Formal Languages and Compilers - Exercises
Lecture 6
Composite data types: vectors and matrices

03/04/2012

Outline

1 Definition

2 Arrays in crème CAraMeL

3 Bidimensional Matrices

4 Multidimensional matrices

5 Slices

6 Implementation

Definition

Data: "Container" for values (var or const)

Value: Something that is put in the data (everything that is representable with a sequence of bits)

Data type (DT): Class for data and operations to manipulate it

Definition

Data: "Container" for values (var or const)

Value: Something that is put in the data (everything that is representable with a sequence of bits)

Data type (DT): Class for data and operations to manipulate it

Definition

Data: "Container" for values (var or const)

Value: Something that is put in the data (everything that is representable with a sequence of bits)

Data type (DT): Class for data and operations to manipulate it

Data

Categories

- Basic data types: integers, floats, characters, enumerable types,...
- Structured data (data structures): matrices, records, lists,...

Specification

- Attributes: "technical" aspects for managing data
- Values: what you can put inside the data
- Operations: what you can do with that data

Implementation

How the specification is realized in practice

Data

Categories

- Basic data types: integers, floats, characters, enumerable types,...
- Structured data (data structures): matrices, records, lists,...

Specification

- Attributes: "technical" aspects for managing data
- Values: what you can put inside the data
- Operations: what you can do with that data

Implementation

How the specification is realized in practice

Data

Categories

- Basic data types: integers, floats, characters, enumerable types,...
- Structured data (data structures): matrices, records, lists,...

Specification

- Attributes: "technical" aspects for managing data
- Values: what you can put inside the data
- Operations: what you can do with that data

Implementation

How the specification is realized in practice

Data

Categories

- Basic data types: integers, floats, characters, enumerable types,...
- Structured data (data structures): matrices, records, lists,...

Specification

- Attributes: "technical" aspects for managing data
- Values: what you can put inside the data
- Operations: what you can do with that data

Implementation

How the specification is realized in practice

Data

Categories

- Basic data types: integers, floats, characters, enumerable types,...
- Structured data (data structures): matrices, records, lists,...

Specification

- Attributes: "technical" aspects for managing data
- Values: what you can put inside the data
- Operations: what you can do with that data

Implementation

How the specification is realized in practice

Data

Categories

- Basic data types: integers, floats, characters, enumerable types,...
- Structured data (data structures): matrices, records, lists,...

Specification

- Attributes: "technical" aspects for managing data
- Values: what you can put inside the data
- Operations: what you can do with that data

Implementation

How the specification is realized in practice

Example of Basic Data Type: Integers

Specification

- Attributes: how it is represented in the internal memory
- Values: the maximum and minimum are defined as *MinInt*, *MaxInt*
- Operations: Sum, Multiplication, Subtraction, Division,...

Implementation

- Attributes: decide at compile-time or at run-time
- Values: nothing to declare
- Operations: HW operations `ADD`, `MUL`,... or `procedure`: `Sum(x,y) = x + y`,...

Example of Basic Data Type: Integers

Specification

- Attributes: how it is represented in the internal memory
- Values: the maximum and minimum are defined as *MinInt*, *MaxInt*
- Operations: Sum, Multiplication, Subtraction, Division,...

Implementation

- Attributes: decide at compile-time or at run-time
- Values: nothing to declare
- Operations: HW operations `ADD`, `MUL`,... or `procedure`: `Sum(x,y) = x + y`,...

Example of Basic Data Type: Integers

Specification

- Attributes: how it is represented in the internal memory
- Values: the maximum and minimum are defined as *MinInt*, *MaxInt*
- Operations: Sum, Multiplication, Subtraction, Division,...

Implementation

- Attributes: decide at compile-time or at run-time
- Values: nothing to declare
- Operations: HW operations `ADD`, `MUL`,... or `procedure`: `Sum(x,y) = x + y`,...

Example of Basic Data Type: Integers

Specification

- Attributes: how it is represented in the internal memory
- Values: the maximum and minimum are defined as *MinInt*, *MaxInt*
- Operations: Sum, Multiplication, Subtraction, Division,...

Implementation

- Attributes: decide at compile-time or at run-time
- Values: nothing to declare
- Operations: HW operations `ADD`, `MUL`,... or `procedure: Sum(x,y) = x + y`,...

Example of Basic Data Type: Integers

Specification

- Attributes: how it is represented in the internal memory
- Values: the maximum and minimum are defined as *MinInt*, *MaxInt*
- Operations: Sum, Multiplication, Subtraction, Division,...

Implementation

- Attributes: decide at compile-time or at run-time
- Values: nothing to declare
- Operations: HW operations `ADD`, `MUL`,... or `procedure: Sum(x,y) = x + y`,...

