The Delinearization of *C* programs

Chirantan Mukherjee, Marc Moreno Maza, Linxiao Wang

Ontario Research Centre for Computer Algebra

April 9, 2024

cmukher@uwo.ca

Content

Linearized multi-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    A[i * n + j] = A[(n * j - n + j - i - 1];
```

Linearized multi-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    A[i * n + j] = A[(n * j - n + j - i - 1];
```

Can we parallelize the two for-loops?

We need to see whether there is data dependence between two different iterations of the nest.

Linearized multi-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    A[i * n + j] = A[(n * j - n + j - i - 1];
```

Can we parallelize the two for-loops?

We need to see whether there is data dependence between two different iterations of the nest.

Are there integer solutions to the following system of linear inequalities:

$$\begin{cases} 0 \leq i_1 < n \\ i_1 + 1 \leq j_1 < n \\ 0 \leq i_2 < n \\ i_2 + 1 \leq j_2 < n \\ i_1 \times n + j_1 = n \times j_2 - n + j_2 - i_2 - 1 \end{cases}$$

Delinearize the array accesses

Linearized one-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    A[i * n + j] =
        A[(n * j - n + j - i - 1];
```

Delinearize the array accesses

Linearized one-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    A[i * n + j] =
        A[(n * j - n + j - i - 1];
```

Delinearized multi-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    B[i][j] = B[j - 1][j - i - 1];
```

Delinearize the array accesses

Linearized one-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    A[i * n + j] =
        A[(n * j - n + j - i - 1];
```

Delinearized multi-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    B[i][j] = B[j - 1][j - i - 1];
```

$$\begin{cases} 0 \leq i_1 < n \\ i_1 + 1 \leq j_1 < n \\ 0 \leq i_2 < n \\ i_2 + 1 \leq j_2 < n \\ i_1 \times n + j_1 = n \times j_2 - n + j_2 - i_2 - 1 \end{cases}$$

Delinearize the array accesses

Linearized one-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    A[i * n + j] =
        A[(n * j - n + j - i - 1];
```

$$\begin{cases} 0 \leq i_1 < n \\ i_1 + 1 \leq j_1 < n \\ 0 \leq i_2 < n \\ i_2 + 1 \leq j_2 < n \\ i_1 \times n + j_1 = n \times j_2 - n + j_2 - i_2 - 1 \end{cases}$$

Delinearized multi-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    B[i][j] = B[j - 1][j - i - 1];
```

$$\begin{cases} 0 \leq i_1 < n \\ i_1 + 1 \leq j_1 < n \\ 0 \leq i_2 < n \\ i_2 + 1 \leq j_2 < n \\ i_1 = j_2 - 1 \\ j_1 = j_2 - i_2 - 1 \end{cases}$$

Delinearize the array accesses

Linearized one-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    A[i * n + j] =
        A[(n * j - n + j - i - 1];
```

$$\begin{cases} 0 \leq i_1 < n \\ i_1 + 1 \leq j_1 < n \\ 0 \leq i_2 < n \\ i_2 + 1 \leq j_2 < n \\ i_1 \times n + j_1 = n \times j_2 - n + j_2 - i_2 - 1 \end{cases}$$

Delinearized multi-dimensional array

```
for(int i = 0; i < n; i++)
  for(int j = i + 1; j < n; j ++)
    B[i][j] = B[j - 1][j - i - 1];
```

$$\begin{cases} 0 \leq i_1 < n \\ i_1 + 1 \leq j_1 < n \\ 0 \leq i_2 < n \\ i_2 + 1 \leq j_2 < n \\ i_1 = j_2 - 1 \\ j_1 = j_2 - i_2 - 1 \end{cases}$$

There is no integer solution, therefore, no dependence.

Can we replace a one-dimension array `A` by another multi-dimensional array `B` so that every subscript of `B` is an affine expression of the loop counters?

Can we replace a one-dimension array `A` by another multi-dimensional array `B` so that every subscript of `B` is an affine expression of the loop counters?

Input

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   A[R(i_1, ..., i_d, m_1, ..., m_e)] <- ...
```

Can we replace a one-dimension array `A` by another multi-dimensional array `B` so that every subscript of `B` is an affine expression of the loop counters?

