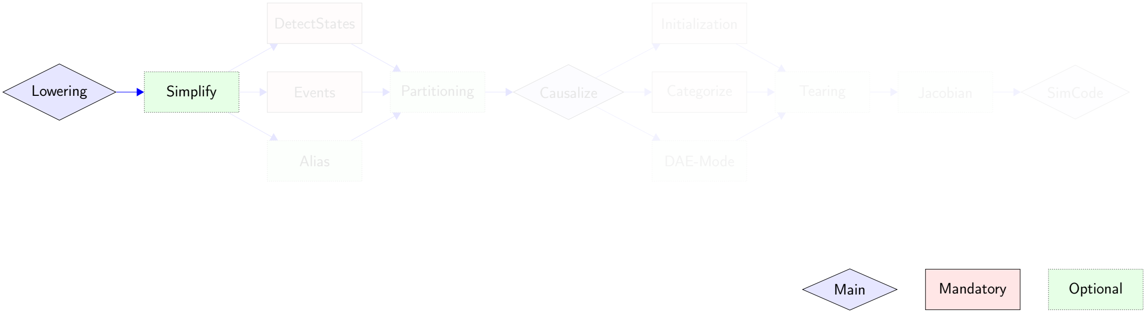
# General Changes

## Coding Ethics

1. `encapsulated uniontypes` with pseudo member functions,
2. use `match` instead of `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. ...