Example of Basic Data Type: Integers

Specification

- Attributes: how it is represented in the internal memory
- Values: the maximum and minimum are defined as *MinInt*, *MaxInt*
- Operations: Sum, Multiplication, Subtraction, Division,...

Implementation

- Attributes: decide at compile-time or at run-time
- Values: nothing to declare
- Operations: HW operations `ADD`, `MUL`,... or `procedure`: `Sum(x,y) = x + y`,...

Data structure: array

Specification

Attributes
- Number of the components
- Type of components
- A way to access them,...

Values
- Decided by the attributes

Operations
- Modify the structure (insert, delete,...)
- Operations over one component
- Operations over the entire structure (comparison, copy)

Data structure: array

Specification

Attributes
- Number of the components
- Type of components
- A way to access them,...

Values
- Decided by the attributes

Operations
- Modify the structure (insert, delete,...)
- Operations over one component
- Operations over the entire structure (comparison, copy)

Data structure: array

Specification

Attributes
- Number of the components
- Type of components
- A way to access them,...

Values
- Decided by the attributes

Operations
- Modify the structure (insert, delete,...)
- Operations over one component
- Operations over the entire structure (comparison, copy)

Data structure: array

Specification

Attributes
- Number of the components
- Type of components
- A way to access them,...

Values
- Decided by the attributes

Operations
- Modify the structure (insert, delete,...)
- Operations over one component
- Operations over the entire structure (comparison, copy)

Data structure: array

Specification

Attributes
- Number of the components
- Type of components
- A way to access them,...

Values
- Decided by the attributes

Operations
- Modify the structure (insert, delete,...)
- Operations over one component
- Operations over the entire structure (comparison, copy)

Data structure: array

Specification

Attributes
- Number of the components
- Type of components
- A way to access them,...

Values
- Decided by the attributes

Operations
- Modify the structure (insert, delete,...)
- Operations over one component
- Operations over the entire structure (comparison, copy)

Data structure: array

Specification

Attributes
- Number of the components
- Type of components
- A way to access them,...

Values
- Decided by the attributes

Operations
- Modify the structure (insert, delete,...)
- Operations over one component
- Operations over the entire structure (comparison, copy)

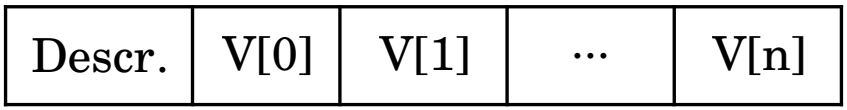
Data structure: array

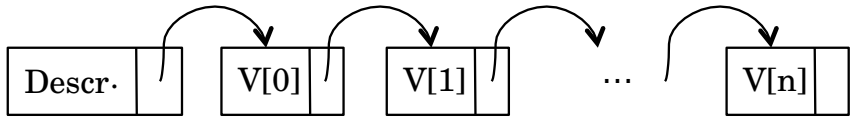
Implementation

Attributes in the *descriptor*

Values like before

Operations access to the elements





Pros.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\| = B + O(1)$

Cons.

Insertion and deletion

Pros.

Insertion and deletion

Cons.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\|$ = scanning the whole list

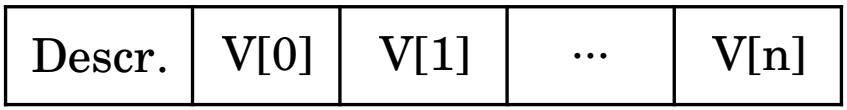
Data structure: array

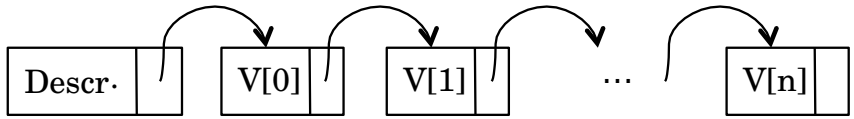
Implementation

Attributes in the *descriptor*

Values like before

Operations access to the elements





Pros.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\| = B + O(1)$

Cons.

Insertion and deletion

Pros.

Insertion and deletion

Cons.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\|$ = scanning the whole list

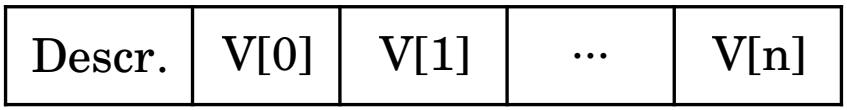
Data structure: array

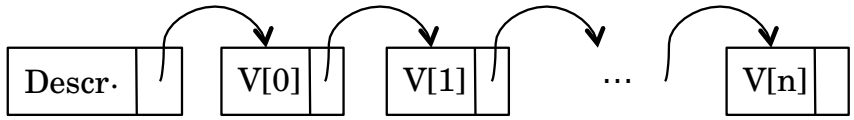
Implementation

Attributes in the *descriptor*

Values like before

Operations access to the elements





Pros.

