

Presburger Sets and Relations: from High-level Modeling to Low-level Implementation

Sven Verdoolaege

École Normale Supérieure and INRIA
Sven.Verdoolaege@ens.fr

May 14, 2013

Introduction

A presentation about polyhedral **analysis**,
not about polyhedral **transformation**

Two parts

- ① Representation and Modeling
- ② Algorithms and Implementation

- no complexity
- no scheduling
- no motivation
- no matrices
- no polyhedra (in first part)

Question

Who knows?

Note

Personal observation.

Outline of Part I: Representation and Modeling

1 Introduction

2 Representation

3 Operations

- Basic Operations
- Application 1: Dependence Analysis
- Application 2: Array Dataflow Analysis
- Cardinality
- Application 3: Reuse Distance Computation
- Application 4: Maximal Number of Live Memory elements
- Nested Relations
- Application 5: Representing Dynamic Conditions
- Weighted Counting
- Application 6: Dynamic Memory Requirement Estimation
- Transitive Closures
- Application 7: Reachability Analysis

4 Availability

Outline of Part II: Algorithms and Implementation

5 Tools

6 Internal Representation

7 Weighted Counting

- Introduction
- Nested Sums
- Unweighted Counting
- Derivation
- Local Euler-Maclaurin Formulas
- Laurent Series Expansion
- Interpolation
- Set Operations

Part I

Representation and Modeling

Outline

- 1 Introduction
- 2 Representation

3 Operations

- Basic Operations
- Application 1: Dependence Analysis
- Application 2: Array Dataflow Analysis
- Cardinality
- Application 3: Reuse Distance Computation
- Application 4: Maximal Number of Live Memory elements
- Nested Relations
- Application 5: Representing Dynamic Conditions
- Weighted Counting
- Application 6: Dynamic Memory Requirement Estimation
- Transitive Closures
- Application 7: Reachability Analysis

4 Availability

Outline

- 1 Introduction
- 2 Representation
- 3 Operations
 - Basic Operations
 - Application 1: Dependence Analysis
 - Application 2: Array Dataflow Analysis
 - Cardinality
 - Application 3: Reuse Distance Computation
 - Application 4: Maximal Number of Live Memory elements
 - Nested Relations
 - Application 5: Representing Dynamic Conditions
 - Weighted Counting
 - Application 6: Dynamic Memory Requirement Estimation
 - Transitive Closures
 - Application 7: Reachability Analysis
- 4 Availability

Polyhedral Analysis

Key features

- instance based
 - ⇒ statement *instances*
 - ⇒ array *elements*
- compact representation based on polyhedra or similar objects
 - ⇒ Presburger sets and relations
 - ⇒ ...

Polyhedral Analysis

Key features

- instance based
 - ⇒ statement *instances*
 - ⇒ array *elements*
- compact representation based on polyhedra or similar objects
 - ⇒ Presburger sets and relations
 - ⇒ ...

Main constituents of program representation

- Iteration domain
 - ⇒ the set of all statement instances
- Access relations
 - ⇒ the array elements accessed by a statement instance
- Dependences
 - ⇒ the statement instances that depend on a statement instance
- Schedule
 - ⇒ the execution time of a statement instance

Illustrative Example

```
R:  h(A[2]);  
    for (int i = 0; i < 2; ++i)  
        for (int j = 0; j < 2; ++j)  
S:          A[i + j] = f(i, j);  
    for (int k = 0; k < 2; ++k)  
T:          g(A[k], A[0]);
```

Illustrative Example

```
R:  h(A[2]);  
    for (int i = 0; i < 2; ++i)  
        for (int j = 0; j < 2; ++j) • instance based  
S:          A[i + j] = f(i, j);  
    for (int k = 0; k < 2; ++k)  
T:          g(A[k], A[0]);
```

- Iteration domain (set of statement instances)

$$I = \{ R(); S(0, 0); S(0, 1); S(1, 0); S(1, 1); T(0); T(1) \}$$

Illustrative Example

```
R: h(A[2]);  
    for (int i = 0; i < 2; ++i)  
        for (int j = 0; j < 2; ++j) • instance based  
S:         A[i + j] = f(i, j);  
    for (int k = 0; k < 2; ++k)  
T:         g(A[k], A[0]);
```

- Iteration domain (set of statement instances)

$$I = \{ R(); S(0,0); S(0,1); S(1,0); S(1,1); T(0); T(1) \}$$

- Access relations (accessed array elements; W : write, R : read)

$$W = \{ S(0,0) \rightarrow A(0); S(0,1) \rightarrow A(1); S(1,0) \rightarrow A(1);
S(1,1) \rightarrow A(2) \}$$

$$R = \{ R() \rightarrow A(2); T(0) \rightarrow A(0); T(1) \rightarrow A(1); T(1) \rightarrow A(0) \}$$

Illustrative Example

```
R: h(A[2]);  
    for (int i = 0; i < 2; ++i)  
        for (int j = 0; j < 2; ++j) • instance based  
S:         A[i + j] = f(i, j);  
    for (int k = 0; k < 2; ++k)  
T:         g(A[k], A[0]);
```

- Iteration domain (set of statement instances)

$$I = \{ R(); S(0,0); S(0,1); S(1,0); S(1,1); T(0); T(1) \}$$

- Access relations (accessed array elements; W : write, R : read)

$$W = \{ S(0,0) \rightarrow A(0); S(0,1) \rightarrow A(1); S(1,0) \rightarrow A(1);
S(1,1) \rightarrow A(2) \}$$

$$R = \{ R() \rightarrow A(2); T(0) \rightarrow A(0); T(1) \rightarrow A(1); T(1) \rightarrow A(0) \}$$

- Schedule (execution time)

$$\{ R() \rightarrow (0); S(0,0) \rightarrow (1); S(0,1) \rightarrow (2); S(1,0) \rightarrow (3);
S(1,1) \rightarrow (4); T(0) \rightarrow (5); T(1) \rightarrow (6) \}$$

Illustrative Example

```
R: h(A[2]);  
    for (int i = 0; i < 2; ++i)  
        for (int j = 0; j < 2; ++j)  
            A[i + j] = f(i, j);  
    for (int k = 0; k < 2; ++k)  
        g(A[k], A[0]);
```

- instance based
- compact representation

- Iteration domain (set of statement instances)

$$I = \{ R(); S(0,0); S(0,1); S(1,0); S(1,1); T(0); T(1) \}$$

- Access relations (accessed array elements): $\{ R(); S(i,j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2; \}$

$$W = \{ S(0,0) \rightarrow A(0); S(0,1) \rightarrow A(1); S(1,0) \rightarrow A(1); \\ S(1,1) \rightarrow A(2) \}$$

$$R = \{ R() \rightarrow A(2); T(0) \rightarrow A(0); T(1) \rightarrow A(1); T(1) \rightarrow A(0) \}$$

- Schedule (execution time)

$$\{ R() \rightarrow (0); S(0,0) \rightarrow (1); S(0,1) \rightarrow (2); S(1,0) \rightarrow (3); \\ S(1,1) \rightarrow (4); T(0) \rightarrow (5); T(1) \rightarrow (6) \}$$

Illustrative Example

```
R: h(A[2]);
   for (int i = 0; i < 2; ++i)
       for (int j = 0; j < 2; ++j)
           A[i + j] = f(i, j);
S:   for (int k = 0; k < 2; ++k)
      g(A[k], A[0]);
T:
```

- instance based
- compact representation

- Iteration domain (set of statement instances)

$$I = \{ R(); S(0,0); S(0,1); S(1,0); S(1,1); T(0); T(1) \}$$

- Access relations (accessed array elements): $\{ R(); S(i,j) : 0 \leq i < 2 \wedge 0 \leq j < 2, T(k) : 0 \leq k < 2 \}$

$W = \text{Constraints}$

$S(1, "read off")$, or

$R = \{ P() \rightarrow (0), T(0) \rightarrow (1), A(0) \rightarrow (2), S(1,0) \rightarrow (3), S(1,1) \rightarrow (4), T(1) \rightarrow (5), T(0) \rightarrow (6) \}$

- Schedule (execution time)

$$\{ R() \rightarrow (0); S(0,0) \rightarrow (1); S(0,1) \rightarrow (2); S(1,0) \rightarrow (3); S(1,1) \rightarrow (4); T(0) \rightarrow (5); T(1) \rightarrow (6) \}$$

Illustrative Example

```
R: h(A[2]);  
    for (int i = 0; i < 2; ++i)  
        for (int j = 0; j < 2; ++j)  
            A[i + j] = f(i, j);  
S:    for (int k = 0; k < 2; ++k)  
T:        g(A[k], A[0]);
```

- instance based
- compact representation

- Iteration domain (set of statement instances)

$$I = \{ R(); S(i, j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2 \}$$

- Access relations (accessed array elements; W : write, R : read)

$$W = \{ S(0, 0) \rightarrow A(0); S(0, 1) \rightarrow A(1); S(1, 0) \rightarrow A(1); \\ S(1, 1) \rightarrow A(2) \}$$

$$R = \{ R() \rightarrow A(2); T(0) \rightarrow A(0); T(1) \rightarrow A(1); T(1) \rightarrow A(0) \}$$

- Schedule (execution time)

$$\{ R() \rightarrow (0); S(0, 0) \rightarrow (1); S(0, 1) \rightarrow (2); S(1, 0) \rightarrow (3); \\ S(1, 1) \rightarrow (4); T(0) \rightarrow (5); T(1) \rightarrow (6) \}$$

Illustrative Example

```
R: h(A[2]);
   for (int i = 0; i < 2; ++i)
       for (int j = 0; j < 2; ++j)
           A[i + j] = f(i, j);
S:   for (int k = 0; k < 2; ++k)
T:       g(A[k], A[0]);
```