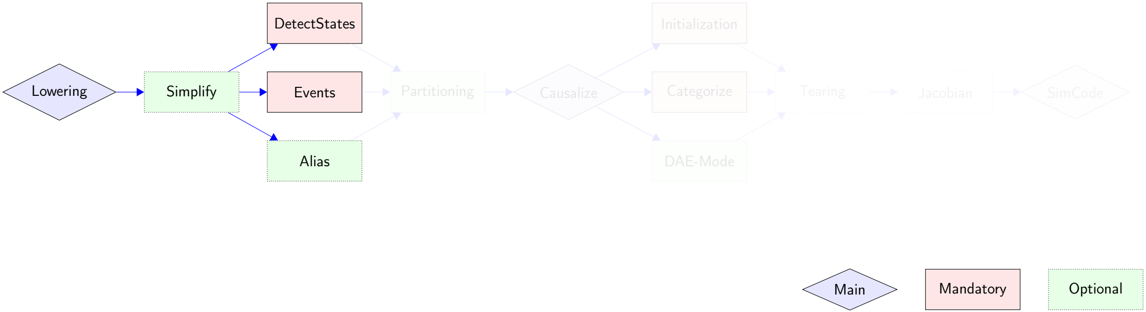
# Structures in the New Backend

## Module Dependencies



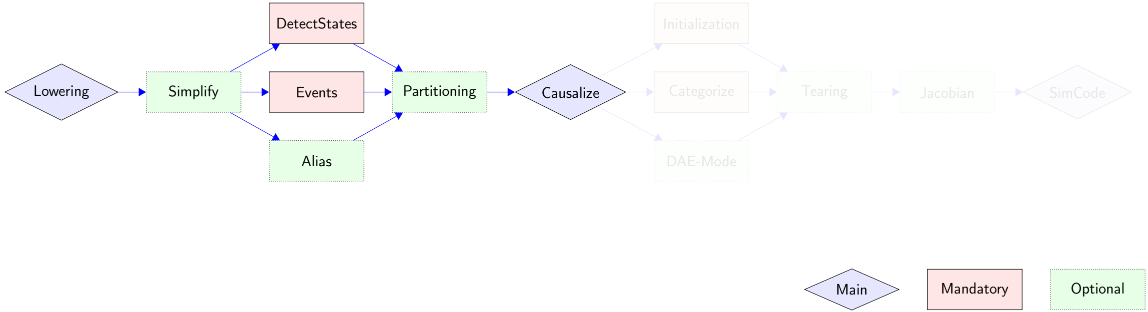
# Structures in the New Backend

## Module Dependencies



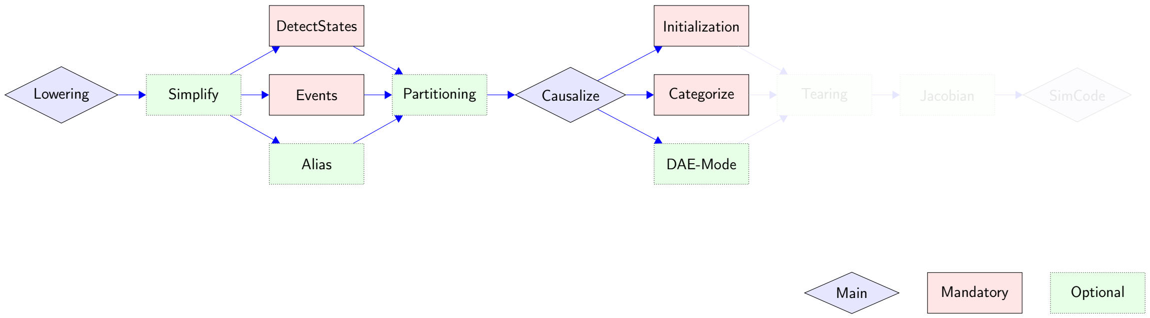
# Structures in the New Backend

## Module Dependencies



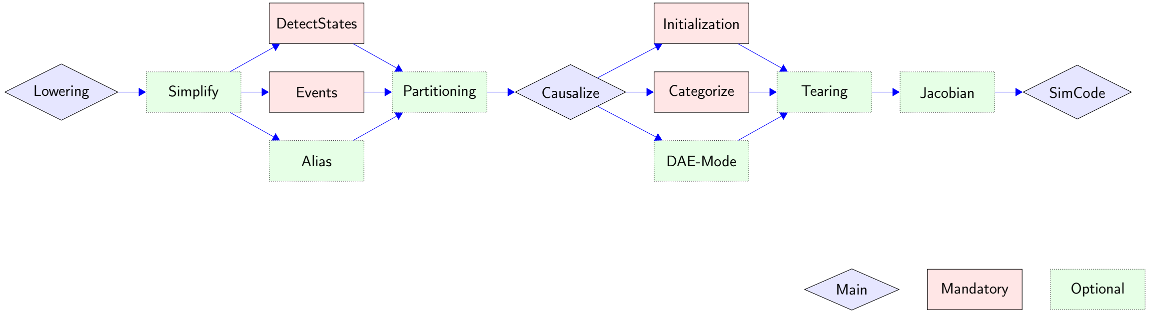
# Structures in the New Backend

## Module Dependencies



# Structures in the New Backend

## Module Dependencies



# Structures in the New Backend

## Main Structure

```
record MAIN
  list<System> ode           "Systems for ode equations";
  list<System> alg           "Systems for algebraic equations";
  list<System> ode_event     "Systems for ode event iteration";
  list<System> alg_event     "Systems for alg. event iteration";
  list<System> init          "Systems for initialization";
  Option<list<System>> init_0 "Systems for homotopy initialization";
  Option<list<System>> dae   "Systems for dae mode";

  BVariable.VarData varData  "Variable data.";
  BEquation.EqData eqData    "Equation data.";

  Events.EventInfo eventInfo "contains time and state events";
  FunctionTree funcTree      "Function bodies.";
end MAIN;
```

# Structures in the New Backend

## Variables

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

```
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.

```
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

## 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`

```
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;
```

# 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. $\$RES\_i$

```
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;
```

# 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).

```
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).

```
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.

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

partial function tearingInterface
  input output StrongComponent comp    "the suspected algebraic loop";
  input output FunctionTree funcTree   "Function call bodies";
  input output Integer index           "current unique loop index";
  input System.SystemType systemType   "system type";
end tearingInterface;
```

# 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.

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

partial function tearingInterface
  input output StrongComponent comp     "the suspected algebraic loop";
  input output FunctionTree funcTree    "Function call bodies";
  input output Integer index            "current unique loop index";
  input System.SystemType systemType    "system type";
end tearingInterface;
```

# 3. Set-Based Graphs

# Array Processing

## Set-Based Graphs

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

# Array Processing

## Set-Based Graphs

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

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

# Array Processing

## Set-Based Graphs

*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 $e_i = \{v_k^a \in V^a, v_l^b \in V^b\}$.

# 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.

# 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.

# 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.

# Array Processing

## Set-Based Graphs

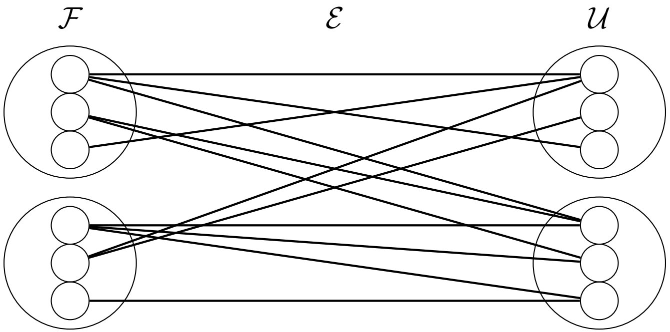
*Theory by Ernesto Kofman (CIFASIS, Rosario Argentina)*

**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 Graphs

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

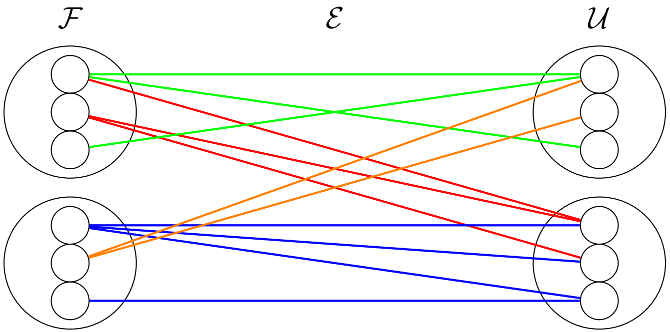
**Definition 4 (Bipartite Set–Based Graph).** A bipartite set–based graph is a set–based graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where two disjoints sets of set–vertices $\mathcal{F}, \mathcal{U}$ can be found verifying $\mathcal{F} \cup \mathcal{U} = \mathcal{V}$ and $\mathcal{F} \cap \mathcal{U}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