$\Lambda\|\mathtt{V}[\mathtt{k}]\| = B + O(1)$

Cons.

Insertion and deletion

Pros.

Insertion and deletion

Cons.

$\Lambda\|\mathtt{V}[\mathtt{k}]\|$ = scanning the whole list

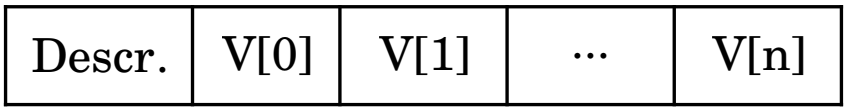
Data structure: array

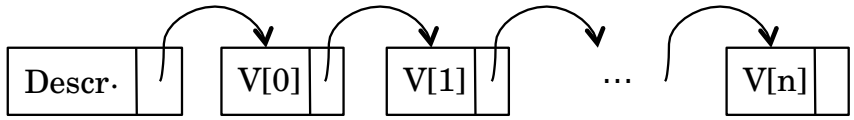
Implementation

Attributes in the *descriptor*

Values like before

Operations access to the elements





Pros.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\| = B + O(1)$

Cons.

Insertion and deletion

Pros.

Insertion and deletion

Cons.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\|$ = scanning the whole list

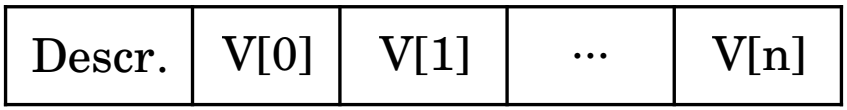
Data structure: array

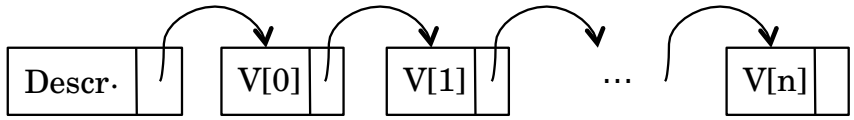
Implementation

Attributes in the *descriptor*

Values like before

Operations access to the elements





Pros.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\| = B + O(1)$

Cons.

Insertion and deletion

Pros.

Insertion and deletion

Cons.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\|$ = scanning the whole list

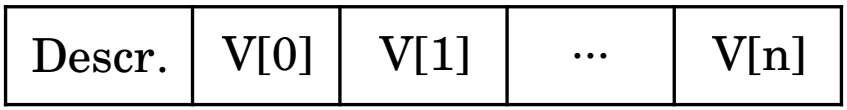
Data structure: array

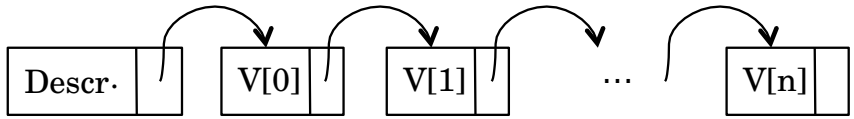
Implementation

Attributes in the *descriptor*

Values like before

Operations access to the elements





Pros.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\| = B + O(1)$

Cons.

Insertion and deletion

Pros.

Insertion and deletion

Cons.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\|$ = scanning the whole list

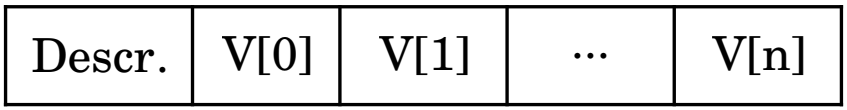
Data structure: array

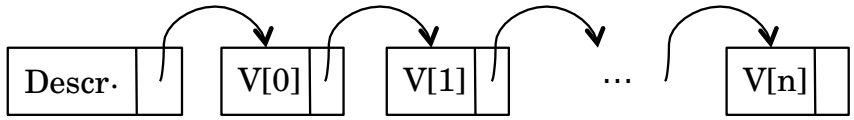
Implementation

Attributes in the *descriptor*

Values like before

Operations access to the elements





Pros.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\| = B + O(1)$

Cons.

Insertion and deletion

Pros.

Insertion and deletion

Cons.

