Status of the New Backend

Karim Abdelhak, Bernhard Bachmann
University of Applied Sciences Bielefeld
Bielefeld, Germany

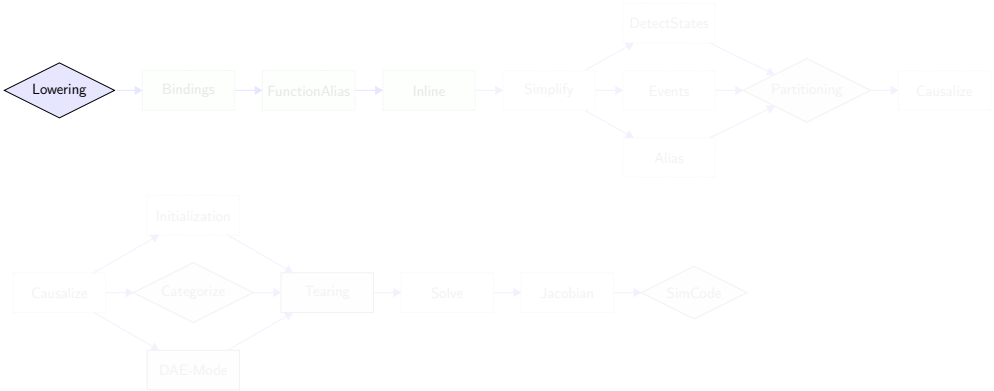
February 3, 2025

Section 1

Overview

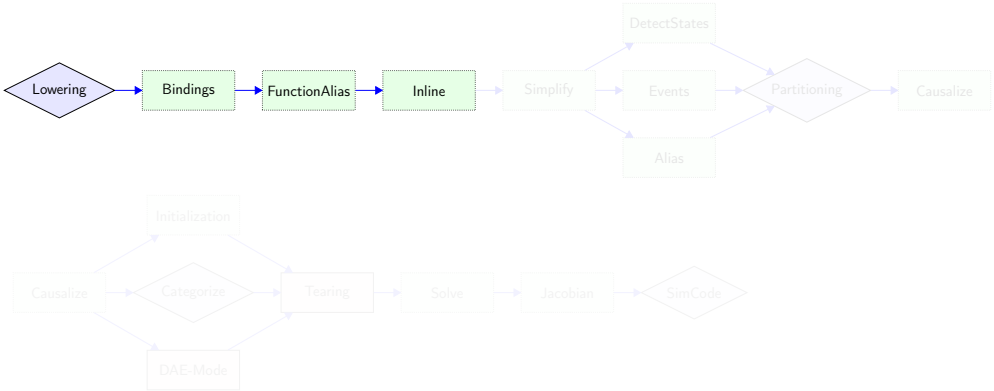
Backend Modules

Status on Array-Handling



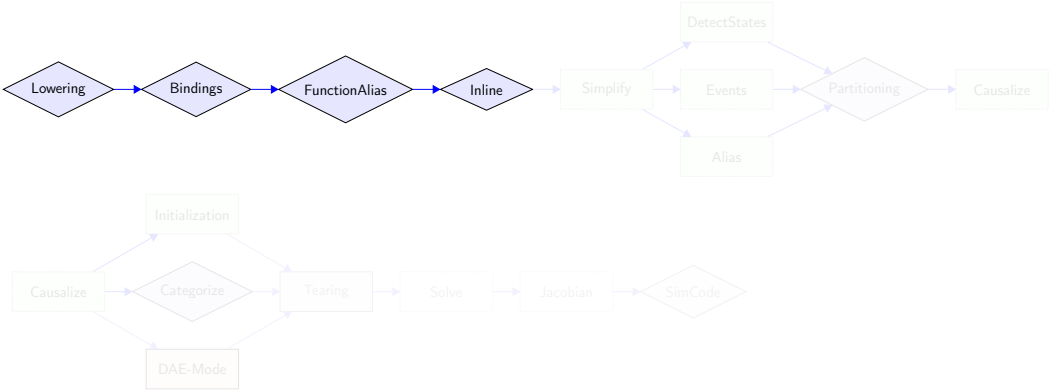
Backend Modules

Status on Array-Handling



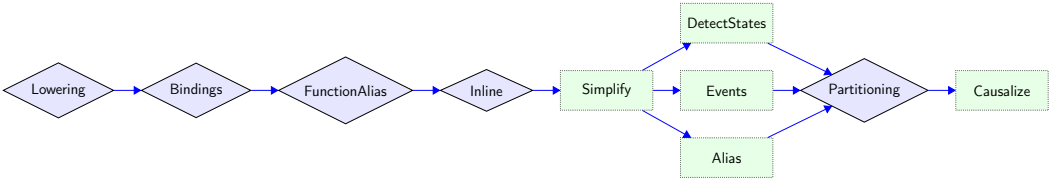
Backend Modules

Status on Array-Handling



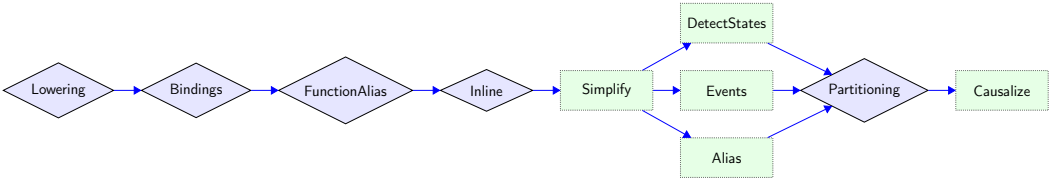
Backend Modules

Status on Array-Handling



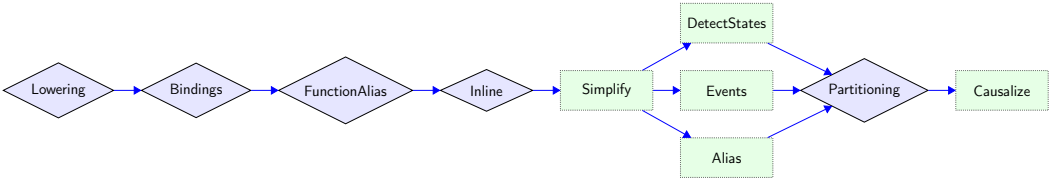
Backend Modules

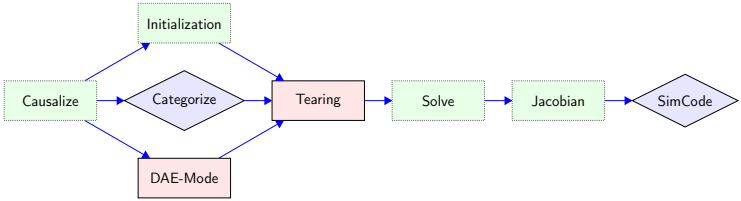
Status on Array-Handling

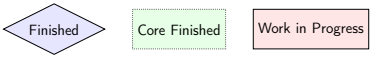


Backend Modules

Status on Array-Handling







Section 2

Resizable Arrays

Flags

Main Flag

`--resizableArrays` Assumes all arrays are resizable. Current restrictions:

- connect equations
- split arrays
- entwined for-equations

Individual Flag

Parameters that steer array sizes can be assumed to be non structural with: `annotation(__OpenModelica_resizable=true)`. Same restrictions as with `--resizableArrays`.

Debugging

Flag: `-d=dumpResizable`

Flags

Main Flag

`--resizableArrays` Assumes all arrays are resizable. Current restrictions:

- connect equations
- split arrays
- entwined for-equations

Individual Flag

Parameters that steer array sizes can be assumed to be non structural with: `annotation(__OpenModelica_resizable=true)`. Same restrictions as with `--resizableArrays`.

Debugging

Flag: `-d=dumpResizable`

Flags

Main Flag

`--resizableArrays` Assumes all arrays are resizable. Current restrictions:

- connect equations
- split arrays
- entwined for-equations

Individual Flag

Parameters that steer array sizes can be assumed to be non structural with: `annotation(__OpenModelica_resizable=true)`. Same restrictions as with `--resizableArrays`.

Debugging

Flag: `-d=dumpResizable`

Main Challenges

Resizable Strong Components

1. detecting resizable strong components after causalization
2. generating efficient code for resizable strong components
3. adapting runtime/codegen to make it resizable

Optimizing Resizable Values

- formulate an optimization problem
- solve the optimization problem
- use the solution for causalization

Motivation

1. resizing arrays after code generation
2. achieving array size independent execution time for the backend

Main Challenges

Resizable Strong Components

1. detecting resizable strong components after causalization
2. generating efficient code for resizable strong components
3. adapting runtime/codegen to make it resizable

Optimizing Resizable Values

- formulate an optimization problem
- solve the optimization problem
- use the solution for causalization