- instance based
- compact representation

- Iteration domain (set of statement instances)

$$I = \{ R(); S(i, j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2 \}$$

- Access relations (accessed array elements; W : write, R : read)

$$W = \{ S(0, 0) \rightarrow A(0); S(0, 1) \rightarrow A(1); S(1, 0) \rightarrow A(1); \\ S(1, 1) \rightarrow A(2) \}$$

$$R = \{ \text{R}(\dots) \rightarrow A(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

- Schedule (execution time)

$$\{ R() \rightarrow (0); S(0, 0) \rightarrow (1); S(0, 1) \rightarrow (2); S(1, 0) \rightarrow (3); \\ S(1, 1) \rightarrow (4); T(0) \rightarrow (5); T(1) \rightarrow (6) \}$$

Illustrative Example

```
R: h(A[2]);  
    for (int i = 0; i < 2; ++i)  
        for (int j = 0; j < 2; ++j)  
S:            A[i + j] = f(i, j);  
    for (int k = 0; k < 2; ++k)  
T:            g(A[k], A[0]);
```

- instance based
- compact representation

- Iteration domain (set of statement instances)

$$I = \{ R(); S(i, j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2 \}$$

- Access relations (accessed array elements; W : write, R : read)

$$W = \{ S(i, j) \rightarrow A(i + j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

$$R = \{ R() \rightarrow A(2); T(0) \rightarrow A(0); T(1) \rightarrow A(1); T(1) \rightarrow A(0) \}$$

- Schedule (execution time)

$$\{ R() \rightarrow (0); S(0, 0) \rightarrow (1); S(0, 1) \rightarrow (2); S(1, 0) \rightarrow (3);
S(1, 1) \rightarrow (4); T(0) \rightarrow (5); T(1) \rightarrow (6) \}$$

Illustrative Example

```
R: h(A[2]);
   for (int i = 0; i < 2; ++i)
       for (int j = 0; j < 2; ++j)
           A[i + j] = f(i, j);
S:   for (int k = 0; k < 2; ++k)
T:       g(A[k], A[0]);
```

- instance based
- compact representation

- Iteration domain (set of statement instances)

$$I = \{ R(); S(i, j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2 \}$$

- Access relations (accessed array elements; W : write, R : read)

$$W = \{ S(i, j) \rightarrow A(i + j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

$$R = \{ R() \rightarrow A(2); T(0) \rightarrow A(0); T(1) \rightarrow A(1); T(1) \rightarrow A(0) \}$$

- Schedule $\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$
- $\{ R() \rightarrow (0); S(0, 0) \rightarrow (1); S(0, 1) \rightarrow (2); S(1, 0) \rightarrow (3);$
 $S(1, 1) \rightarrow (4); T(0) \rightarrow (5); T(1) \rightarrow (6) \}$

Illustrative Example

```
R: h(A[2]);  
    for (int i = 0; i < 2; ++i)  
        for (int j = 0; j < 2; ++j)  
            A[i + j] = f(i, j);  
    for (int k = 0; k < 2; ++k)  
        g(A[k], A[0]);
```

- instance based
- compact representation

- Iteration domain (set of statement instances)

$$I = \{ R(); S(i, j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2 \}$$

- Access relations (accessed array elements; W : write, R : read)

$$W = \{ S(i, j) \rightarrow A(i + j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

$$R = \{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$$

- Schedule (execution time)

$$\{ R() \rightarrow (0); S(0, 0) \rightarrow (1); S(0, 1) \rightarrow (2); S(1, 0) \rightarrow (3);
S(1, 1) \rightarrow (4); T(0) \rightarrow (5); T(1) \rightarrow (6) \}$$

Illustrative Example

```
R: h(A[2]);
   for (int i = 0; i < 2; ++i)
       for (int j = 0; j < 2; ++j)
           A[i + j] = f(i, j);
S:   for (int k = 0; k < 2; ++k)
T:     g(A[k], A[0]);
```

- instance based
- compact representation

- Iteration domain (set of statement instances)

$$I = \{ R(); S(i, j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2 \}$$

- Access relations (accessed array elements; W : write, R : read)

$$W = \{ S(i, j) \rightarrow A(i + j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

$$\{ R() \rightarrow (0); S(i, j) \rightarrow (1 + 2i + j) : 0 \leq i, j < 2; T(k) \rightarrow (5 + k) : 0 \leq k < 2 \}$$

- Schedule (execution time)

$$\{ R() \rightarrow (0); S(0, 0) \rightarrow (1); S(0, 1) \rightarrow (2); S(1, 0) \rightarrow (3);$$

$$S(1, 1) \rightarrow (4); T(0) \rightarrow (5); T(1) \rightarrow (6) \}$$

Illustrative Example

```
R: h(A[2]);
   for (int i = 0; i < 2; ++i)
       for (int j = 0; j < 2; ++j)
           A[i + j] = f(i, j);
S:   for (int k = 0; k < 2; ++k)
T:       g(A[k], A[0]);
```

- instance based
- compact representation

- Iteration domain (set of statement instances)

$$I = \{ R(); S(i, j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2 \}$$

- Access relations (accessed array elements; W : write, R : read)

$$W = \{ S(i, j) \rightarrow A(i + j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

$\{R() \rightarrow (0); S(i, j) \rightarrow (1 + 2i + j) : 0 \leq i, j < 2; T(k) \rightarrow (5 + k) : 0 \leq k < 2\}$

$\{R() \rightarrow (0, 0, 0); S(i, j) \rightarrow (1, i, j) : 0 \leq i, j < 2; T(k) \rightarrow (2, k, 0) : 0 \leq k < 2\}$

$\{R() \rightarrow (0); S(0, 0) \rightarrow (1); S(0, 1) \rightarrow (2); S(1, 0) \rightarrow (3);$