$\Lambda\|\mathtt{V}\,[\mathtt{k}]\,\|$ = scanning the whole list

Outline

1 Definition

2 Arrays in crème CAraMeL

3 Bidimensional Matrices

4 Multidimensional matrices

5 Slices

6 Implementation

Arrays in crème CAraMeL

- Data structure
- Homogenous (consists of elements of one type)
- Fixed length represented by a sequence

Linear array: vector

Multidimensional array: matrix (remembered line by line)

Arrays in crème CAraMeL

- Data structure
- Homogenous (consists of elements of one type)
- Fixed length represented by a sequence

Linear array: vector

Multidimensional array: matrix (remembered line by line)

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\texttt{V[k]}\| = \alpha + (\texttt{k} - \texttt{LB}) \times M$
- Declaration

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\texttt{V[k]}\| = \alpha + (\texttt{k} - \texttt{LB}) \times M$
- Declaration

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\texttt{V[k]}\| = \alpha + (\texttt{k} - \texttt{LB}) \times M$
- Declaration

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\mathtt{V[k]}\| = \alpha + (\mathtt{k} - \mathtt{LB}) \times M$
- Declaration

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\texttt{V[k]}\| = \alpha + (\texttt{k} - \texttt{LB}) \times M$
- Declaration

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\mathtt{V[k]}\| = \alpha + (\mathtt{k} - \mathtt{LB}) \times M$
- Declaration

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\texttt{V[k]}\| = \alpha + (\texttt{k} - \texttt{LB}) \times M$
- Declaration

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\mathtt{V[k]}\| = \alpha + (\mathtt{k} - \mathtt{LB}) \times M$
- Declaration

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\texttt{V[k]}\| = \alpha + (\texttt{k} - \texttt{LB}) \times M$
- Declaration

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\texttt{V[k]}\| = \alpha + (\texttt{k} - \texttt{LB}) \times M$
- Declaration

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\texttt{V[k]}\| = \alpha + (\texttt{k} - \texttt{LB}) \times M$
- Declaration

Vector in crème CAraMeL

Specification

Attributes

- Number of elements
- Type (dim.) of elements
- Component name = index

Values

v. number and type

Operations

- Access to the elements
- Creation/elimination of the vectors

Implementation

Attributes

- `var V: array [LB...UB] of type`
- type $\longrightarrow$ Multiplier
- `O(k) = M` $\times$ `k`

Values

`UB - LB + 1` elements of type *type*

Operations

- $\Lambda\|\texttt{V[k]}\| = \alpha + (\texttt{k} - \texttt{LB}) \times M$
- Declaration

Vectors: implementation

Address of the k-th element

$$\begin{aligned}\Lambda\|V[k]\| &= \alpha + (k - \mathtt{LB}) \times \mathtt{M} \\ &= (\alpha - \mathtt{LB} \times \mathtt{M}) + k \times \mathtt{M} \\ &= VO + k \times \mathtt{M}\end{aligned}$$

$$\begin{aligned}VO &= \alpha - \mathtt{LB} \times \mathtt{M} \\ &= \Lambda\|V[0]\|\end{aligned}$$

Vectors: implementation

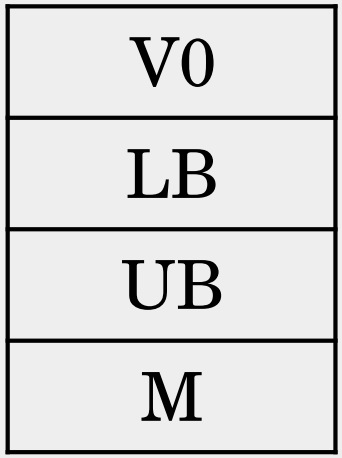
Address of the k-th element

$$\begin{aligned}\Lambda\|V[k]\| &= \alpha + (k - \mathtt{LB}) \times \mathtt{M} \\ &= (\alpha - \mathtt{LB} \times \mathtt{M}) + k \times \mathtt{M} \\ &= VO + k \times \mathtt{M}\end{aligned}$$

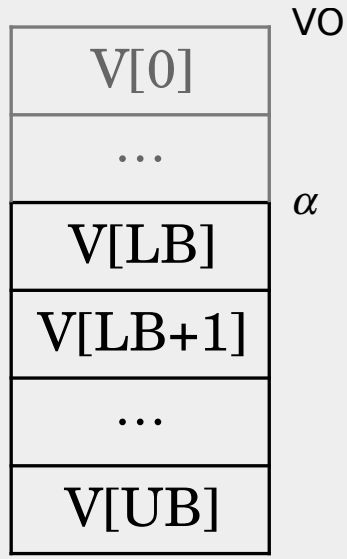
$$\begin{aligned}VO &= \alpha - \mathtt{LB} \times \mathtt{M} \\ &= \Lambda\|V[0]\|\end{aligned}$$