Input

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   A[R(i_1, ..., i_d, m_1, ..., m_e)] <- ...
```

- $i_1, \ldots, i_d$ take non-negative integer values

$$L \begin{pmatrix} i_1 \\ \vdots \\ i_d \end{pmatrix} \leq \begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix}$$

Can we replace a one-dimension array `A` by another multi-dimensional array `B` so that every subscript of `B` is an affine expression of the loop counters?

Input

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   A[R(i_1, ..., i_d, m_1, ..., m_e)] <- ...
```

- $i_1, \ldots, i_d$ take non-negative integer values

$$L \begin{pmatrix} i_1 \\ \vdots \\ i_d \end{pmatrix} \leq \begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix}$$

- L is a lower-triangular full-rank matrix over $\mathbb{Z}$ (known at compile time) defining the iteration domain

Can we replace a one-dimension array `A` by another multi-dimensional array `B` so that every subscript of `B` is an affine expression of the loop counters?

Input

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   A[R(i_1, ..., i_d, m_1, ..., m_e)] <- ...
```

- $i_1, \ldots, i_d$ take non-negative integer values

$$L \begin{pmatrix} i_1 \\ \vdots \\ i_d \end{pmatrix} \leq \begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix}$$

- L is a lower-triangular full-rank matrix over $\mathbb{Z}$ (known at compile time) defining the iteration domain
- $m_1, \ldots, m_e, r_1, \ldots, r_d$: data parameters (known only at execution time)

Can we replace a one-dimension array `A` by another multi-dimensional array `B` so that every subscript of `B` is an affine expression of the loop counters?

Input

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   A[R(i_1, ..., i_d, m_1, ..., m_e)] <- ...
```

- $i_1, \ldots, i_d$ take non-negative integer values

$$L \begin{pmatrix} i_1 \\ \vdots \\ i_d \end{pmatrix} \leq \begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix}$$

- L is a lower-triangular full-rank matrix over $\mathbb{Z}$ (known at compile time) defining the iteration domain
- $m_1, \ldots, m_e, r_1, \ldots, r_d$: data parameters (known only at execution time)
- $R(i_1, \ldots, i_d, m_1, \ldots, m_e)$ is a polynomial, whose coefficients are known at compile time

Can we replace a one-dimension array `A` by another multi-dimensional array `B` so that every subscript of `B` is an affine expression of the loop counters?

Input

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   A[R(i_1, ..., i_d, m_1, ..., m_e)] <- ...
```

- $i_1, \ldots, i_d$ take non-negative integer values

$$L \begin{pmatrix} i_1 \\ \vdots \\ i_d \end{pmatrix} \leq \begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix}$$

- L is a lower-triangular full-rank matrix over $\mathbb{Z}$ (known at compile time) defining the iteration domain
- $m_1, \ldots, m_e, r_1, \ldots, r_d$: data parameters (known only at execution time)
- $R(i_1, \ldots, i_d, m_1, \ldots, m_e)$ is a polynomial, whose coefficients are known at compile time

Output

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   B[f_1], ..., B[f_e] <- ...
```

Can we replace a one-dimension array `A` by another multi-dimensional array `B` so that every subscript of `B` is an affine expression of the loop counters?

Input

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   A[R(i_1, ..., i_d, m_1, ..., m_e)] <- ...
```

- $i_1, \ldots, i_d$ take non-negative integer values

$$L \begin{pmatrix} i_1 \\ \vdots \\ i_d \end{pmatrix} \leq \begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix}$$

- L is a lower-triangular full-rank matrix over $\mathbb{Z}$ (known at compile time) defining the iteration domain
- $m_1, \ldots, m_e, r_1, \ldots, r_d$: data parameters (known only at execution time)
- $R(i_1, \ldots, i_d, m_1, \ldots, m_e)$ is a polynomial, whose coefficients are known at compile time

Output

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   B[f_1], ..., B[f_e] <- ...
```

- $f_1, \ldots, f_e$ are affine forms in $i_1, \ldots, i_d$ the coefficients of which are integers to-be-determined

Can we replace a one-dimension array `A` by another multi-dimensional array `B` so that every subscript of `B` is an affine expression of the loop counters?

Input

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   A[R(i_1, ..., i_d, m_1, ..., m_e)] <- ...
```

- $i_1, \ldots, i_d$ take non-negative integer values

$$L \begin{pmatrix} i_1 \\ \vdots \\ i_d \end{pmatrix} \leq \begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix}$$

- L is a lower-triangular full-rank matrix over $\mathbb{Z}$ (known at compile time) defining the iteration domain
- $m_1, \ldots, m_e, r_1, \ldots, r_d$: data parameters (known only at execution time)
- $R(i_1, \ldots, i_d, m_1, \ldots, m_e)$ is a polynomial, whose coefficients are known at compile time

Output

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   B[f_1], ..., B[f_e] <- ...
```

- $f_1, \ldots, f_e$ are affine forms in $i_1, \ldots, i_d$ the coefficients of which are integers to-be-determined
- B is an $M_1 \times \cdots \times M_e$-array

Can we replace a one-dimension array `A` by another multi-dimensional array `B` so that every subscript of `B` is an affine expression of the loop counters?