Motivation

1. resizing arrays after code generation
2. achieving array size independent execution time for the backend

Main Challenges

Resizable Strong Components

1. detecting resizable strong components after causalization
2. generating efficient code for resizable strong components
3. adapting runtime/codegen to make it resizable

Optimizing Resizable Values

- formulate an optimization problem
- solve the optimization problem
- use the solution for causalization

Motivation

1. resizing arrays after code generation
2. achieving array size independent execution time for the backend

Subsection 1

Structurally Resizable Strong Components

Structurally Resizable Strong Components

Definition

If the inner sorting of the *Simple For-Equation* results in a trivial solution, the original for-equation ranges can be kept. The solution is considered trivial if it allows for each of the ranges to be evaluated

1. in the original order (forwards)
2. reverse to the original order (backwards)
3. in any order (arbitrary)

Current Restrictions

- no split arrays
- no entwined for-equations

Structurally Resizable Strong Components

Definition

If the inner sorting of the *Simple For-Equation* results in a trivial solution, the original for-equation ranges can be kept. The solution is considered trivial if it allows for each of the ranges to be evaluated

1. in the original order (forwards)
2. reverse to the original order (backwards)
3. in any order (arbitrary)

Current Restrictions

- no split arrays
- no entwined for-equations

Structurally Resizable Strong Components

Definition

- $0 = f(X, Y, I)$: the for-equation in residual form
- $\hat{x} \in X$: the component reference for which to solve
- X: the set of all component references belonging to the same variable x as $\hat{x}$
- Y: the set of all other occuring component references (irrelevant)
- I: is the set of all for-equation iterators and their ranges

Algorithm Outline

1. compare all dimensions of $\hat{x}$ and other variables in x
2. for each dimension: see if it allows for one of the three trivial solutions

Structurally Resizable Strong Components

Definition

- $0 = f(X, Y, I)$: the for-equation in residual form
- $\hat{x} \in X$: the component reference for which to solve
- X: the set of all component references belonging to the same variable x as $\hat{x}$
- Y: the set of all other occuring component references (irrelevant)
- I: is the set of all for-equation iterators and their ranges

Algorithm Outline

1. compare all dimensions of $\hat{x}$ and other variables in x
2. for each dimension: see if it allows for one of the three trivial solutions

Structurally Resizable Strong Components

Definition

- $0 = f(X, Y, I)$: the for-equation in residual form
- $\hat{x} \in X$: the component reference for which to solve
- X: the set of all component references belonging to the same variable x as $\hat{x}$
- Y: the set of all other occuring component references (irrelevant)
- I: is the set of all for-equation iterators and their ranges

Algorithm Outline

1. compare all dimensions of $\hat{x}$ and other variables in x
2. for each dimension: see if it allows for one of the three trivial solutions

Structurally Resizable Strong Components

Definition

- $0 = f(X, Y, I)$: the for-equation in residual form
- $\hat{x} \in X$: the component reference for which to solve
- X: the set of all component references belonging to the same variable x as $\hat{x}$
- Y: the set of all other occuring component references (irrelevant)
- I: is the set of all for-equation iterators and their ranges

Algorithm Outline

1. compare all dimensions of $\hat{x}$ and other variables in x
2. for each dimension: see if it allows for one of the three trivial solutions

Structurally Resizable Strong Components

Definition

- $0 = f(X, Y, I)$: the for-equation in residual form
- $\hat{x} \in X$: the component reference for which to solve
- X: the set of all component references belonging to the same variable x as $\hat{x}$
- Y: the set of all other occuring component references (irrelevant)
- I: is the set of all for-equation iterators and their ranges

Algorithm Outline

1. compare all dimensions of $\hat{x}$ and other variables in x
2. for each dimension: see if it allows for one of the three trivial solutions

Structurally Resizable Strong Components

Definition

- $0 = f(X, Y, I)$: the for-equation in residual form
- $\hat{x} \in X$: the component reference for which to solve
- X: the set of all component references belonging to the same variable x as $\hat{x}$
- Y: the set of all other occuring component references (irrelevant)
- I: is the set of all for-equation iterators and their ranges

Algorithm Outline

1. compare all dimensions of $\hat{x}$ and other variables in x
2. for each dimension: see if it allows for one of the three trivial solutions