Vectors: implementation

Descriptor



In Memory



Vectors: implementation

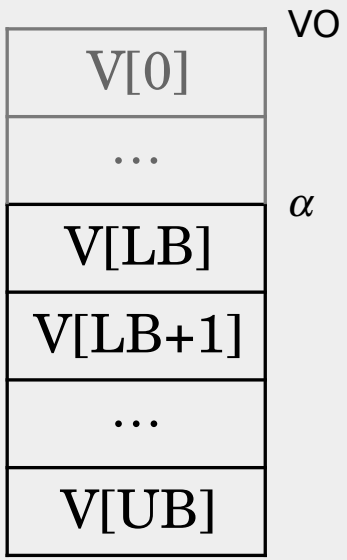
Simplification

By having $\mathtt{M} = 1$, we obtain $\Lambda\|V[k]\| = VO + k$ and $VO = \alpha - \mathtt{LB} = \Lambda\|V[0]\|$

Descriptor

V0
LB
UB

In Memory



Outline

1 Definition

2 Arrays in crème CAraMeL

3 Bidimensional Matrices

4 Multidimensional matrices

5 Slices

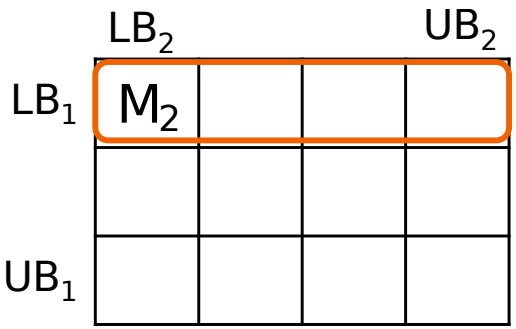
6 Implementation

Bidimensional Matrices

Definition

`var V : array[`$\mathtt{LB}_1$ `..`$\mathtt{UB}_1$`,`$\mathtt{LB}_2$`..`$\mathtt{UB}_2$`] of` *type*

- Dimension of an element: $\mathtt{M}_2$
- Dimension of a row: $\mathtt{M}_1 = (\mathtt{UB}_2 - \mathtt{LB}_2 + 1) \times \mathtt{M}_2$

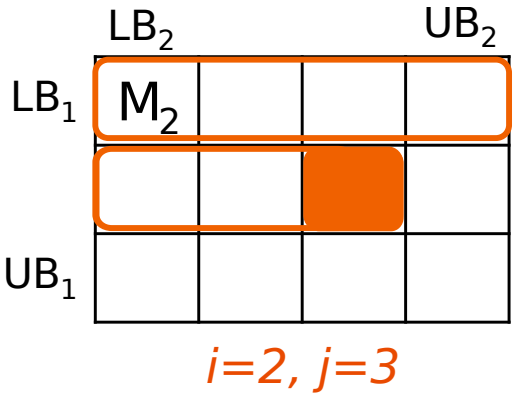


Bidimensional Matrices

Definition

`var V : array[LB1 ..UB1,LB2..UB2] of type`

- Dimension of an element: M_2
- Dimension of a row: $M_1 = (UB_2 - LB_2 + 1) \times M_2$



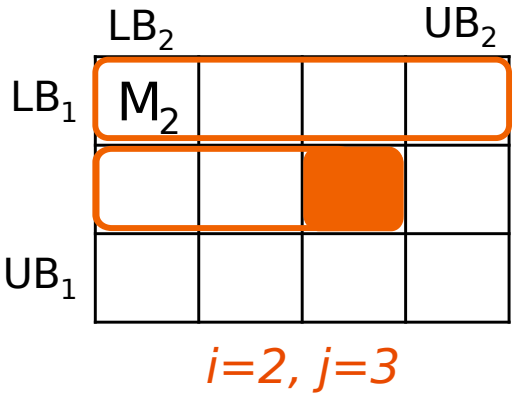
- Virtual Origin:
  $VO = \alpha - LB_1 \times M_1 - LB_2 \times M_2$
- $\Lambda\|V[i,j]\| = VO + i \times M_1 + j \times M_2$

Bidimensional Matrices

Definition

`var V : array[LB1 ..UB1,LB2..UB2] of type`

- Dimension of an element: $\mathtt{M}_2$
- Dimension of a row: $\mathtt{M}_1 = (\mathtt{UB}_2 - \mathtt{LB}_2 + 1) \times \mathtt{M}_2$



- Virtual Origin:
  $VO = \alpha - \mathtt{LB}_1 \times \mathtt{M}_1 - \mathtt{LB}_2 \times \mathtt{M}_2$
- $\Lambda\|V[\mathtt{i},\mathtt{j}]\| = VO + \mathtt{i} \times \mathtt{M}_1 + \mathtt{j} \times \mathtt{M}_2$

Outline

1 Definition

2 Arrays in crème CAraMeL

3 Bidimensional Matrices

4 Multidimensional matrices

5 Slices

6 Implementation

Multidimensional matrices

$\texttt{var V : array}[\texttt{LB}_1..\texttt{UB}_1, \ldots, \texttt{LB}_n..\texttt{UB}_n] \texttt{ of } \textit{type}$