Input

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   A[R(i_1, ..., i_d, m_1, ..., m_e)] <- ...
```

- $i_1, \ldots, i_d$ take non-negative integer values

$$L \begin{pmatrix} i_1 \\ \vdots \\ i_d \end{pmatrix} \leq \begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix}$$

- L is a lower-triangular full-rank matrix over $\mathbb{Z}$ (known at compile time) defining the iteration domain
- $m_1, \ldots, m_e, r_1, \ldots, r_d$: data parameters (known only at execution time)
- $R(i_1, \ldots, i_d, m_1, \ldots, m_e)$ is a polynomial, whose coefficients are known at compile time

Output

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   B[f_1], ..., B[f_e] <- ...
```

- $f_1, \ldots, f_e$ are affine forms in $i_1, \ldots, i_d$ the coefficients of which are integers to-be-determined
- B is an $M_1 \times \cdots \times M_e$-array
- $M_1, \ldots, M_e$ are affine forms in $m_1, \ldots, m_d$ the coefficients of which are integers to-be-determined

Can we replace a one-dimension array A by another multi-dimensional array B so that every subscript of B is an affine expression of the loop counters?

Input

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   A[R(i_1, ..., i_d, m_1, ..., m_e)] <- ...
```

- $i_1, \ldots, i_d$ take non-negative integer values

$$L \begin{pmatrix} i_1 \\ \vdots \\ i_d \end{pmatrix} \leq \begin{pmatrix} r_1 \\ \vdots \\ r_d \end{pmatrix}$$

- L is a lower-triangular full-rank matrix over $\mathbb{Z}$ (known at compile time) defining the iteration domain
- $m_1, \ldots, m_e, r_1, \ldots, r_d$: data parameters (known only at execution time)
- $R(i_1, \ldots, i_d, m_1, \ldots, m_e)$ is a polynomial, whose coefficients are known at compile time

Output