Structurally Resizable Strong Components

Example

```
for i in 1:p loop
  x[i] = x[i+1]*2 + x[i+2]*3;
end for;
```

1. if solved for $x[i+2]$: i has to be forwards
2. if solved for $x[i]$: i has to be backwards
3. if solved for $x[i+1]$: i cannot be solved trivially

Motivation

- avoiding generation of index lists when not necessary
- allowing the possibility of resizable for-equations

Structurally Resizable Strong Components

Example

```
for i in 1:p loop
  x[i] = x[i+1]*2 + x[i+2]*3;
end for;
```

1. if solved for $x[i+2]$: i has to be forwards
2. if solved for $x[i]$: i has to be backwards
3. if solved for $x[i+1]$: i cannot be solved trivially

Motivation

- avoiding generation of index lists when not necessary
- allowing the possibility of resizable for-equations

Structurally Resizable Strong Components

Example

```
for i in 1:p loop
  x[i] = x[i+1]*2 + x[i+2]*3;
end for;
```

1. if solved for $x[i+2]$: i has to be forwards
2. if solved for $x[i]$: i has to be backwards
3. if solved for $x[i+1]$: i cannot be solved trivially

Motivation

- avoiding generation of index lists when not necessary
- allowing the possibility of resizable for-equations

Structurally Resizable Strong Components

Example

```
for i in 1:p loop
  x[i] = x[i+1]*2 + x[i+2]*3;
end for;
```

1. if solved for $x[i+2]$: i has to be forwards
2. if solved for $x[i]$: i has to be backwards
3. if solved for $x[i+1]$: i cannot be solved trivially

Motivation

- avoiding generation of index lists when not necessary
- allowing the possibility of resizable for-equations

Structurally Resizable Strong Components

Example

```
for i in 1:p loop
  x[i] = x[i+1]*2 + x[i+2]*3;
end for;
```

1. if solved for $x[i+2]$: i has to be forwards
2. if solved for $x[i]$: i has to be backwards
3. if solved for $x[i+1]$: i cannot be solved trivially

Motivation

- avoiding generation of index lists when not necessary
- allowing the possibility of resizable for-equations

Subsection 2

Optimizing Resizable Values

Optimizing Resizable Values

Main Advantage

The main advantage of detecting resizable for-equations lies in the possiblity to size them down as much as possible for all symbolical optimizations.

Target Function

Optimizing parameter values (x) such that it minimizes the size of the equation system.

Constraints

- structural variable constraints (e.g. box-constraints of `min` and `max` values)
- structural equation constraints (e.g. implied array size equalities)
- retain equation structure such that resizable strong component analysis is not compromised

Optimizing Resizable Values

Main Advantage

The main advantage of detecting resizable for-equations lies in the possiblity to size them down as much as possible for all symbolical optimizations.

Target Function

Optimizing parameter values (x) such that it minimizes the size of the equation system.

Constraints

- structural variable constraints (e.g. box-constraints of `min` and `max` values)
- structural equation constraints (e.g. implied array size equalities)
- retain equation structure such that resizable strong component analysis is not compromised

Optimizing Resizable Values

Main Advantage

The main advantage of detecting resizable for-equations lies in the possiblity to size them down as much as possible for all symbolical optimizations.

Target Function

Optimizing parameter values (x) such that it minimizes the size of the equation system.

Constraints

- structural variable constraints (e.g. box-constraints of `min` and `max` values)
- structural equation constraints (e.g. implied array size equalities)
- retain equation structure such that resizable strong component analysis is not compromised

Optimizing Resizable Values

Main Advantage

The main advantage of detecting resizable for-equations lies in the possiblity to size them down as much as possible for all symbolical optimizations.

Target Function

Optimizing parameter values (x) such that it minimizes the size of the equation system.

Constraints

- structural variable constraints (e.g. box-constraints of `min` and `max` values)
- structural equation constraints (e.g. implied array size equalities)
- retain equation structure such that resizable strong component analysis is not compromised

Optimizing Resizable Values

Main Advantage

The main advantage of detecting resizable for-equations lies in the possiblity to size them down as much as possible for all symbolical optimizations.

Target Function

Optimizing parameter values (x) such that it minimizes the size of the equation system.