Multipliers

$$\texttt{M}_n = \texttt{M}$$

$$\texttt{M}_i = (\texttt{UB}_{i+1} - \texttt{LB}_{i+1} + 1) \times \texttt{M}_{i+1}, \quad i \in [1, n-1]$$

$$VO = \alpha - \sum_{i=1}^{n} \texttt{LB}_i \times \texttt{M}_i$$

$$\Lambda \| V[k_1, \ldots, k_n] = VO + \sum_{i=1}^{n} k_i \times \texttt{M}_i$$

Multidimensional matrices

`var V : array[`$\mathtt{LB}_1..\mathtt{UB}_1, \ldots, \mathtt{LB}_n..\mathtt{UB}_n$`] of` *type*

Multipliers

$$\mathtt{M}_n = \mathtt{M}$$

$$\mathtt{M}_i = (\mathtt{UB}_{i+1} - \mathtt{LB}_{i+1} + 1) \times \mathtt{M}_{i+1}, \quad i \in [1, n-1]$$

$$VO = \alpha - \sum_{i=1}^{n} \mathtt{LB}_i \times \mathtt{M}_i$$

$$\Lambda \| V[k_1, \ldots, k_n] = VO + \sum_{i=1}^{n} k_i \times \mathtt{M}_i$$

Multidimensional matrices

Attention

$$\texttt{array}\,[\texttt{LB}_1\,..\,\texttt{UB}_1, \texttt{LB}_n\,..\,\texttt{UB}_n]\ \texttt{of}\ \textit{type}$$

$$\sim$$

$$\texttt{array}\,[\texttt{LB}_1\,..\,\texttt{UB}_1]\ \texttt{of}\ (\texttt{array}\,[\texttt{LB}_2\,..\,\texttt{UB}_2,\ \texttt{LB}_n\,..\,\texttt{UB}_n]\ \texttt{of}\ \textit{type})$$