```
for(i_1, ..., i_1 ++)
 ...
  for(i_d, ..., i_d ++)
   B[f_1], ..., B[f_e] <- ...
```

- $f_1, \ldots, f_e$ are affine forms in $i_1, \ldots, i_d$ the coefficients of which are integers to-be-determined
- B is an $M_1 \times \cdots \times M_e$-array
- $M_1, \ldots, M_e$ are affine forms in $m_1, \ldots, m_d$ the coefficients of which are integers to-be-determined

such that:

$$\boxed{R = f_1 M_2 \cdots M_e \ + \ \cdots \ + \ f_{e-1} M_2 + f_e}$$

holds and for each $(i_1, \ldots, i_d)$ in the iteration domain we have the validity conditions:

$$\boxed{0 \leq f_1 < M_1, \quad \ldots, 0 \leq f_e < M_e.}$$

The delinearization problem can be divided into:

1. Polynomial System Solving Problem expressing $f_1, \ldots, f_e$ and $M_1, \ldots, M_e$ offline as coefficients of $R = f_1 M_2 \cdots M_e + \cdots + f_{e-1} M_2 + f_e$,

The delinearization problem can be divided into:

1. Polynomial System Solving Problem expressing $f_1, \ldots, f_e$ and $M_1, \ldots, M_e$ offline as coefficients of $R = f_1 M_2 \cdots M_e + \cdots + f_{e-1} M_2 + f_e$, when d and e are known.
2. Quantifier Elimination Problem for each $(i_1, \ldots, i_d)$ in the iteration domain we obtain the validity conditions $0 \leq f_1 < M_1, \quad \ldots, 0 \leq f_e < M_e$.

The delinearization problem can be divided into:

1. Polynomial System Solving Problem expressing $f_1, \ldots, f_e$ and $M_1, \ldots, M_e$ offline as coefficients of $R = f_1 M_2 \cdots M_e + \cdots + f_{e-1} M_2 + f_e$, when d and e are known.
2. Quantifier Elimination Problem for each $(i_1, \ldots, i_d)$ in the iteration domain we obtain the validity conditions $0 \leq f_1 < M_1, \quad \ldots, 0 \leq f_e < M_e$. This non-linear QE problem can only be solved offline over $\mathbb{R}$

The delinearization problem can be divided into:

1. Polynomial System Solving Problem expressing $f_1, \ldots, f_e$ and $M_1, \ldots, M_e$ offline as coefficients of $R = f_1 M_2 \cdots M_e + \cdots + f_{e-1} M_2 + f_e$, when d and e are known.
2. Quantifier Elimination Problem for each $(i_1, \ldots, i_d)$ in the iteration domain we obtain the validity conditions $0 \leq f_1 < M_1, \ \ldots, 0 \leq f_e < M_e$. This non-linear QE problem can only be solved offline over $\mathbb{R}$ and then can be reduced to Presburger arithmetic (that is, QE on affine forms over $\mathbb{Z}$).

The delinearization problem can be divided into:

1. Polynomial System Solving Problem expressing $f_1, \ldots, f_e$ and $M_1, \ldots, M_e$ offline as coefficients of $R = f_1 M_2 \cdots M_e + \cdots + f_{e-1} M_2 + f_e$, when d and e are known.
2. Quantifier Elimination Problem for each $(i_1, \ldots, i_d)$ in the iteration domain we obtain the validity conditions $0 \le f_1 < M_1, \ \ldots, 0 \le f_e < M_e$. This non-linear QE problem can only be solved offline over $\mathbb{R}$ and then can be reduced to Presburger arithmetic (that is, QE on affine forms over $\mathbb{Z}$).

Parametric Integer Linear Programming

$$\begin{array}{ll} \max\limits_{(i_1,\ldots,i_d)} & f_k \\ \text{subject to} & (i_1, \ldots, i_d) \in \text{iteration domain} \\ & i_1, \ldots, i_d \in \mathbb{Z} \end{array}$$

For which, $0 \le \max\limits_{(i_1,\ldots,i_d)} f_k < M_k$ for all $k \in [1, \ldots, e]$.

2D-2D delinearization problem

$d = 2$, means there are two loop counter i_1, i_2 known at compile time.

2D-2D delinearization problem

$d = 2$, means there are two loop counter i_1, i_2 known at compile time.
$e = 2$, means there are two program parameters m_1, m_2 also known at compile time.

2D-2D delinearization problem

$d = 2$, means there are two loop counter i_1, i_2 known at compile time.
$e = 2$, means there are two program parameters m_1, m_2 also known at compile time.
Should return two polynomials $M_1 = a_1 m_1 + b_1$ and $M_2 = a_2 m_2 + b_2$, where a_1, b_1, a_2, b_2 are integers to-be-determined.

2D-2D delinearization problem

$d = 2$, means there are two loop counter i_1, i_2 known at compile time.
$e = 2$, means there are two program parameters m_1, m_2 also known at compile time.
Should return two polynomials $M_1 = a_1 m_1 + b_1$ and $M_2 = a_2 m_2 + b_2$, where a_1, b_1, a_2, b_2 are integers to-be-determined.
The reference $A[R]$ to A which encodes a reference $B[f_1][f_2]$ to B, where:

2D-2D delinearization problem

$d = 2$, means there are two loop counter i_1, i_2 known at compile time.
$e = 2$, means there are two program parameters m_1, m_2 also known at compile time.
Should return two polynomials $M_1 = a_1 m_1 + b_1$ and $M_2 = a_2 m_2 + b_2$, where a_1, b_1, a_2, b_2 are integers to-be-determined.
The reference $A[R]$ to A which encodes a reference $B[f_1][f_2]$ to B, where:

- $f1 = f_{11} i_1 + f_{12} i_2 + f_{10}$ and $f2 = f_{21} i_1 + f_{22} i_2 + f_{20}$ are affine functions

2D-2D delinearization problem

$d = 2$, means there are two loop counter i_1, i_2 known at compile time.
$e = 2$, means there are two program parameters m_1, m_2 also known at compile time.
Should return two polynomials $M_1 = a_1 m_1 + b_1$ and $M_2 = a_2 m_2 + b_2$, where a_1, b_1, a_2, b_2 are integers to-be-determined.
The reference $A[R]$ to A which encodes a reference $B[f_1][f_2]$ to B, where:

- $f1 = f_{11} i_1 + f_{12} i_2 + f_{10}$ and $f2 = f_{21} i_1 + f_{22} i_2 + f_{20}$ are affine functions
- $R = f1 M_2 + f2$ is a polynomial

2D-2D delinearization problem

$d = 2$, means there are two loop counter i_1, i_2 known at compile time.
$e = 2$, means there are two program parameters m_1, m_2 also known at compile time.
Should return two polynomials $M_1 = a_1 m_1 + b_1$ and $M_2 = a_2 m_2 + b_2$, where a_1, b_1, a_2, b_2 are integers to-be-determined.
The reference $A[R]$ to A which encodes a reference $B[f_1][f_2]$ to B, where:

- $f1 = f_{11} i_1 + f_{12} i_2 + f_{10}$ and $f2 = f_{21} i_1 + f_{22} i_2 + f_{20}$ are affine functions
- $R = f1 M_2 + f2$ is a polynomial
- for each (i_1, i_2), we have, the validity conditions $0 \le f1 < M_1$ and $0 \le f2 < M_2$, and $0 \le \max f1 < M_1$ and $0 \le \max f2 < M_2$.

2D-2D polynomial system solving

Substituting $f1$ and $f2$ in R, we obtain, $R = \underbrace{a_2 f_{11}}_{T_1} i_1 m_2 + \underbrace{a_2 f_{12}}_{T_2} i_2 m_2 + \underbrace{a_2 f_{10}}_{T_3} m_2 + (\underbrace{b_2 f_{11} + f_{21}}_{T_4}) i_1 + (\underbrace{b_2 f_{12} + f_{22}}_{T_5}) i_2 + (\underbrace{b_2 f_{10} + f_{20}}_{T_6})$.

2D-2D polynomial system solving

Substituting $f1$ and $f2$ in R, we obtain, $R = \underbrace{a_2 f_{11}}_{T_1} i_1 m_2 + \underbrace{a_2 f_{12}}_{T_2} i_2 m_2 + \underbrace{a_2 f_{10}}_{T_3} m_2 + (\underbrace{b_2 f_{11} + f_{21}}_{T_4}) i_1 + (\underbrace{b_2 f_{12} + f_{22}}_{T_5}) i_2 + (\underbrace{b_2 f_{10} + f_{20}}_{T_6})$.

$$\begin{cases} T_1 = a_2 f_{11} \\ T_2 = a_2 f_{12} \\ T_3 = a_2 f_{10} \\ T_4 = b_2 f_{11} + f_{21} \\ T_5 = b_2 f_{12} + f_{22} \\ T_6 = b_2 f_{10} + f_{20} \end{cases} \implies \begin{cases} f_{11} = \dfrac{T_1}{a_2} \\ f_{12} = \dfrac{T_2}{a_2} \\ f_{10} = \dfrac{T_3}{a_2} \\ f_{21} = T_4 - b_2 f_{11} \\ f_{22} = T_5 - b_2 f_{12} \\ f_{20} = T_6 - b_2 f_{10} \end{cases}$$

2D-2D polynomial system solving

Substituting $f1$ and $f2$ in R, we obtain, $R = \underbrace{a_2 f_{11}}_{T_1} i_1 m_2 + \underbrace{a_2 f_{12}}_{T_2} i_2 m_2 + \underbrace{a_2 f_{10}}_{T_3} m_2 + (\underbrace{b_2 f_{11} + f_{21}}_{T_4}) i_1 + (\underbrace{b_2 f_{12} + f_{22}}_{T_5}) i_2 + (\underbrace{b_2 f_{10} + f_{20}}_{T_6})$.

$$\begin{cases} T_1 = a_2 f_{11} \\ T_2 = a_2 f_{12} \\ T_3 = a_2 f_{10} \\ T_4 = b_2 f_{11} + f_{21} \\ T_5 = b_2 f_{12} + f_{22} \\ T_6 = b_2 f_{10} + f_{20} \end{cases} \implies \begin{cases} f_{11} = \dfrac{T_1}{a_2} \\ f_{12} = \dfrac{T_2}{a_2} \\ f_{10} = \dfrac{T_3}{a_2} \\ f_{21} = T_4 - b_2 f_{11} \\ f_{22} = T_5 - b_2 f_{12} \\ f_{20} = T_6 - b_2 f_{10} \end{cases}$$

a_2 and b_2 can not be uniquely determined,

2D-2D polynomial system solving

Substituting $f1$ and $f2$ in R, we obtain, $R = \underbrace{a_2 f_{11}}_{T_1} i_1 m_2 + \underbrace{a_2 f_{12}}_{T_2} i_2 m_2 + \underbrace{a_2 f_{10}}_{T_3} m_2 + (\underbrace{b_2 f_{11} + f_{21}}_{T_4}) i_1 + (\underbrace{b_2 f_{12} + f_{22}}_{T_5}) i_2 + (\underbrace{b_2 f_{10} + f_{20}}_{T_6})$.

$$\begin{cases} T_1 = a_2 f_{11} \\ T_2 = a_2 f_{12} \\ T_3 = a_2 f_{10} \\ T_4 = b_2 f_{11} + f_{21} \\ T_5 = b_2 f_{12} + f_{22} \\ T_6 = b_2 f_{10} + f_{20} \end{cases} \implies \begin{cases} f_{11} = \dfrac{T_1}{a_2} \\ f_{12} = \dfrac{T_2}{a_2} \\ f_{10} = \dfrac{T_3}{a_2} \\ f_{21} = T_4 - b_2 f_{11} \\ f_{22} = T_5 - b_2 f_{12} \\ f_{20} = T_6 - b_2 f_{10} \end{cases}$$

a_2 and b_2 can not be uniquely determined, but $a_2 \mid gcd(T_1, T_2, T_3)$.

2D-2D quantifier elimination (I/II)

Recall for each (i_1, i_2), we have the validity condition, $0 \leq f2 < M_2$ and $0 \leq \max f2 < M_2$, that is,

$$\begin{cases} \max\limits_{(i_1, i_2)} & f_2 \\ \text{subject to} & (i_1, i_2) \in \text{iteration domain} \\ & i_1, i_2 \in \mathbb{Z} \end{cases}$$

Depending on the shape of the iteration domain, we solve on a case to case basis.

2D-2D quantifier elimination (I/II) - Rectangular domain

For a `for loop` of the form,

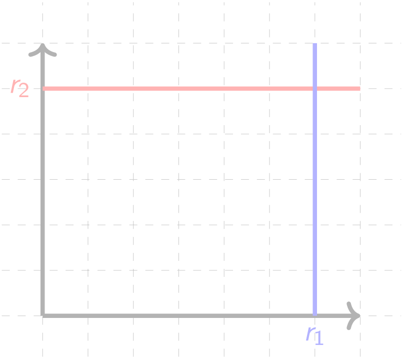
```
for (int i_1 = 0; i_1 < r_1; i_1 ++)
  for (int i_2 = 0; i_2 < r_2; i_2 ++)
```

2D-2D quantifier elimination (I/II) - Rectangular domain

For a `for loop` of the form,