# Array Processing
## Set-Based Graphs

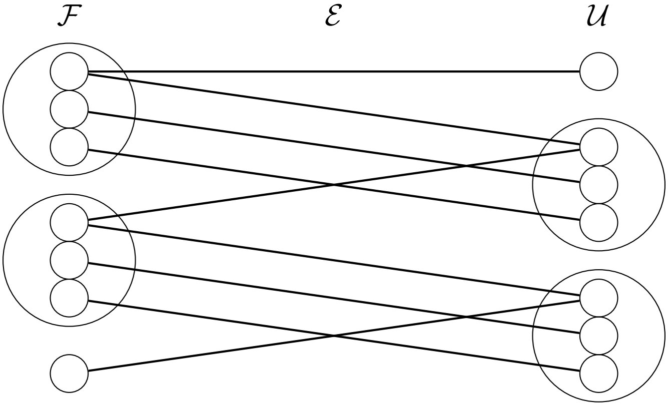
*Theory by Ernesto Kofman (CIFASIS, Rosario Argentina)*

**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 $\mathcal{U}$.



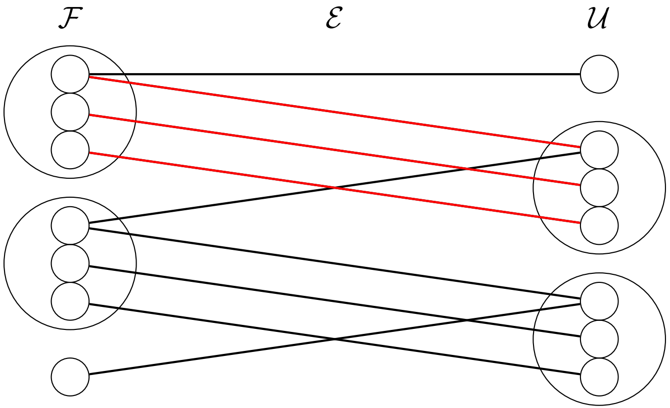
# Array Processing

## Set-Based Matching



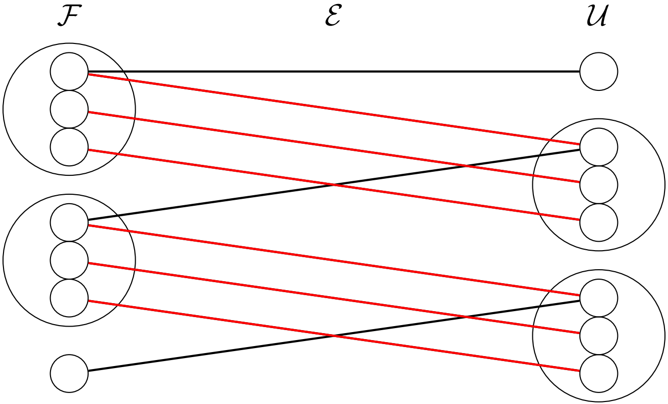
# Array Processing

## Set-Based Matching

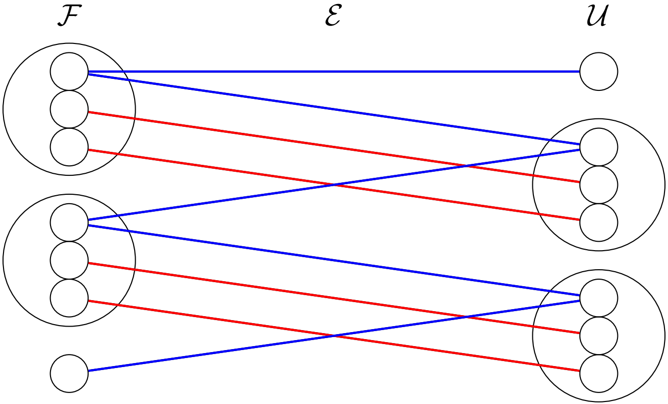


# Array Processing

## Set-Based Matching

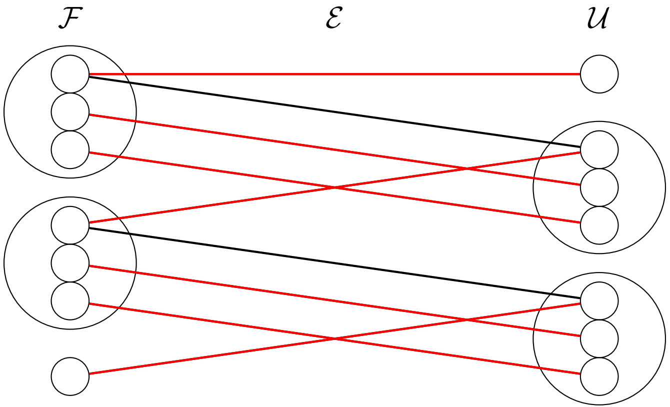


# Array Processing
## Set-Based Matching



# 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.

# Summary

Thank you for your attention!