Outline

1 Definition

2 Arrays in crème CAraMeL

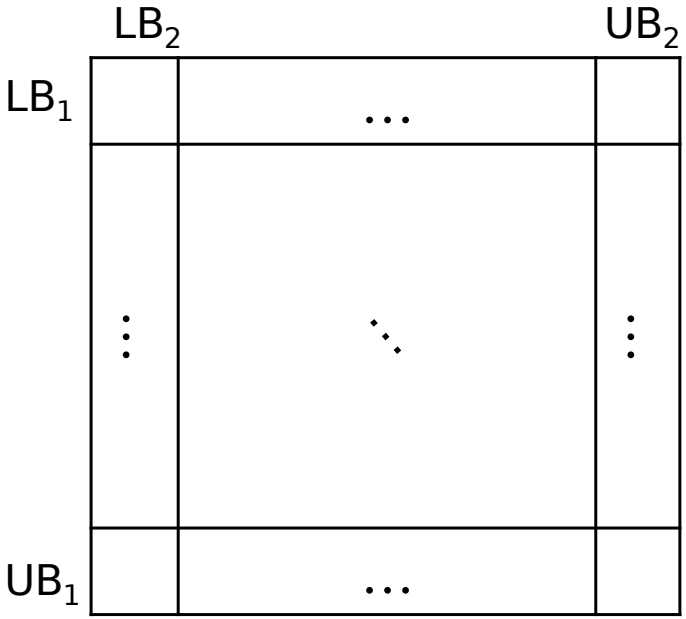
3 Bidimensional Matrices

4 Multidimensional matrices

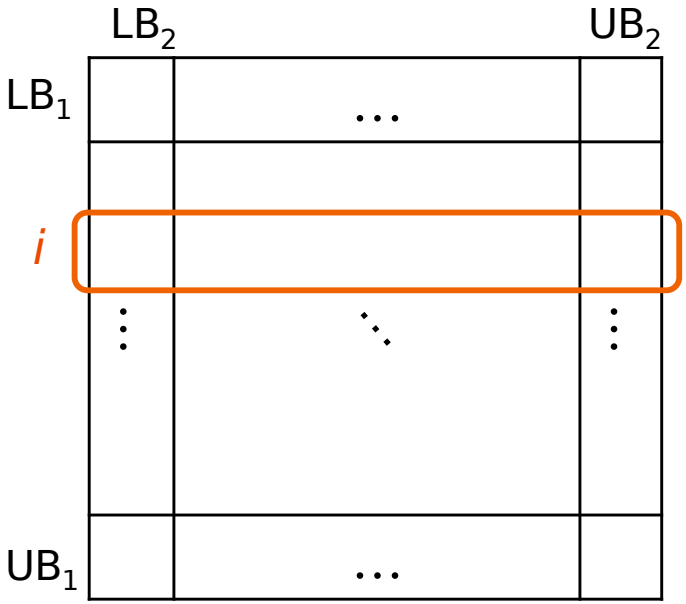
5 Slices

6 Implementation

Slices of array



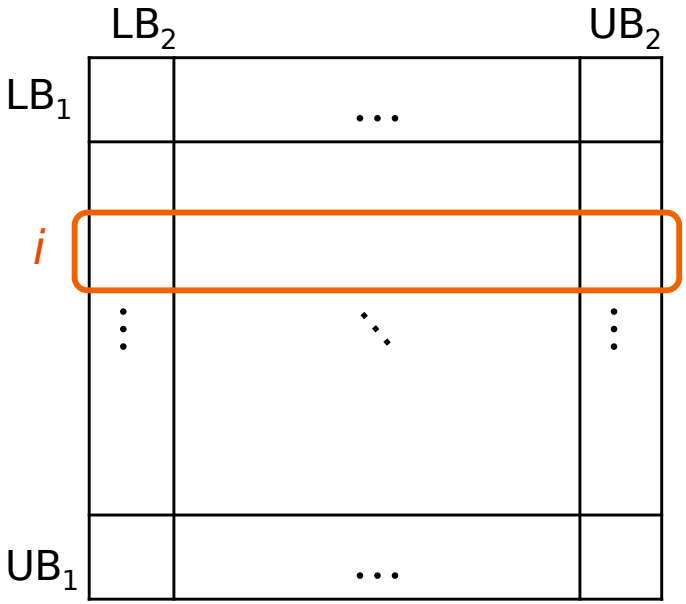
Slices of array



$$\begin{aligned}
\mathtt{M} &= \mathtt{M}_2 \\
\mathtt{VO}_I &= \mathtt{VO}_V + i \times (\mathtt{UB}_2 - \mathtt{LB}_2 + 1) \times \mathtt{M}_2 \\
&= \mathtt{VO}_V + i \times \mathtt{M}_1 \\
\mathtt{LB} &= \mathtt{LB}_2 \\
\mathtt{UB} &= \mathtt{UB}_2 \\
\Lambda\|I[k]\| &= \mathtt{VO}_I + k \times \mathtt{M}
\end{aligned}$$

$$\mathtt{I} = \mathtt{V[i][*]} = V[i][\mathtt{LB}_2], V[i][\mathtt{LB}_2 + 1], \ldots, V[i][\mathtt{UB}_2]$$

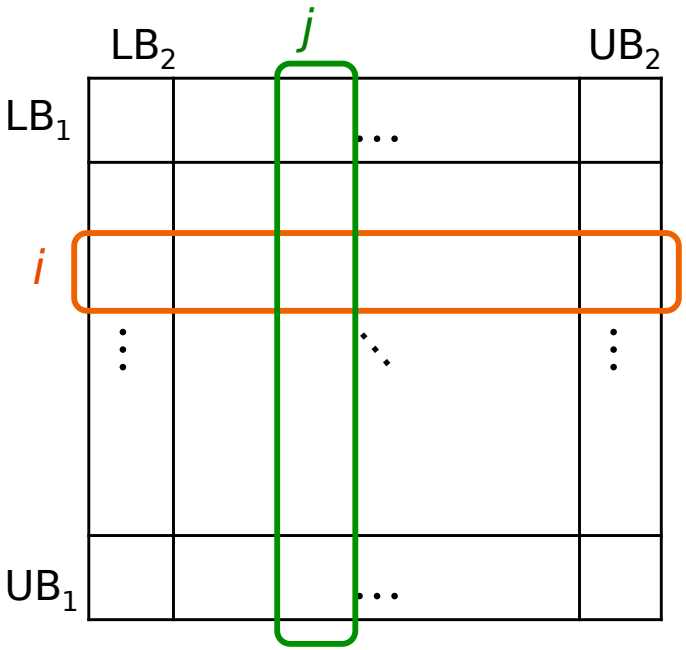
Slices of array



$$
\begin{aligned}
\mathtt{M} &= \mathtt{M}_2 \\
\mathtt{VO}_I &= \mathtt{VO}_V + i \times (\mathtt{UB}_2 - \mathtt{LB}_2 + 1) \times \mathtt{M}_2 \\
&= \mathtt{VO}_V + i \times \mathtt{M}_1 \\
\mathtt{LB} &= \mathtt{LB}_2 \\
\mathtt{UB} &= \mathtt{UB}_2 \\
\Lambda\|I[k]\| &= \mathtt{VO}_I + k \times \mathtt{M}
\end{aligned}
$$

$$
\mathtt{I} = \mathtt{V}\,\mathtt{[i]}\,\mathtt{[*]} = V[i][\mathtt{LB}_2], V[i][\mathtt{LB}_2 + 1], \ldots, V[i][\mathtt{UB}_2]
$$