```
for (int i_1 = 0; i_1 < r_1; i_1 ++)
  for (int i_2 = 0; i_2 < r_2; i_2 ++)
```

the iteration domain is of the shape,



2D-2D quantifier elimination (I/II) - Triangular domain

For a `for loop` of the form,

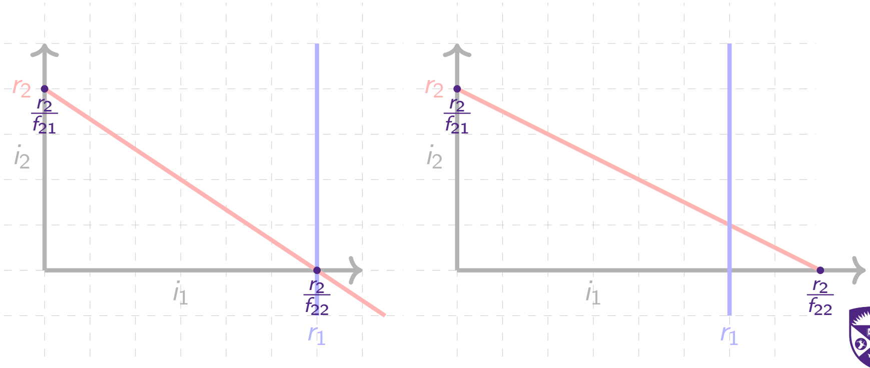
```
for (int i_1 = 0; i_1 < r_1; i_1 ++)
  for (int i_2 = 0; f_{21} * i_1 + f_{22} * i_2 < r_2; i_2 ++)
```

2D-2D quantifier elimination (I/II) - Triangular domain

For a `for loop` of the form,