Constraints

- structural variable constraints (e.g. box-constraints of `min` and `max` values)
- structural equation constraints (e.g. implied array size equalities)
- retain equation structure such that resizable strong component analysis is not compromised

Target Function

Dimension Target Function

Each dimension has a seperate target function that differs if it is a dimension which is span by a resizable-constrained iterator of a for-equation (a_1):

$$d_l(x) = \begin{cases} (D_{\text{stop}}(x) - D_{\text{start}}(x))/D_{\text{step}} + 1 & \text{if } (a_l) \\ D_{size}(x) & \text{else} \end{cases} \quad (1)$$

Equation Target Function

Multiplying all dimension sizes lead to the equation size:

$$f_k(x) = \Pi_l d_l(x) \quad (2)$$

System Target Function

Accumulating all (relevant) equation sizes leads to the system target function:

$$\text{min!} \quad F(x) = \Sigma_k f_k(x) \quad (3)$$

Target Function

Dimension Target Function

Each dimension has a seperate target function that differs if it is a dimension which is span by a resizable-constrained iterator of a for-equation (a_1):

$$d_l(x) = \begin{cases} (D_{\text{stop}}(x) - D_{\text{start}}(x))/D_{\text{step}} + 1 & \text{if } (a_l) \\ D_{size}(x) & \text{else} \end{cases} \quad (1)$$

Equation Target Function

Multiplying all dimension sizes lead to the equation size:

$$f_k(x) = \Pi_l d_l(x) \quad (2)$$

System Target Function

Accumulating all (relevant) equation sizes leads to the system target function:

$$\text{min!} \quad F(x) = \Sigma_k f_k(x) \quad (3)$$

Target Function

Dimension Target Function

Each dimension has a seperate target function that differs if it is a dimension which is span by a resizable-constrained iterator of a for-equation (a_1):

$$d_l(x) = \begin{cases} (D_{\text{stop}}(x) - D_{\text{start}}(x))/D_{\text{step}} + 1 & \text{if } (a_l) \\ D_{size}(x) & \text{else} \end{cases} \quad (1)$$

Equation Target Function

Multiplying all dimension sizes lead to the equation size:

$$f_k(x) = \Pi_l d_l(x) \quad (2)$$

System Target Function

Accumulating all (relevant) equation sizes leads to the system target function:

$$\text{min!} \quad F(x) = \Sigma_k f_k(x) \quad (3)$$

Constraints

Variable Constraints

- user-defined box-constraints:

$$x \geq x_{\text{min}} \tag{4}$$
$$x \leq x_{\text{max}} \tag{5}$$

- subscript-implied constraints:

$$\text{sub}_k(x) \leq \text{dim}_k(x) \tag{6}$$

for each component reference in each equation with $sub_k(x)$ being the expression representing the subscript and $dim_k(x)$ being the expression representing the dimension with dimension index k.

Constraints

Variable Constraints

- user-defined box-constraints:

$$x \geq x_{\text{min}} \tag{4}$$
$$x \leq x_{\text{max}} \tag{5}$$

- subscript-implied constraints:

$$\text{sub}_k(x) \leq \text{dim}_k(x) \tag{6}$$

for each component reference in each equation with $sub_k(x)$ being the expression representing the subscript and $dim_k(x)$ being the expression representing the dimension with dimension index k.

Constraints

Equation Constraints

For all equations the size of the left hand and the right hand side have to be equal.

$$\text{lhs}_i^k(x) = \text{rhs}_i^k(x) \tag{7}$$

where $\text{lhs}_i^k(x)$ and $\text{rhs}_i^k(x)$ are the left hand side and respective right hand side expression of dimension k in equation i.

Iterator Constraints

Ensure a minimal size 2 for each iterator to retain necessary structures. This leads to constraints of the form

$$2 \leq (D^i_{\text{stop}}(x) - D^i_{\text{start}}(x)) / D^i_{\text{step}} \tag{8}$$

for all iterators i.

Constraints

Equation Constraints

For all equations the size of the left hand and the right hand side have to be equal.

$$\text{lhs}_i^k(x) = \text{rhs}_i^k(x) \tag{7}$$

where $\text{lhs}_i^k(x)$ and $\text{rhs}_i^k(x)$ are the left hand side and respective right hand side expression of dimension k in equation i.

Iterator Constraints

Ensure a minimal size 2 for each iterator to retain necessary structures. This leads to constraints of the form

$$2 \leq (D^i_{\text{stop}}(x) - D^i_{\text{start}}(x)) / D^i_{\text{step}} \tag{8}$$

for all iterators i.

Solving the Optimization Problem

Classification

The parameters have to be integer valued and the target function as well as the constraints can be nonlinear. Even though it oftentimes is only linear/convex we have to consider the possibility of a general **integer valued non linear optimization problem** which is **NP-hard**.