Slices of array



$$\mathtt{M} = (\mathtt{UB}_2 - \mathtt{LB}_2 + 1) \times \mathtt{M}_2 = \mathtt{M}_1$$

$$\mathtt{VO}_J = \mathtt{VO}_V + j \times \mathtt{M}_2$$

$$\mathtt{LB} = \mathtt{LB}_1$$

$$\mathtt{UB} = \mathtt{UB}_1$$

$$\Lambda\|J[k]\| = \mathtt{VO}_J + k \times \mathtt{M}$$

$$\mathtt{I} = \mathtt{V}\,[\mathtt{i}]\,[*] = V[i][\mathtt{LB}_2], V[i][\mathtt{LB}_2 + 1], \ldots, V[i][\mathtt{UB}_2]$$

$$\mathtt{J} = \mathtt{V}\,[*]\,[\mathtt{j}] = V[\mathtt{LB}_1][j], V[\mathtt{LB}_1 + 1][j], \ldots, V[\mathtt{UB}_2][j]$$

Outline

1 Definition

2 Arrays in crème CAraMeL

3 Bidimensional Matrices

4 Multidimensional matrices

5 Slices

6 Implementation

Implementation of array in crème CAraMeL

Syntax

parser.mly: new token `ARRAY`, `OF`, `LBRACKET`, `RBRACKET`, `DOTS`

lexer.mll: strings corresponding to new tokens

syntaxtree.ml: constructors

- `Vector of bType * int * int` for declaration
  `var v:array [0..6] of int`
- `LVec of ide * aexp` for the left side of the assignment
  `v[0]:=5;`
- `Vec of ide * aexp` for expressions
  `x:= v[2];`

parser.mly: productions for constructing new nodes of a.s.t.

Implementation of array in crème CAraMeL

Syntax

parser.mly: new token `ARRAY`, `OF`, `LBRACKET`, `RBRACKET`, `DOTS`

lexer.mll: strings corresponding to new tokens

syntaxtree.ml: constructors

- `Vector of bType * int * int` for declaration
  `var v:array [0..6] of int`
- `LVec of ide * aexp` for the left side of the assignment
  `v[0]:=5;`
- `Vec of ide * aexp` for expressions
  `x:= v[2];`

parser.mly: productions for constructing new nodes of a.s.t.

Implementation of array in crème CAraMeL

Syntax

parser.mly: new token `ARRAY`, `OF`, `LBRACKET`, `RBRACKET`, `DOTS`

lexer.mll: strings corresponding to new tokens

syntaxtree.ml: constructors

- `Vector of bType * int * int` for declaration
  `var v:array [0..6] of int`
- `LVec of ide * aexp` for the left side of the assignment
  `v[0]:=5;`
- `Vec of ide * aexp` for expressions
  `x:= v[2];`

parser.mly: productions for constructing new nodes of a.s.t.

Implementation of array in crème CAraMeL

Syntax

parser.mly: new token `ARRAY`, `OF`, `LBRACKET`, `RBRACKET`, `DOTS`

lexer.mll: strings corresponding to new tokens

syntaxtree.ml: constructors

- `Vector of bType * int * int` for declaration
  `var v:array [0..6] of int`
- `LVec of ide * aexp` for the left side of the assignment
  `v[0]:=5;`
- `Vec of ide * aexp` for expressions
  `x:= v[2];`

parser.mly: productions for constructing new nodes of a.s.t.

Implementation of array in crème CAraMeL

Semantics - `interpreter.ml`

- New value for the environment: `Descr_Vector of loc * int * int` (VO, LB, UB)
- Declaration with initialization to 0 (or 0.)
- Evaluation of expression (r-value)
- Evaluation of the address (l-value)

Implementation of array in crème CAraMeL

Semantics - `interpreter.ml`

- New value for the environment: `Descr_Vector of loc * int * int` (VO, LB, UB)
- Declaration with initialization to 0 (or 0.)
- Evaluation of expression (r-value)
- Evaluation of the address (l-value)

Implementation of array in crème CAraMeL

Semantics - `interpreter.ml`

- New value for the environment: `Descr_Vector of loc * int * int` (VO, LB, UB)
- Declaration with initialization to 0 (or 0.)
- Evaluation of expression (r-value)
- Evaluation of the address (l-value)

Implementation of array in crème CAraMeL

Semantics - `interpreter.ml`

- New value for the environment: `Descr_Vector of loc * int * int` (VO, LB, UB)
- Declaration with initialization to 0 (or 0.)
- Evaluation of expression (r-value)
- Evaluation of the address (l-value)