```
for (int i_1 = 0; i_1 < r_1; i_1 ++)
  for (int i_2 = 0; f_{21} * i_1 + f_{22} * i_2 < r_2; i_2 ++)
```

the iteration domain is of the shape,



Notice, that $i_2 = \frac{-i_1 f_{22} + r_2}{f_{21}}$

2D-2D quantifier elimination (I/II)

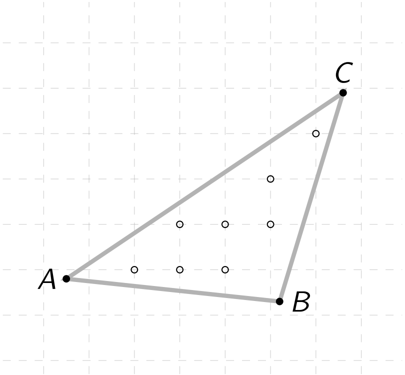
The parametric integer linear problem can be solved for:

1. Rectangular domain by case inspection
2. Triangular domain by case inspection except when `f_21, f_22 > 0`, in which case the problem becomes,

$$\begin{array}{lll} \max\limits_{i_1} & f_{21} i_1 + f_{22} \left\lfloor \frac{-i_1 f_{22} + r_2}{f_{21}} \right\rfloor + f_{20} & \\ \text{subject to} & 0 \leq i_1 < r_1, & i_1 \in \mathbb{Z} \end{array}$$

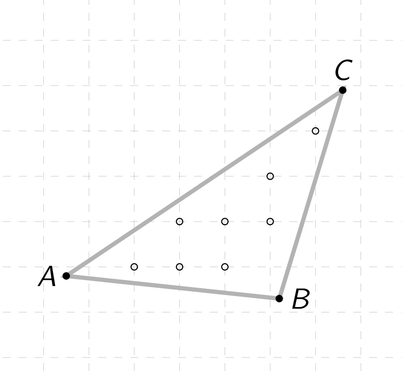
2D-2D quantifier elimination (I/II)

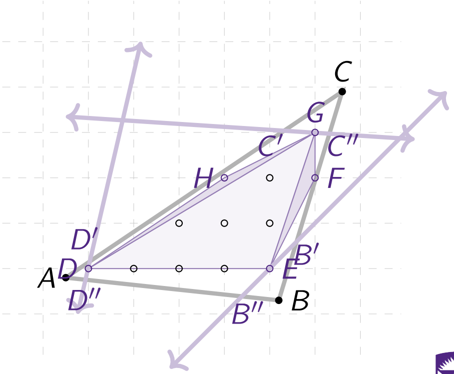
Since the loop counters can only be integers this leads to the problem of finding the integer hull of a polyhedral set, for which we propose the following `integer hull algorithm`,



2D-2D quantifier elimination (I/II)

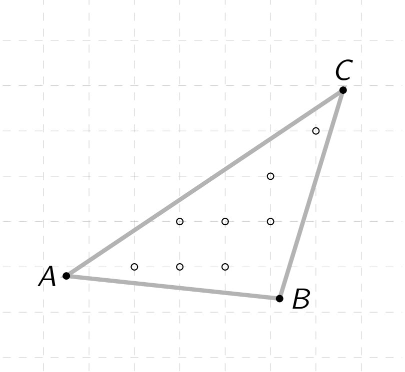
Since the loop counters can only be integers this leads to the problem of finding the integer hull of a polyhedral set, for which we propose the following `integer hull algorithm`,

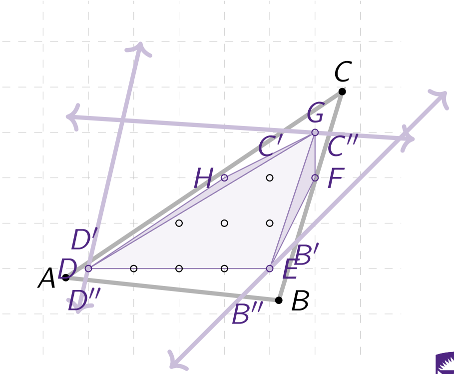




2D-2D quantifier elimination (I/II)

Since the loop counters can only be integers this leads to the problem of finding the integer hull of a polyhedral set, for which we propose the following `integer hull algorithm`,





The integer hull $\{D, E, F, G, H\}$ is formed by $\{D, E, G\}$ and searching integer points $\{F, H\}$ in quadrilaterals $DD''B''E$, $EB'C''G$ and $D'DGC'$.

2D-2D quantifier elimination (II/II)

Another approach is to do QE over $\mathbb{R}$,

2D-2D quantifier elimination (II/II)

Another approach is to do QE over $\mathbb{R}$, for which we can use the function `QuantifierElimination` from the `SemiAlgebraicSetTools` package, which is a subpackage of the `RegularChains` library.

2D-2D quantifier elimination (II/II)

Another approach is to do QE over $\mathbb{R}$, for which we can use the function `QuantifierElimination` from the `SemiAlgebraicSetTools` package, which is a subpackage of the `RegularChains` library.