$S(1, 1) \rightarrow (4); T(0) \rightarrow (5); T(1) \rightarrow (6)\}$

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)  
    for (int j = 0; j < N; j++) {  
S1:    C[i][j] = 0;  
        for (int k = 0; k < K; k++)  
S2:        C[i][j] = C[i][j] + A[i][k] * B[k][j];  
    }
```

- Iteration domain (set of statement instances)

$$\{ S1(i,j) : 0 \leq i < M \wedge 0 \leq j < N;$$

$$S2(i,j,k) : 0 \leq i < M \wedge 0 \leq j < N \wedge 0 \leq k < K \}$$

- Access relations (accessed array elements; W : write, R : read)

$$W = \{ S1(i,j) \rightarrow C(i,j); S2(i,j,k) \rightarrow C(i,j) \}$$

$$R = \{ S2(i,j,k) \rightarrow C(i,j); S2(i,j,k) \rightarrow A(i,k); S2(i,j,k) \rightarrow B(k,j) \}$$

- Schedule (execution time)

$$\{ S1(i,j) \rightarrow (i,j,0,0); S2(i,j,k) \rightarrow (i,j,1,k) \}$$

Outline

- 1 Introduction
- 2 Representation

- 3 Operations
 - Basic Operations
 - Application 1: Dependence Analysis
 - Application 2: Array Dataflow Analysis
 - Cardinality
 - Application 3: Reuse Distance Computation
 - Application 4: Maximal Number of Live Memory elements
 - Nested Relations
 - Application 5: Representing Dynamic Conditions
 - Weighted Counting
 - Application 6: Dynamic Memory Requirement Estimation
 - Transitive Closures
 - Application 7: Reachability Analysis
- 4 Availability

Presburger Sets and Relations

Examples

$$\{ R(); S(i, j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2 \}$$

$$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$$

General form

- Sets

$$\{ S_1(\mathbf{i}) : f_1(\mathbf{i}); S_2(\mathbf{i}) : f_2(\mathbf{i}); \dots \},$$

with f_k Presburger formulas

⇒ set of elements of the form $S_1(\mathbf{i})$, one for each \mathbf{i} satisfying $f_1(\mathbf{i}), \dots$

- Binary relations

$$\{ S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j}) : f_1(\mathbf{i}, \mathbf{j}); S_2(\mathbf{i}) \rightarrow T_2(\mathbf{j}) : f_2(\mathbf{i}, \mathbf{j}); \dots \}$$

⇒ set of pairs of elements of the form $S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j})$

Presburger Sets and Relations

Examples

$$\{ R(); S(i, j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2 \}$$

$$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$$

General form

- Sets

$$\{ S_1(\mathbf{i}) : f_1(\mathbf{i}); S_2(\mathbf{i}) : f_2(\mathbf{i}); \dots \},$$

with f_k Presburger formulas

⇒ set of elements of the form $S_1(\mathbf{i})$, one for each \mathbf{i} satisfying $f_1(\mathbf{i}), \dots$

- Binary relations

$$\{ S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j}) : f_1(\mathbf{i}, \mathbf{j}); S_2(\mathbf{i}) \rightarrow T_2(\mathbf{j}) : f_2(\mathbf{i}, \mathbf{j}); \dots \}$$

⇒ set of pairs of elements of the form $S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j})$

Note: despite “ \rightarrow ”, not necessarily (single valued) functions

First Order Formulas

- **Language** $\mathcal{L} = \{ f_1/r_1, f_2/r_2, \dots, P_1/s_1, P_2/s_2, \dots \}$
 f_i function symbol with arity $r_i \geq 0$

P_i predicate symbol with arity $s_i \geq 0$

- **Terms** (inductive definition)
 - ▶ v is a term if v is a variable
 - ▶ $f_i(t_1, \dots, t_{r_i})$ is a term if t_j are terms
(in particular, if $r_i = 0$, then $f_i()$ is a term)
- **Formulas** (inductive definition):

- | | | |
|------------------------------|----------------------------------|-----------------------------------|
| ▶ true | ▶ $F_1 \wedge F_2$ (conjunction) | quantification: |
| ▶ $P_i(t_1, \dots, t_{s_i})$ | ▶ $F_1 \vee F_2$ (disjunction) | ▶ $\exists v : F_1$ (existential) |
| ▶ $t_1 = t_2$ | ▶ $\neg F_1$ (negation) | ▶ $\forall v : F_1$ (universal) |

if P_i/s_i is a predicate, t_j are terms, v is a variable, and F_k are formulas

First Order Formulas — Presburger formulas

- Language $\mathcal{L} = \{ f_1/r_1, f_2/r_2, \dots, P_1/s_1, P_2/s_2, \dots \}$

f_i function symbol with arity $r_i \geq 0$

- ★ $+/2, -/2$
- ★ a constant $d/0$ for each integer d
- ★ a symbol $\lfloor \cdot/d \rfloor /1$ for each positive integer d
- ★ a set of symbolic constants $c_i/0$

P_i predicate symbol with arity $s_i \geq 0$

- ★ $\leq/2$

- Terms (inductive definition)

- ▶ v is a term if v is a variable
- ▶ $f_i(t_1, \dots, t_{r_i})$ is a term if t_j are terms
(in particular, if $r_i = 0$, then $f_i()$ is a term)

- Formulas (inductive definition):

- | | | |
|------------------------------|----------------------------------|-----------------------------------|
| ▶ true | ▶ $F_1 \wedge F_2$ (conjunction) | quantification: |
| ▶ $P_i(t_1, \dots, t_{s_i})$ | ▶ $F_1 \vee F_2$ (disjunction) | ▶ $\exists v : F_1$ (existential) |
| ▶ $t_1 = t_2$ | ▶ $\neg F_1$ (negation) | ▶ $\forall v : F_1$ (universal) |

if P_i/s_i is a predicate, t_j are terms, v is a variable, and F_k are formulas

Truth Value

- Domain of discourse (“universe”)
 - ▶ \mathbb{Z} (set of integers)

Truth Value

- Domain of discourse (“universe”)
 - ▶ \mathbb{Z} (set of integers)
- Interpretation function/predicate symbols → functions/predicates
 - ▶ $+/2$ and $-/2$ map to addition and subtraction on integers
 - ▶ constants $d/0$ map to corresponding value
 - ▶ $\lfloor \cdot/d \rfloor /1$ maps to integer division by d
 - ▶ symbolic constants $c_i/0$ are “uninterpreted”
 - ⇒ consider all possible interpretations as integers
 - ▶ \leq maps to “less than or equal” relation

Truth Value

- Domain of discourse (“universe”)
 - ▶ \mathbb{Z} (set of integers)
- Interpretation function/predicate symbols → functions/predicates
 - ▶ $+/2$ and $-/2$ map to addition and subtraction on integers
 - ▶ constants $d/0$ map to corresponding value
 - ▶ $\lfloor \cdot/d \rfloor /1$ maps to integer division by d
 - ▶ symbolic constants $c_i/0$ are “uninterpreted”