Solving Algorithms

No *optimal* solution needed. Any reasonably good one that is feasable, suffices. Two attempts are designed:

1. starting in a reasonably good point using min/max values and trying to reach feasability
2. starting in a feasable point using default bindings and trying to reach optimality under the assumption of a convex search space

Solving the Optimization Problem

Classification

The parameters have to be integer valued and the target function as well as the constraints can be nonlinear. Even though it oftentimes is only linear/convex we have to consider the possibility of a general **integer valued non linear optimization problem** which is **NP-hard**.

Solving Algorithms

No *optimal* solution needed. Any reasonably good one that is feasable, suffices. Two attempts are designed:

1. starting in a reasonably good point using min/max values and trying to reach feasability
2. starting in a feasable point using default bindings and trying to reach optimality under the assumption of a convex search space

Solving the Optimization Problem

Classification

The parameters have to be integer valued and the target function as well as the constraints can be nonlinear. Even though it oftentimes is only linear/convex we have to consider the possibility of a general **integer valued non linear optimization problem** which is **NP-hard**.

Solving Algorithms

No *optimal* solution needed. Any reasonably good one that is feasable, suffices. Two attempts are designed:

1. starting in a reasonably good point using min/max values and trying to reach feasability
2. starting in a feasable point using default bindings and trying to reach optimality under the assumption of a convex search space

Solving the Optimization Problem

Classification

The parameters have to be integer valued and the target function as well as the constraints can be nonlinear. Even though it oftentimes is only linear/convex we have to consider the possibility of a general **integer valued non linear optimization problem** which is **NP-hard**.

Solving Algorithms

No *optimal* solution needed. Any reasonably good one that is feasable, suffices. Two attempts are designed:

1. starting in a reasonably good point using min/max values and trying to reach feasability
2. starting in a feasable point using default bindings and trying to reach optimality under the assumption of a convex search space

Subsection 3

Target Code

Resizable Arrays: Variables

Old Memory Layout

flat variables:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}

New Memory Layout

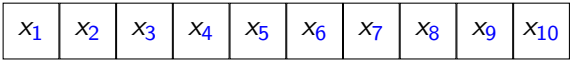
index map:

1	5	6	7

Resizable Arrays: Variables

Old Memory Layout

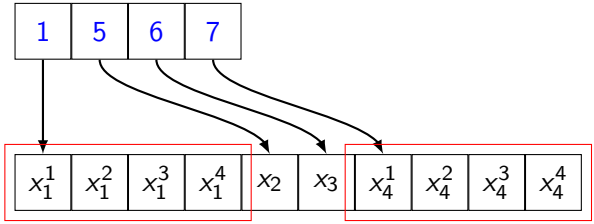
flat variables:



New Memory Layout

index map:

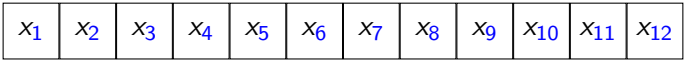
flat variables:



Resizable Arrays: Variables

Old Memory Layout

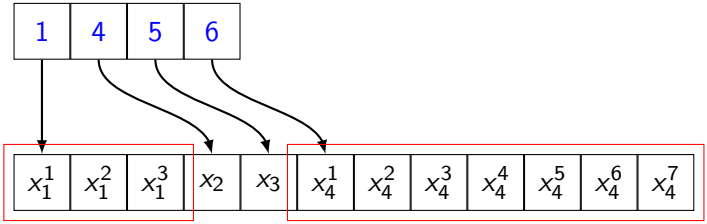
flat variables:



New Memory Layout

index map:

flat variables:



Section 3

Summary

Results

- Overview
- Large TestSuite NB resizable NB OB
- Recent Coverage Scalable TestSuite PowerGrids

Summary

Recent Development

- Compact for-loop structures
- Backend resizable support

Current Development

- Jacobian and sparsity updates
- Target code resizable support

Upcoming Plans

- Pseudo-Array Index Reduction

Summary

Recent Development

- Compact for-loop structures
- Backend resizable support

Current Development

- Jacobian and sparsity updates
- Target code resizable support

Upcoming Plans

- Pseudo-Array Index Reduction

Summary

Recent Development

- Compact for-loop structures
- Backend resizable support

Current Development

- Jacobian and sparsity updates
- Target code resizable support

Upcoming Plans

- Pseudo-Array Index Reduction