```
f := &A([i_1, i_2]), ((0 < i_1) &and (i_1 < r_1) &and
                      (0 < i_2) &and (i_2 < r_2) &and
                      (0 < r_1) &and (0 < r_2) &and
                      (0 < f_{21}) &and (0 < f_{22}) &and (0 < B))
                      \\ B = M_2 - f_{20}
           &implies (f_{21} * i_1 + f_{22} * i_2 < B);
```

2D-2D quantifier elimination (II/II)

Another approach is to do QE over $\mathbb{R}$, for which we can use the function `QuantifierElimination` from the `SemiAlgebraicSetTools` package, which is a subpackage of the `RegularChains` library.

```
f := &A([i_1, i_2]), ((0 < i_1) &and (i_1 < r_1) &and
                      (0 < i_2) &and (i_2 < r_2) &and
                      (0 < r_1) &and (0 < r_2) &and
                      (0 < f_{21}) &and (0 < f_{22}) &and (0 < B))
                      \\ B = M_2 - f_{20}
            &implies (f_{21} * i_1 + f_{22} * i_2 < B);
```

After simplification, `r_1 * f_{21} + r_2 * f_{22} + f_{20} < M_2`.

2D-2D quantifier elimination (II/II)

Another approach is to do QE over $\mathbb{R}$, for which we can use the function `QuantifierElimination` from the `SemiAlgebraicSetTools` package, which is a subpackage of the `RegularChains` library.