 - ⇒ consider all possible interpretations as integers
 - ▶ \leq maps to “less than or equal” relation
- Truth value
 - ▶ true is true
 - ▶ $P_i(t_1, \dots, t_{s_i})$ is true if interpretation is true
 - ▶ $t_1 = t_2$ is true if interpretations are equal
 - ▶ $F_1 \wedge F_2$ is true if both F_1 and F_2 are true
 - ▶ $F_1 \vee F_2$ is true if at least one of F_1 or F_2 is true
 - ▶ $\neg F_1$ is true if F_1 is not true
 - ▶ $\exists v : F(v)$ is true if $F(d)$ is true for some element d in the universe (\mathbb{Z})
 - ▶ $\forall v : F(v)$ is true if $F(d)$ is true for every element d in the universe (\mathbb{Z})

Truth Value

- Domain of discourse (“universe”)
 - ▶ \mathbb{Z} (set of integers)
- Interpretation function/predicate symbols → functions/predicates
 - ▶ $+/2$ and $-/2$ map to addition and subtraction on integers
 - ▶ constants $d/0$ map to corresponding value
 - ▶ $\lfloor \cdot/d \rfloor /1$ maps to integer division by d
 - ▶ symbolic constants $c_i/0$ are “uninterpreted”
 - ⇒ consider all possible interpretations as integers
 - ▶ \leq maps to “less than or equal” relation
- Truth value
 - ▶ true is true
 - ▶ $P_i(t_1, \dots, t_{s_i})$ is true if interpretation is true
 - ▶ $t_1 = t_2$ is true if interpretations are equal
 - ▶ $F_1 \wedge F_2$ is true if both F_1 and F_2 are true
 - ▶ $F_1 \vee F_2$ is true if at least one of F_1 or F_2 is true
 - ▶ $\neg F_1$ is true if F_1 is not true
 - ▶ $\exists v : F(v)$ is true if $F(d)$ is true for **some** element d in the universe (\mathbb{Z})
 - ▶ $\forall v : F(v)$ is true if $F(d)$ is true for **every** element d in the universe (\mathbb{Z})

$F(d)$: result of replacing every free occurrence (not bound by quantifier) of v in $F(v)$ by d

Presburger Sets and Relations

General form

- Sets

$$\{ S_1(\mathbf{i}) : f_1(\mathbf{i}); S_2(\mathbf{i}) : f_2(\mathbf{i}); \dots \},$$

where $f_k(\mathbf{i})$ are Presburger formulas with \mathbf{i} as only free variables

⇒ set of elements of the form $S_1(\mathbf{i})$, one for each \mathbf{i} such that $f_1(\mathbf{i})$ is true,

...

Presburger Sets and Relations

General form

- Sets

$$\{ S_1(\mathbf{i}) : f_1(\mathbf{i}); S_2(\mathbf{i}) : f_2(\mathbf{i}); \dots \},$$

where $f_k(\mathbf{i})$ are Presburger formulas with \mathbf{i} as only free variables

⇒ set of elements of the form $S_1(\mathbf{i})$, one for each \mathbf{i} such that $f_1(\mathbf{i})$ is true,

...

Note: may depend on interpretation of symbolic constants

$$\{ S(i) : 0 \leq i \leq n() \}$$

is equal to

$$\begin{cases} \emptyset & \text{if } n < 0 \\ \{ S(0) \} & \text{if } n = 0 \\ \{ S(0); S(1) \} & \text{if } n = 1 \\ \{ S(0); S(1); S(2) \} & \text{if } n = 2 \\ \dots & \end{cases}$$

Syntactic Sugar

- `false` is equal to $\neg\text{true}$

Syntactic Sugar

- false is equal to $\neg\text{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$

Syntactic Sugar

- false is equal to $\neg\text{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \text{true} \}$

Syntactic Sugar

- false is equal to $\neg \text{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \text{true} \}$
- $\{ S(i_1, \dots, i_{n-1}, g(i_1, \dots, i_{n-1}), i_{n+1}, \dots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \dots, i_{n-1}, i_n, i_{n+1}, \dots) : f(\mathbf{i}) \wedge i_n = g(i_1, \dots, i_{n-1}) \}$
Example: $\{ S(i) \rightarrow S(i + 1) \}$ is equal to $\{ S(i) \rightarrow S(j) : j = i + 1 \}$

Syntactic Sugar

- false is equal to $\neg \text{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \text{true} \}$
- $\{ S(i_1, \dots, i_{n-1}, g(i_1, \dots, i_{n-1}), i_{n+1}, \dots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \dots, i_{n-1}, i_n, i_{n+1}, \dots) : f(\mathbf{i}) \wedge i_n = g(i_1, \dots, i_{n-1}) \}$
Example: $\{ S(i) \rightarrow S(i + 1) \}$ is equal to $\{ S(i) \rightarrow S(j) : j = i + 1 \}$
- n is equal to $n()$

Syntactic Sugar

- false is equal to $\neg \text{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \text{true} \}$
- $\{ S(i_1, \dots, i_{n-1}, g(i_1, \dots, i_{n-1}), i_{n+1}, \dots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \dots, i_{n-1}, i_n, i_{n+1}, \dots) : f(\mathbf{i}) \wedge i_n = g(i_1, \dots, i_{n-1}) \}$
Example: $\{ S(i) \rightarrow S(i + 1) \}$ is equal to $\{ S(i) \rightarrow S(j) : j = i + 1 \}$
- n is equal to $n()$
- $a < b$ is equal to $a \leq b - 1$

Syntactic Sugar

- false is equal to $\neg \text{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \text{true} \}$
- $\{ S(i_1, \dots, i_{n-1}, g(i_1, \dots, i_{n-1}), i_{n+1}, \dots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \dots, i_{n-1}, i_n, i_{n+1}, \dots) : f(\mathbf{i}) \wedge i_n = g(i_1, \dots, i_{n-1}) \}$
Example: $\{ S(i) \rightarrow S(i + 1) \}$ is equal to $\{ S(i) \rightarrow S(j) : j = i + 1 \}$
- n is equal to $n()$
- $a < b$ is equal to $a \leq b - 1$
- $a \geq b$ is equal to $b \leq a$

Syntactic Sugar

- false is equal to $\neg \text{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \text{true} \}$
- $\{ S(i_1, \dots, i_{n-1}, g(i_1, \dots, i_{n-1}), i_{n+1}, \dots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \dots, i_{n-1}, i_n, i_{n+1}, \dots) : f(\mathbf{i}) \wedge i_n = g(i_1, \dots, i_{n-1}) \}$
Example: $\{ S(i) \rightarrow S(i + 1) \}$ is equal to $\{ S(i) \rightarrow S(j) : j = i + 1 \}$
- n is equal to $n()$
- $a < b$ is equal to $a \leq b - 1$
- $a \geq b$ is equal to $b \leq a$
- $a > b$ is equal to $a \geq b + 1$

Syntactic Sugar

- false is equal to $\neg \text{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \text{true} \}$
- $\{ S(i_1, \dots, i_{n-1}, g(i_1, \dots, i_{n-1}), i_{n+1}, \dots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \dots, i_{n-1}, i_n, i_{n+1}, \dots) : f(\mathbf{i}) \wedge i_n = g(i_1, \dots, i_{n-1}) \}$
Example: $\{ S(i) \rightarrow S(i + 1) \}$ is equal to $\{ S(i) \rightarrow S(j) : j = i + 1 \}$
- n is equal to $n()$
- $a < b$ is equal to $a \leq b - 1$
- $a \geq b$ is equal to $b \leq a$
- $a > b$ is equal to $a \geq b + 1$
- $a, b \oplus c$ is equal to $a \oplus c \wedge b \oplus c$ with $\oplus \in \{\leq, <, \geq, >, =\}$
Example: $\{ S(i, j) : i, j \geq 0 \}$ is equal to $\{ S(i, j) : i \geq 0 \wedge j \geq 0 \}$

Syntactic Sugar

- false is equal to $\neg \text{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \text{true} \}$
- $\{ S(i_1, \dots, i_{n-1}, g(i_1, \dots, i_{n-1}), i_{n+1}, \dots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \dots, i_{n-1}, i_n, i_{n+1}, \dots) : f(\mathbf{i}) \wedge i_n = g(i_1, \dots, i_{n-1}) \}$
Example: $\{ S(i) \rightarrow S(i + 1) \}$ is equal to $\{ S(i) \rightarrow S(j) : j = i + 1 \}$
- n is equal to $n()$
- $a < b$ is equal to $a \leq b - 1$
- $a \geq b$ is equal to $b \leq a$
- $a > b$ is equal to $a \geq b + 1$
- $a, b \oplus c$ is equal to $a \oplus c \wedge b \oplus c$ with $\oplus \in \{\leq, <, \geq, >, =\}$
Example: $\{ S(i, j) : i, j \geq 0 \}$ is equal to $\{ S(i, j) : i \geq 0 \wedge j \geq 0 \}$
- $a \oplus_1 b \oplus_2 c$ is equal to $a \oplus_1 b \wedge b \oplus_2 c$ with
 $\{\oplus_1, \oplus_2\} \subset \{\leq, <, \geq, >, =\}$
Example: $\{ S(i) : 0 \leq i \leq 10 \}$ is equal to $\{ S(i) : 0 \leq i \wedge i \leq 10 \}$

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$
- $n \cdot e$ is equal to $\underbrace{e + e + \cdots + e}_{n \text{ times}}$ (with n a non-negative integer constant)

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$
- $n \cdot e$ is equal to $\underbrace{e + e + \cdots + e}_{n \text{ times}}$ (with n a non-negative integer constant)
- $a_1, \dots, a_n \prec b_1, \dots, b_n$ is equal to $\bigvee_{i=1}^n \left(\left(\bigwedge_{j=1}^{i-1} a_j = b_j \right) \wedge a_i < b_i \right)$
Example: $\{ S(i_1, i_2) \rightarrow S(j_1, j_2) : i_1, i_2 \prec j_1, j_2 \}$ is equal to
 $\{ S(i_1, i_2) \rightarrow S(j_1, j_2) : i_1 < j_1 \vee (i_1 = j_1 \wedge i_2 < j_2) \}$

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$
- $n \cdot e$ is equal to $\underbrace{e + e + \cdots + e}_{n \text{ times}}$ (with n a non-negative integer constant)
- $a_1, \dots, a_n \prec b_1, \dots, b_n$ is equal to $\bigvee_{i=1}^n \left(\left(\bigwedge_{j=1}^{i-1} a_j = b_j \right) \wedge a_i < b_i \right)$
Example: $\{ S(i_1, i_2) \rightarrow S(j_1, j_2) : i_1, i_2 \prec j_1, j_2 \}$ is equal to
 $\{ S(i_1, i_2) \rightarrow S(j_1, j_2) : i_1 < j_1 \vee (i_1 = j_1 \wedge i_2 < j_2) \}$
- $a_1, \dots, a_n \preccurlyeq b_1, \dots, b_n$ is equal to
 $a_1, \dots, a_n \prec b_1, \dots, b_n \vee a_1, \dots, a_n = b_1, \dots, b_n$

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$
- $n \cdot e$ is equal to $\underbrace{e + e + \cdots + e}_{n \text{ times}}$ (with n a non-negative integer constant)
- $a_1, \dots, a_n \prec b_1, \dots, b_n$ is equal to $\bigvee_{i=1}^n \left(\left(\bigwedge_{j=1}^{i-1} a_j = b_j \right) \wedge a_i < b_i \right)$
Example: $\{ S(i_1, i_2) \rightarrow S(j_1, j_2) : i_1, i_2 \prec j_1, j_2 \}$ is equal to
 $\{ S(i_1, i_2) \rightarrow S(j_1, j_2) : i_1 < j_1 \vee (i_1 = j_1 \wedge i_2 < j_2) \}$
- $a_1, \dots, a_n \preccurlyeq b_1, \dots, b_n$ is equal to
 $a_1, \dots, a_n \prec b_1, \dots, b_n \vee a_1, \dots, a_n = b_1, \dots, b_n$
- $a_1, \dots, a_n \succ b_1, \dots, b_n$ is equal to $b_1, \dots, b_n \prec a_1, \dots, a_n$

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$
- $n \cdot e$ is equal to $\underbrace{e + e + \cdots + e}_{n \text{ times}}$ (with n a non-negative integer constant)
- $a_1, \dots, a_n \prec b_1, \dots, b_n$ is equal to $\bigvee_{i=1}^n \left(\left(\bigwedge_{j=1}^{i-1} a_j = b_j \right) \wedge a_i < b_i \right)$
Example: $\{ S(i_1, i_2) \rightarrow S(j_1, j_2) : i_1, i_2 \prec j_1, j_2 \}$ is equal to
 $\{ S(i_1, i_2) \rightarrow S(j_1, j_2) : i_1 < j_1 \vee (i_1 = j_1 \wedge i_2 < j_2) \}$
- $a_1, \dots, a_n \preccurlyeq b_1, \dots, b_n$ is equal to
 $a_1, \dots, a_n \prec b_1, \dots, b_n \vee a_1, \dots, a_n = b_1, \dots, b_n$
- $a_1, \dots, a_n \succ b_1, \dots, b_n$ is equal to $b_1, \dots, b_n \prec a_1, \dots, a_n$
- $a_1, \dots, a_n \succcurlyeq b_1, \dots, b_n$ is equal to $b_1, \dots, b_n \preccurlyeq a_1, \dots, a_n$

Examples

- $\{ \text{S}(i,j) \rightarrow (1, i, j) : 0 \leq i, j < 2; \text{T}(k) \rightarrow (2, k, 0) : 0 \leq k < 2 \}$
is equal to

$$\{ \text{S}(0,0) \rightarrow (1, 0, 0); \text{S}(0,1) \rightarrow (1, 0, 1); \text{S}(1,0) \rightarrow (1, 1, 0); \\ \text{S}(1,1) \rightarrow (1, 1, 1); \text{T}(0) \rightarrow (2, 0, 0); \text{T}(1) \rightarrow (2, 1, 0) \}$$

Examples

- $\{ \text{S}(i,j) \rightarrow (1, i, j) : 0 \leq i, j < 2; \text{T}(k) \rightarrow (2, k, 0) : 0 \leq k < 2 \}$
is equal to

$$\{ \text{S}(0,0) \rightarrow (1, 0, 0); \text{S}(0,1) \rightarrow (1, 0, 1); \text{S}(1,0) \rightarrow (1, 1, 0); \\ \text{S}(1,1) \rightarrow (1, 1, 1); \text{T}(0) \rightarrow (2, 0, 0); \text{T}(1) \rightarrow (2, 1, 0) \}$$

- $\{ (i) : 0 \leq i \leq 10 \wedge \exists \alpha : i = 2\alpha \}$ is equal to

$$\{ (0); (2); (4); (6); (8); (10) \}$$

Examples

- $\{ \text{S}(i,j) \rightarrow (1, i, j) : 0 \leq i, j < 2; \text{T}(k) \rightarrow (2, k, 0) : 0 \leq k < 2 \}$
is equal to

$$\{ \text{S}(0,0) \rightarrow (1, 0, 0); \text{S}(0,1) \rightarrow (1, 0, 1); \text{S}(1,0) \rightarrow (1, 1, 0); \\ \text{S}(1,1) \rightarrow (1, 1, 1); \text{T}(0) \rightarrow (2, 0, 0); \text{T}(1) \rightarrow (2, 1, 0) \}$$