```
f := &A([i_1, i_2]), ((0 < i_1) &and (i_1 < r_1) &and
                      (0 < i_2) &and (i_2 < r_2) &and
                      (0 < r_1) &and (0 < r_2) &and
                      (0 < f_{21}) &and (0 < f_{22}) &and (0 < B))
                      \\ B = M_2 - f_{20}
          &implies (f_{21} * i_1 + f_{22} * i_2 < B);
```

After simplification, `r_1 * f_{21} + r_2 * f_{22} + f_{20} < M_2`.
Substituting, $f_{11} = \frac{T_1}{a_2}, f_{12} = \frac{T_2}{a_2}, f_{10} = \frac{T_3}{a_2}, f_{21} = T_4 - b_2 f_{11}, f_{22} = T_5 - b_2 f_{12}, f_{20} = T_6 - b_2 f_{10}, M_2 = a_2 m_2 + b_2$, we obtain,

$$\boxed{r_1\left(T_4 - b_2\frac{T_1}{a_2}\right) + r_2\left(T_5 - b_2\frac{T_2}{a_2}\right) + T_6 - b_2\frac{T_3}{a_2} < a_2 m_2 + b_2.}$$

Examples (I/II) - Rectangular domain

```
for (int i_1 = 0; i_1 <= r_1; i_1 ++)
  for (int i_2 = 0; i_2 <= r_2; i_2 ++)
    A[2 * i_1 * m_2 + m_2 + 3 * i_2 + 2] = ...;
```

1. $T_1 = 2, T_2 = 0, T_3 = 1, T_4 = 0, T_5 = 3, T_6 = 2$
2. $a_2 =?, b_2 =?, f_{11} = 2, f_{12} = 0, f_{10} = 1, f_{21} = -2b_2, f_{22} = 3, f_{20} = 2 - b_2$
3. the validity condition $-r_1 b_2 \frac{2}{a_2} + 3r_2 + 2 - b_2 \frac{1}{a_2} < a_2 m_2 + b_2$
4. evaluating at $a_2 = 1, b_2 = 0$, we obtain, $f1 = 2i_1 + 1, f2 = 3i_2 + 2$
5. $max\ i_1 = r_1, max\ i_2 = r_2$
6. assuming $m_2 = 10$, i.e. $B[...][10]$, we get,
   delinearization valid when $r_1 = r_2 = 1, max\ f2 = 5 < 10$
   delinearization valid when $r_1 = r_2 = 2, max\ f2 = 8 < 10$
   delinearization invalid when $r_1 = r_2 = 3, max\ f2 = 11 \not< 10$

Examples (II/II) - Triangular domain

```
for (int i_1 = 0; i_1 <= r_1; i_1 ++)
  for (int i_2 = 0; i_1 + 2 * i_2 <= r_2; i_2 ++)
    A[2 * i_1 * m_2 + m_2 + 3 * i_2 + 2] = ...;
```

1. $T_1 = 2, T_2 = 0, T_3 = 1, T_4 = 0, T_5 = 3, T_6 = 2$
2. $a_2 = ?, b_2 = ?, f_{11} = 2, f_{12} = 0, f_{10} = 1, f_{21} = -2b_2, f_{22} = 3, f_{20} = 2 - b_2$
3. the validity condition $-r_1 b_2 \frac{2}{a_2} + 3r_2 + 2 - b_2 \frac{1}{a_2} < a_2 m_2 + b_2$
4. evaluating at $a_2 = 1, b_2 = 0$, we obtain, $f1 = 2i_1 + 1, f2 = 3i_2 + 2$
5. $max\ i_1 = r_1, max\ i_2 = \lfloor \frac{r_2}{2} \rfloor$
6. assuming $m_2 = 10$, i.e. $B[...][10]$, we get,

   delinearization valid when $r_1 = r_2 = 1, max\ f2 = 2 < 10$
   delinearization valid when $r_1 = r_2 = 2, max\ f2 = 5 < 10$
   delinearization valid when $r_1 = r_2 = 3, max\ f2 = 5 < 10$
   delinearization valid when $r_1 = r_2 = 4, max\ f2 = 8 < 10$
   delinearization valid when $r_1 = r_2 = 5, max\ f2 = 8 < 10$
   delinearization valid when $r_1 = r_2 = 6, max\ f2 = 11 \not< 10$