- $\{ (i) : 0 \leq i \leq 10 \wedge \exists \alpha : i = 2\alpha \}$ is equal to

$$\{ (0); (2); (4); (6); (8); (10) \}$$

- $\{ (i) : \forall i : 0 \leq i \leq 10 \}$ is equal to

Examples

- $\{ \text{S}(i,j) \rightarrow (1, i, j) : 0 \leq i, j < 2; \text{T}(k) \rightarrow (2, k, 0) : 0 \leq k < 2 \}$
is equal to

$$\{ \text{S}(0,0) \rightarrow (1, 0, 0); \text{S}(0,1) \rightarrow (1, 0, 1); \text{S}(1,0) \rightarrow (1, 1, 0); \\ \text{S}(1,1) \rightarrow (1, 1, 1); \text{T}(0) \rightarrow (2, 0, 0); \text{T}(1) \rightarrow (2, 1, 0) \}$$

- $\{ (i) : 0 \leq i \leq 10 \wedge \exists \alpha : i = 2\alpha \}$ is equal to

$$\{ (0); (2); (4); (6); (8); (10) \}$$

- $\{ (i) : \forall i : 0 \leq i \leq 10 \}$ is equal to

$$\emptyset$$

Examples

- $\{ \text{S}(i,j) \rightarrow (1, i, j) : 0 \leq i, j < 2; \text{T}(k) \rightarrow (2, k, 0) : 0 \leq k < 2 \}$
is equal to

$$\{ \text{S}(0,0) \rightarrow (1, 0, 0); \text{S}(0,1) \rightarrow (1, 0, 1); \text{S}(1,0) \rightarrow (1, 1, 0); \\ \text{S}(1,1) \rightarrow (1, 1, 1); \text{T}(0) \rightarrow (2, 0, 0); \text{T}(1) \rightarrow (2, 1, 0) \}$$

- $\{ (i) : 0 \leq i \leq 10 \wedge \exists \alpha : i = 2\alpha \}$ is equal to

$$\{ (0); (2); (4); (6); (8); (10) \}$$

- $\{ (i) : \forall i : 0 \leq i \leq 10 \}$ is equal to

$$\emptyset$$

- $\{ (i) : 0 \leq i \leq 10 \wedge i = 2\alpha \}$ is equal to

Examples

- $\{ \text{S}(i,j) \rightarrow (1, i, j) : 0 \leq i, j < 2; \text{T}(k) \rightarrow (2, k, 0) : 0 \leq k < 2 \}$ is equal to

$$\{ \text{S}(0,0) \rightarrow (1, 0, 0); \text{S}(0,1) \rightarrow (1, 0, 1); \text{S}(1,0) \rightarrow (1, 1, 0); \\ \text{S}(1,1) \rightarrow (1, 1, 1); \text{T}(0) \rightarrow (2, 0, 0); \text{T}(1) \rightarrow (2, 1, 0) \}$$

- $\{ (i) : 0 \leq i \leq 10 \wedge \exists \alpha : i = 2\alpha \}$ is equal to

$$\{ (0); (2); (4); (6); (8); (10) \}$$

- $\{ (i) : \forall i : 0 \leq i \leq 10 \}$ is equal to

$$\emptyset$$

- $\{ (i) : 0 \leq i \leq 10 \wedge i = 2\alpha \}$ is equal to

$$\begin{cases} \{ (2\alpha) \} & \text{if } 0 \leq \alpha \leq 5 \\ \emptyset & \text{otherwise} \end{cases}$$

Outline

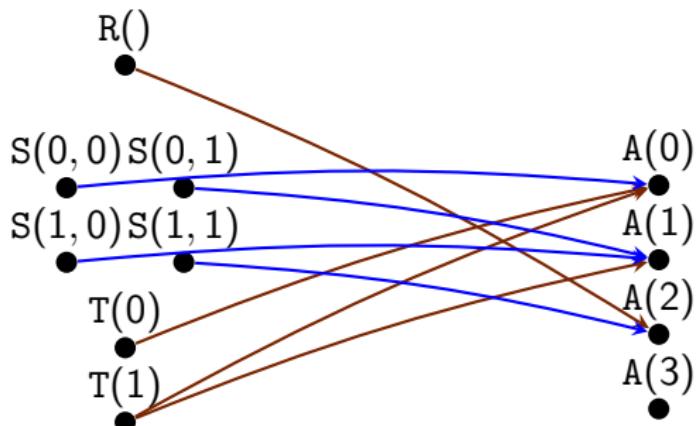
- 1 Introduction
- 2 Representation

3 Operations

- Basic Operations
- Application 1: Dependence Analysis
- Application 2: Array Dataflow Analysis
- Cardinality
- Application 3: Reuse Distance Computation
- Application 4: Maximal Number of Live Memory elements
- Nested Relations
- Application 5: Representing Dynamic Conditions
- Weighted Counting
- Application 6: Dynamic Memory Requirement Estimation
- Transitive Closures
- Application 7: Reachability Analysis

4 Availability

Union



access relation = read access relation \cup write access relation

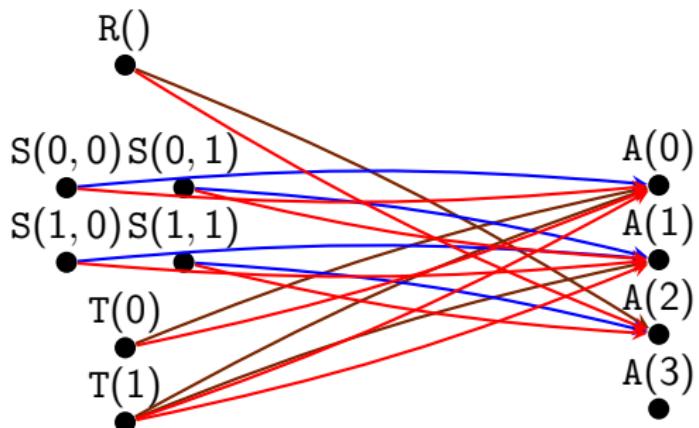
$$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \} \cup$$

$$\{ S(i,j) \rightarrow A(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

$$\Rightarrow \{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2;$$

$$S(i,j) \rightarrow A(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

Union



access relation = read access relation \cup write access relation

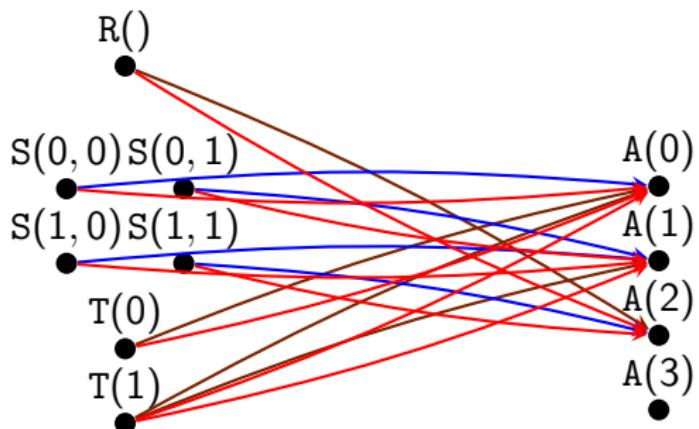
$$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \} \cup$$

$$\{ S(i,j) \rightarrow A(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

$$\Rightarrow \{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2;$$

$$S(i,j) \rightarrow A(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

Union



access relation = read access relation \cup write access relation

$$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \} \cup$$

$$\{ S(i,j) \rightarrow A(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

$$\Rightarrow \{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2;$$

$$S(i,j) \rightarrow A(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

$$\{ S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j}) : f_1(\mathbf{i}, \mathbf{j}); \dots \} \cup \{ S_2(\mathbf{i}) \rightarrow T_2(\mathbf{j}) : f_2(\mathbf{i}, \mathbf{j}); \dots \}$$

$$\Rightarrow \{ S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j}) : f_1(\mathbf{i}, \mathbf{j}); \dots; S_2(\mathbf{i}) \rightarrow T_2(\mathbf{j}) : f_2(\mathbf{i}, \mathbf{j}); \dots \}$$

Spaces

Recall general form

- Sets

$$\{ S_1(\mathbf{i}) : f_1(\mathbf{i}); S_2(\mathbf{i}) : f_2(\mathbf{i}); \dots \},$$

- Binary relations

$$\{ S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j}) : f_1(\mathbf{i}, \mathbf{j}); S_2(\mathbf{i}) \rightarrow T_2(\mathbf{j}) : f_2(\mathbf{i}, \mathbf{j}); \dots \}$$

The **identifier** (e.g., S_1 , S_2 , T_1 , T_2), together with the **dimension**, i.e., the number of elements in the subsequent tuple (e.g., \mathbf{i} , \mathbf{j}), will be called a **space**

When we say $S_2(\mathbf{i}) = T_1(\mathbf{j})$, we mean

- the **identifiers** S_2 and T_1 are the same
- the **dimensions** of \mathbf{i} and \mathbf{j} are the same

Examples: $S() \neq S(i)$, $S(a) = S(b)$, $S() \neq T()$

Space Decomposition

General form can be rewritten

$$\begin{aligned} & \{ S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j}) : f_1(\mathbf{i}, \mathbf{j}); S_2(\mathbf{i}) \rightarrow T_2(\mathbf{j}) : f_2(\mathbf{i}, \mathbf{j}); \dots \} \\ &= \bigcup_k \{ S_k(\mathbf{i}) \rightarrow T_k(\mathbf{j}) : f_k(\mathbf{i}, \mathbf{j}) \} \end{aligned}$$

- some operations distribute with union
- other operations are defined on a single (pair of) space(s)
 - ⇒ “space local” operations
 - ⇒ replace $\{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f_1(\mathbf{i}, \mathbf{j}); S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f_2(\mathbf{i}, \mathbf{j}) \}$
by $\{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f_1(\mathbf{i}, \mathbf{j}) \vee f_2(\mathbf{i}, \mathbf{j}) \}$

In both cases, we define

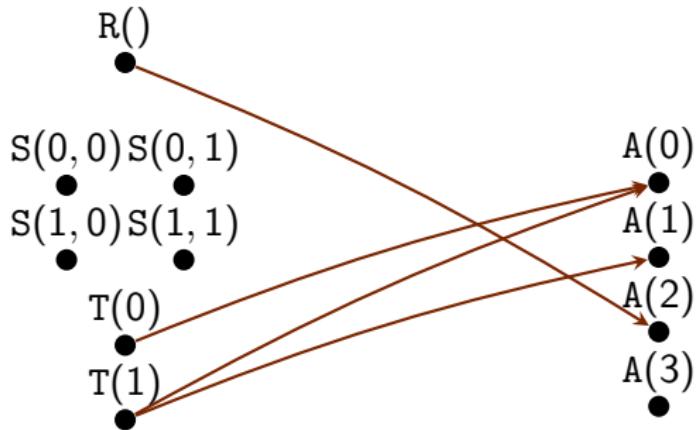
- unary operator \oplus

$$\oplus \bigcup_i R_i := \bigcup_i \oplus R_i$$

- binary operator \oplus

$$\left(\bigcup_i R_i \right) \oplus \left(\bigcup_j S_j \right) := \bigcup_i \bigcup_j (R_i \oplus S_j)$$

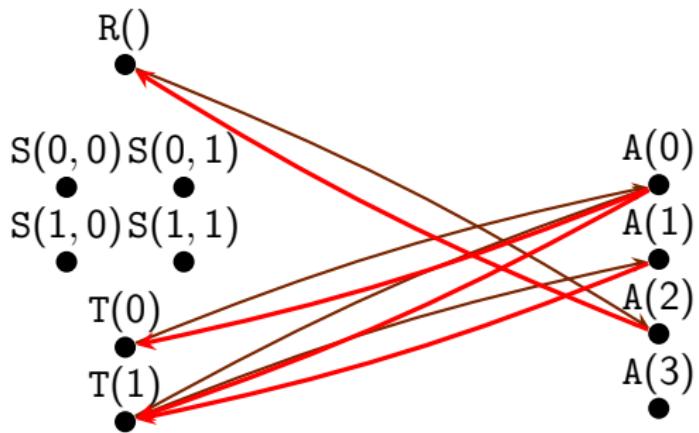
Inverse Relation



R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

Inverse Relation



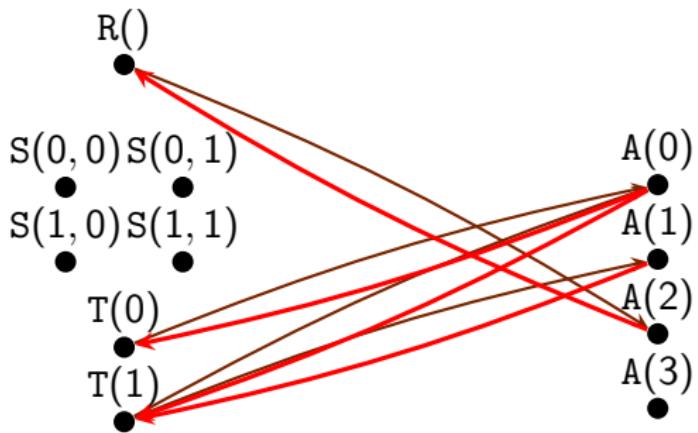
R : array elements read by statement instance

$$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$$

R^{-1} : statement instances reading array element

$$\{ A(2) \rightarrow R(); A(0) \rightarrow T(k) : 0 \leq k < 2; A(k) \rightarrow T(k) : 0 \leq k < 2 \}$$

Inverse Relation



R : array elements read by statement instance

$$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$$

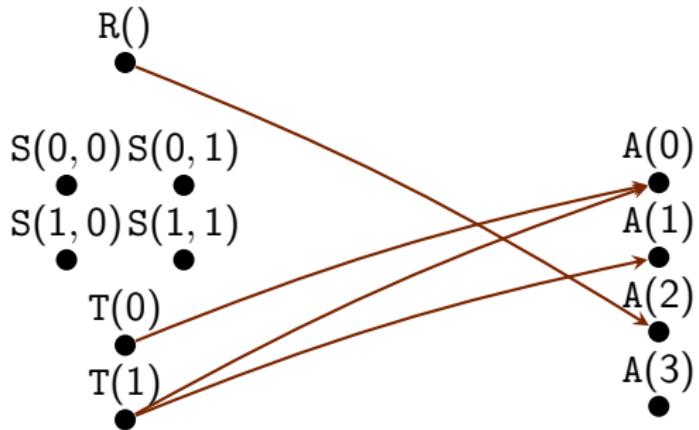
R^{-1} : statement instances reading array element

$$\{ A(2) \rightarrow R(); A(0) \rightarrow T(k) : 0 \leq k < 2; A(k) \rightarrow T(k) : 0 \leq k < 2 \}$$

$$R = \{ S(i) \rightarrow T(j) : f(i,j) \}$$

$$R^{-1} = \{ T(j) \rightarrow S(i) : f(i,j) \}$$

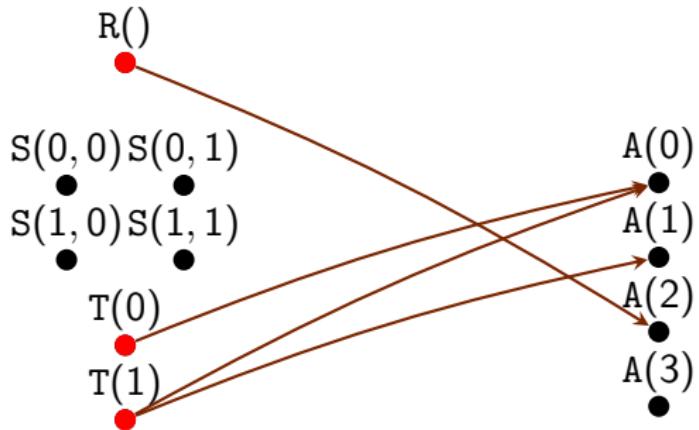
Domain of a Relation



R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

Domain of a Relation



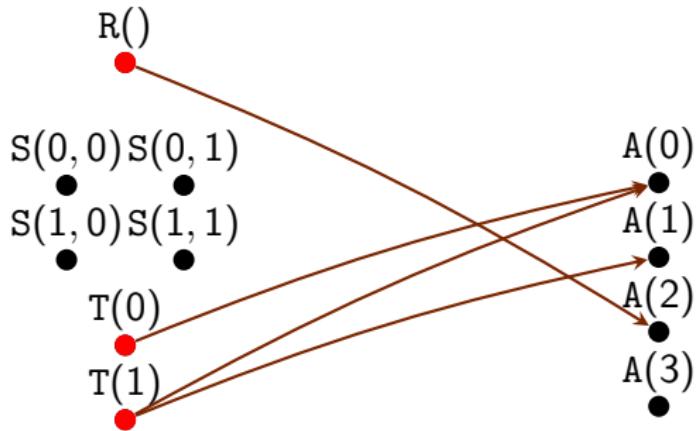
R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

dom R : statement instances reading array elements

$\{ R(); T(k) : 0 \leq k < 2 \}$

Domain of a Relation



R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

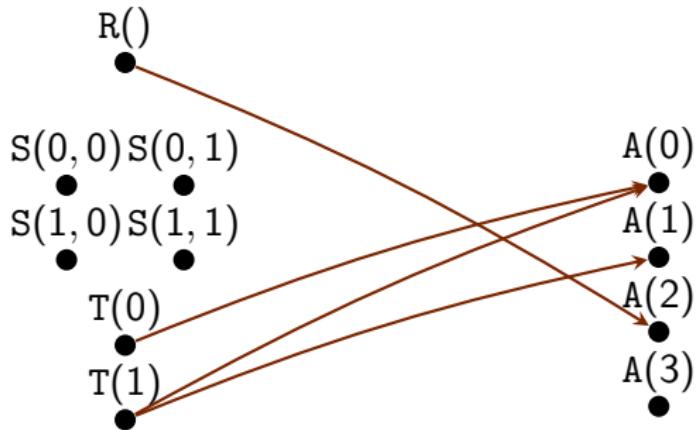
dom R : statement instances reading array elements

$\{ R(); T(k) : 0 \leq k < 2 \}$

$$R = \{ S(i) \rightarrow T(j) : f(i,j) \}$$

dom R = { $S(i) : \exists j : f(i,j)$ }

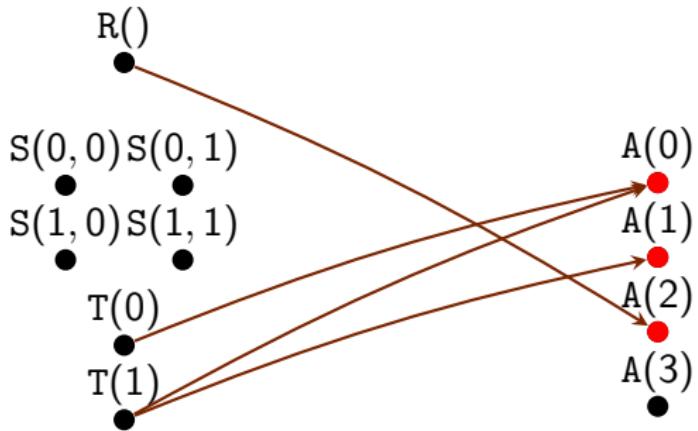
Range of a Relation



R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

Range of a Relation



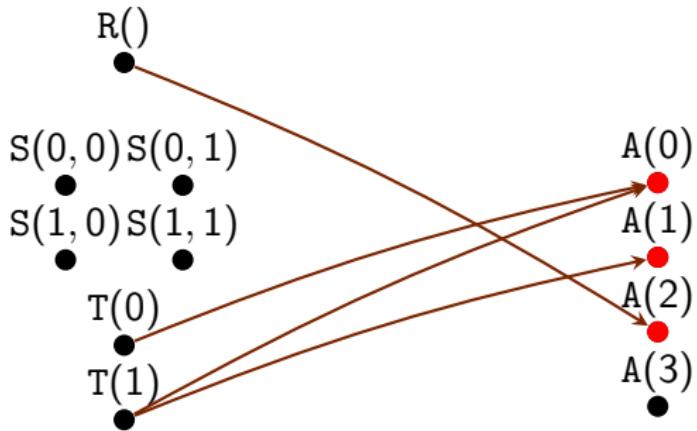
R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

ran R : array elements read by statement instances

$\{ A(k) : 0 \leq k \leq 2 \}$

Range of a Relation



R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

$\text{ran } R$: array elements read by statement instances

$\{ A(k) : 0 \leq k \leq 2 \}$

$$R = \{ S(i) \rightarrow T(j) : f(i,j) \}$$

$$\text{ran } R = \{ T(j) : \exists i : f(i,j) \}$$

Universal Relation

R()



S(0, 0) S(0, 1)



S(1, 0) S(1, 1)



T(0)



T(1)



A(0)



A(1)



A(2)



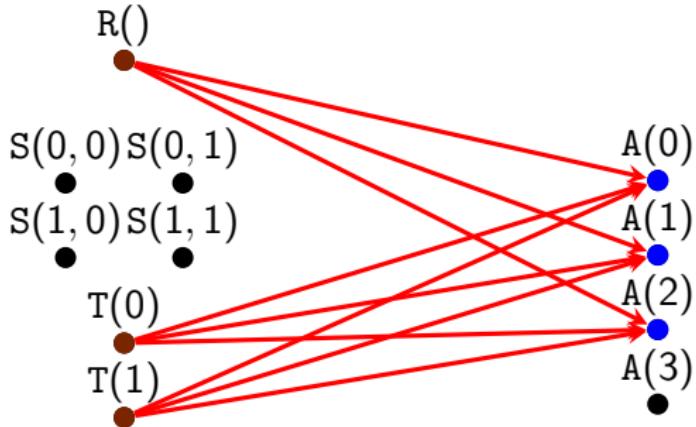
A(3)



A: statement instances reading array elements $\{ R(); T(k) : 0 \leq k < 2 \}$

B: array elements read by statement instances $\{ A(k) : 0 \leq k \leq 2 \}$

Universal Relation

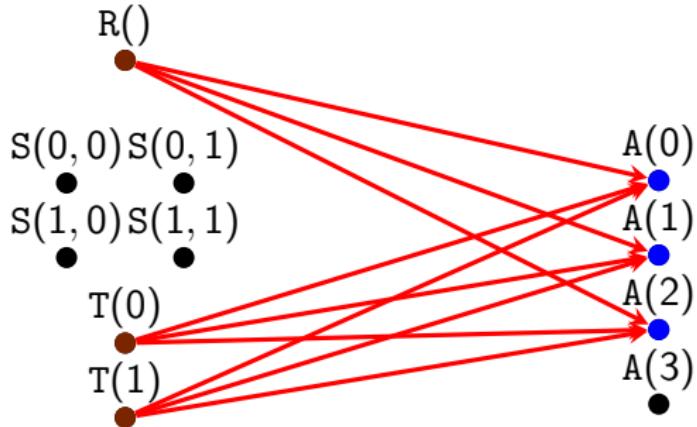


A : statement instances reading array elements $\{ R(); T(k) : 0 \leq k < 2 \}$

B : array elements read by statement instances $\{ A(k) : 0 \leq k \leq 2 \}$

$A \rightarrow B = \{ R() \rightarrow A(k) : 0 \leq k \leq 2; T(k) \rightarrow A(k') : 0 \leq k < 2 \wedge 0 \leq k' \leq 2 \}$

Universal Relation



A : statement instances reading array elements $\{ R(); T(k) : 0 \leq k < 2 \}$

B : array elements read by statement instances $\{ A(k) : 0 \leq k \leq 2 \}$

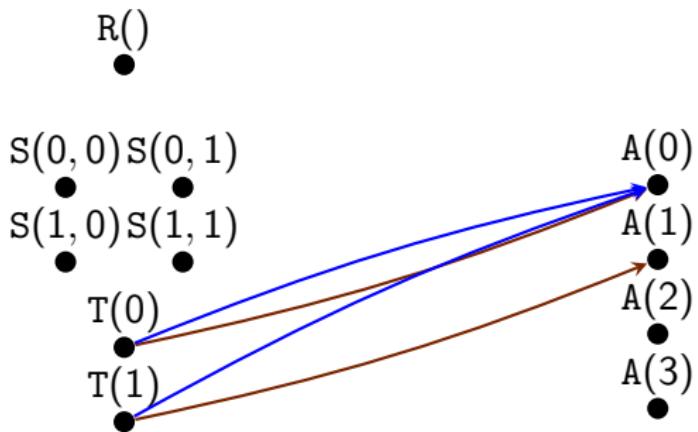
$A \rightarrow B = \{ R() \rightarrow A(k) : 0 \leq k \leq 2; T(k) \rightarrow A(k') : 0 \leq k < 2 \wedge 0 \leq k' \leq 2 \}$

$$A = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$$

$$B = \{ T(\mathbf{j}) : g(\mathbf{j}) \}$$

$$A \rightarrow B = \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f(\mathbf{i}) \wedge g(\mathbf{j}) \}$$

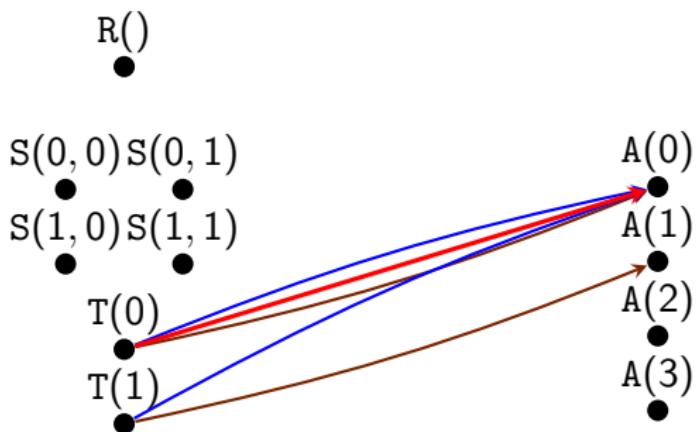
Intersection



A: read accesses through first reference $\{ T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

B: read accesses through second reference $\{ T(k) \rightarrow A(0) : 0 \leq k < 2 \}$

Intersection

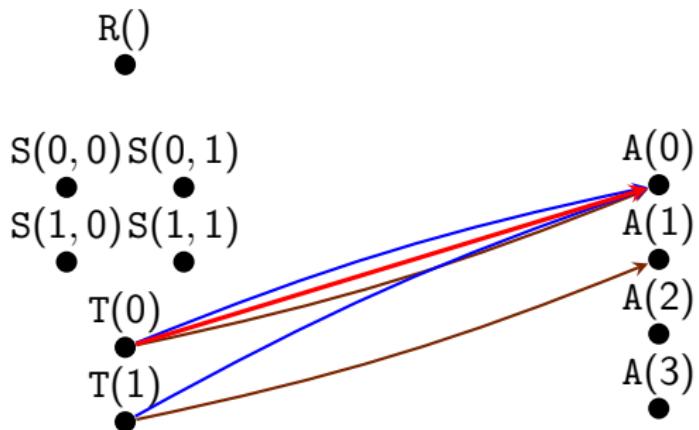


A : read accesses through first reference $\{ T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

B : read accesses through second reference $\{ T(k) \rightarrow A(0) : 0 \leq k < 2 \}$

$A \cap B$: coinciding accesses $\{ T(0) \rightarrow A(0) \}$

Intersection



A : read accesses through first reference $\{ T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

B : read accesses through second reference $\{ T(k) \rightarrow A(0) : 0 \leq k < 2 \}$

$A \cap B$: coinciding accesses $\{ T(0) \rightarrow A(0) \}$

$$A = \{ S_1(\mathbf{i}_1) \rightarrow T_1(\mathbf{j}_1) : f(\mathbf{i}_1, \mathbf{j}_1) \}$$

$$B = \{ S_2(\mathbf{i}_2) \rightarrow T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

$$A \cap B =$$

$$\begin{cases} \{ S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \wedge g(\mathbf{i}, \mathbf{j}) \} & \text{if } S_1(\mathbf{i}_1) = S_2(\mathbf{i}_2) \text{ and } T_1(\mathbf{j}_1) = T_2(\mathbf{j}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

Domain/Range Restriction

- Domain restriction

$$R \cap_{\text{dom}} S = R \cap (S \rightarrow (\text{ran } R))$$

$$A = \{ S_1(\mathbf{i}_1) : f(\mathbf{i}_1) \}$$

$$B = \{ S_2(\mathbf{i}_2) \rightarrow T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

$$B \cap_{\text{dom}} A = \begin{cases} \{ S_2(\mathbf{i}) \rightarrow T_2(\mathbf{j}) : f(\mathbf{i}) \wedge g(\mathbf{i}, \mathbf{j}) \} & \text{if } S_1(\mathbf{i}_1) = S_2(\mathbf{i}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

Domain/Range Restriction

- Domain restriction

$$R \cap_{\text{dom}} S = R \cap (S \rightarrow (\text{ran } R))$$

$$A = \{ S_1(\mathbf{i}_1) : f(\mathbf{i}_1) \}$$

$$B = \{ S_2(\mathbf{i}_2) \rightarrow T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

$$B \cap_{\text{dom}} A = \begin{cases} \{ S_2(\mathbf{i}) \rightarrow T_2(\mathbf{j}) : f(\mathbf{i}) \wedge g(\mathbf{i}, \mathbf{j}) \} & \text{if } S_1(\mathbf{i}_1) = S_2(\mathbf{i}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

- Range restriction

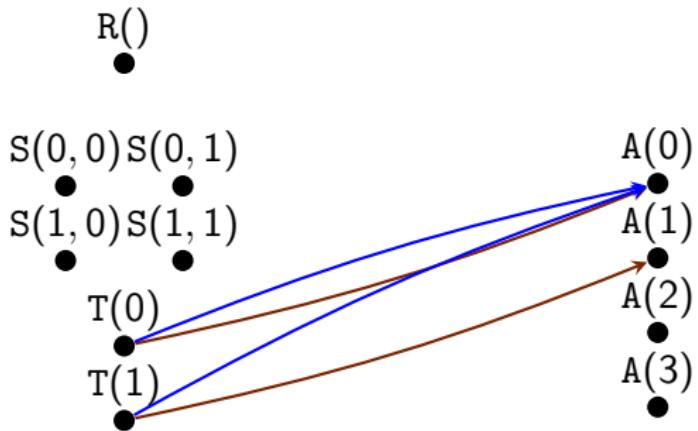
$$R \cap_{\text{ran}} S = R \cap ((\text{dom } R) \rightarrow S)$$

$$A = \{ S_1(\mathbf{i}_1) : f(\mathbf{i}_1) \}$$

$$B = \{ S_2(\mathbf{i}_2) \rightarrow T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

$$B \cap_{\text{ran}} A = \begin{cases} \{ S_2(\mathbf{i}) \rightarrow T_2(\mathbf{j}) : f(\mathbf{j}) \wedge g(\mathbf{i}, \mathbf{j}) \} & \text{if } S_1(\mathbf{i}_1) = T_2(\mathbf{j}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

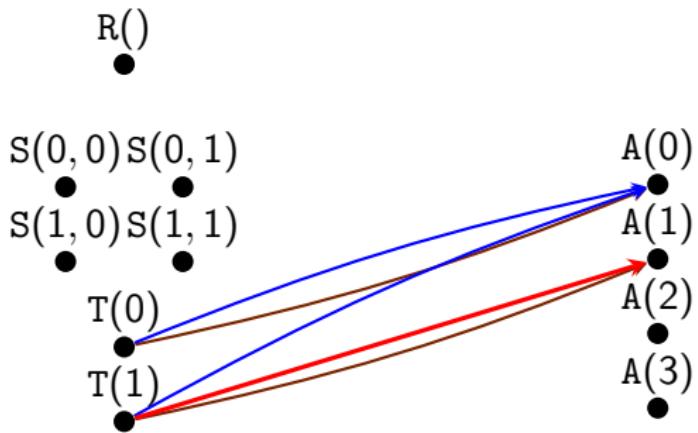
Difference



A: read accesses through first reference $\{ T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

B: read accesses through second reference $\{ T(k) \rightarrow A(0) : 0 \leq k < 2 \}$

Difference

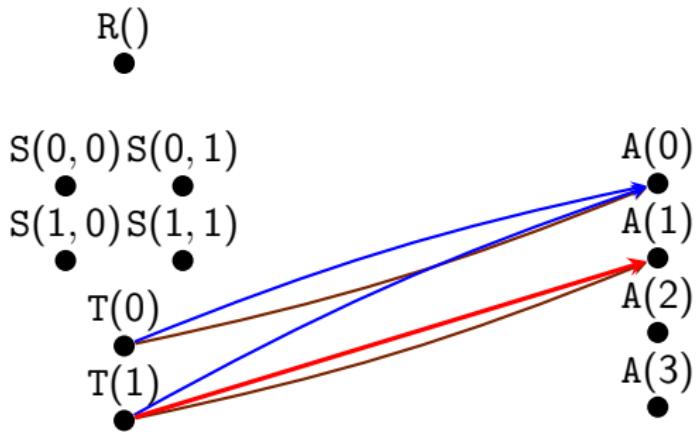


A : read accesses through first reference $\{ T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

B : read accesses through second reference $\{ T(k) \rightarrow A(0) : 0 \leq k < 2 \}$

$A \setminus B$: accesses through first, not coinciding with second $\{ T(1) \rightarrow A(1) \}$

Difference



A: read accesses through first reference $\{ T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

B: read accesses through second reference $\{ T(k) \rightarrow A(0) : 0 \leq k < 2 \}$

A \ B: accesses through first, not coinciding with second $\{ T(1) \rightarrow A(1) \}$

$$A = \{ S_1(\mathbf{i}_1) \rightarrow T_1(\mathbf{j}_1) : f(\mathbf{i}_1, \mathbf{j}_1) \}$$

$$B = \{ S_2(\mathbf{i}_2) \rightarrow T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

$$A \setminus B =$$

$$\begin{cases} \{ S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \wedge \neg g(\mathbf{i}, \mathbf{j}) \} & \text{if } S_1(\mathbf{i}_1) = S_2(\mathbf{i}_2) \text{ and } T_1(\mathbf{j}_1) = T_2(\mathbf{j}_2) \\ \{ S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \} & \text{otherwise} \end{cases}$$

Emptiness Check and Comparisons

- Emptiness check

Does a set or binary relation contain any elements?
(for any value of the symbolic constants)

Emptiness Check and Comparisons

- Emptiness check

Does a set or binary relation contain any elements?
(for any value of the symbolic constants)

Is

$$\{ (a, b, c, d) : 3d \geq -21 + 19a - 11b - 6c \wedge 3d \leq 21 + 17a - b - 6c \wedge 2b \leq -15 + a \wedge 3d \leq 2 + a + b \wedge 3d \geq a + b \}$$

empty?

Emptiness Check and Comparisons

- Emptiness check

Does a set or binary relation contain any elements?
(for any value of the symbolic constants)

Is

$$\{ (a, b, c, d) : 3d \geq -21 + 19a - 11b - 6c \wedge 3d \leq 21 + 17a - b - 6c \wedge 2b \leq -15 + a \wedge 3d \leq 2 + a + b \wedge 3d \geq a + b \}$$

empty?

⇒ No, contains $(13, -1, 38, 4)$ (and infinitely many other elements)

Emptiness Check and Comparisons

- Emptiness check

Does a set or binary relation contain any elements?
(for any value of the symbolic constants)

Is

$$\{ (a, b, c, d) : 3d \geq -21 + 19a - 11b - 6c \wedge 3d \leq 21 + 17a - b - 6c \wedge 2b \leq -15 + a \wedge 3d \leq 2 + a + b \wedge 3d \geq a + b \}$$

empty?

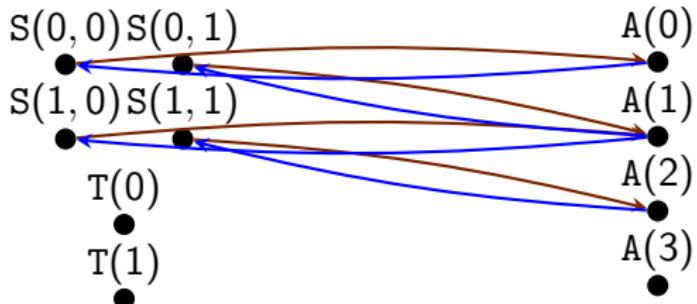
⇒ No, contains $(13, -1, 38, 4)$ (and infinitely many other elements)

- Comparisons

- ▶ $A \subseteq B$ is defined as $A \setminus B = \emptyset$
- ▶ $A \supseteq B$ is defined as $B \subseteq A$
- ▶ $A = B$ is defined as $A \subseteq B \wedge A \supseteq B$
- ▶ $A \subset B$ is defined as $A \subseteq B \wedge \neg(A = B)$
- ▶ $A \supset B$ is defined as $B \subset A$

Composition

R()



W : array elements written by statement instance

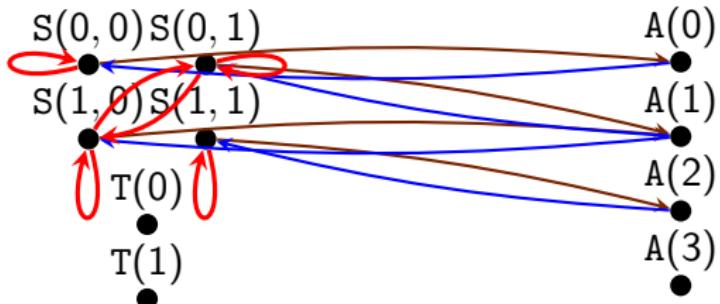
$$\{ S(i,j) \rightarrow A(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

W^{-1} : statement instances writing array element

$$\{ A(a) \rightarrow S(i,j) : a = i + j \wedge 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

Composition

$R()$



W : array elements written by statement instance

$$\{ S(i,j) \rightarrow A(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

W^{-1} : statement instances writing array element

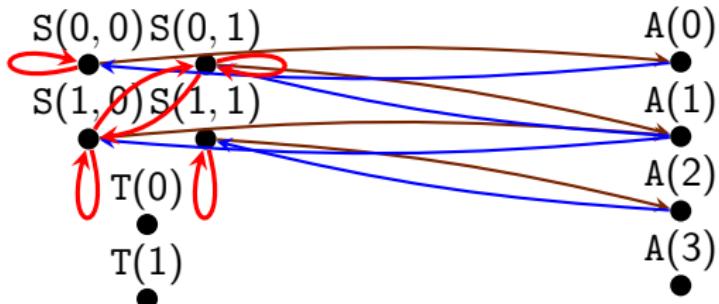
$$\{ A(a) \rightarrow S(i,j) : a = i + j \wedge 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

$W^{-1} \circ W$: pairs of statement instances that write same array element

$$\{ S(i,j) \rightarrow S(i',j') : 0 \leq i, i', j, j' < 2 \wedge i + j = i' + j' \}$$

Composition

$R()$



W : array elements written by statement instance

$$\{ S(i,j) \rightarrow A(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

W^{-1} : statement instances writing array element

$$\{ A(a) \rightarrow S(i,j) : a = i + j \wedge 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

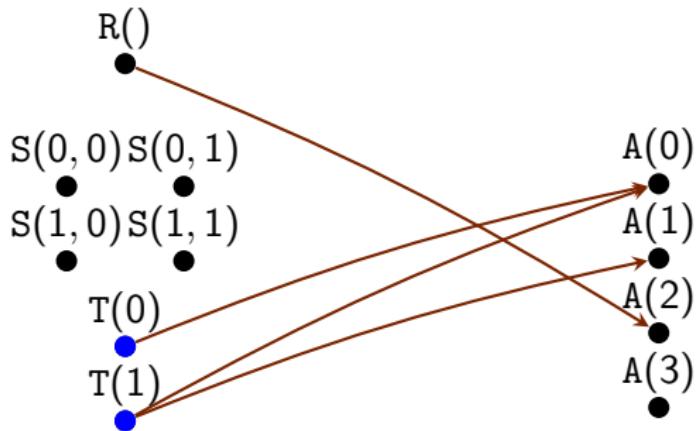
$W^{-1} \circ W$: pairs of statement instances that write same array element

$$\{ S(i,j) \rightarrow S(i',j') : 0 \leq i, i', j, j' < 2 \wedge i + j = i' + j' \}$$

$$A = \{ S_1(\mathbf{i}_1) \rightarrow T_1(\mathbf{j}_1) : f(\mathbf{i}_1, \mathbf{j}_1) \} \quad B = \{ S_2(\mathbf{i}_2) \rightarrow T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

$$B \circ A = \begin{cases} \{ S_1(\mathbf{i}) \rightarrow T_2(\mathbf{j}) : \exists \mathbf{k} : f(\mathbf{i}, \mathbf{k}) \wedge g(\mathbf{k}, \mathbf{j}) \} & \text{if } T_1(\mathbf{j}_1) = S_2(\mathbf{i}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

Application

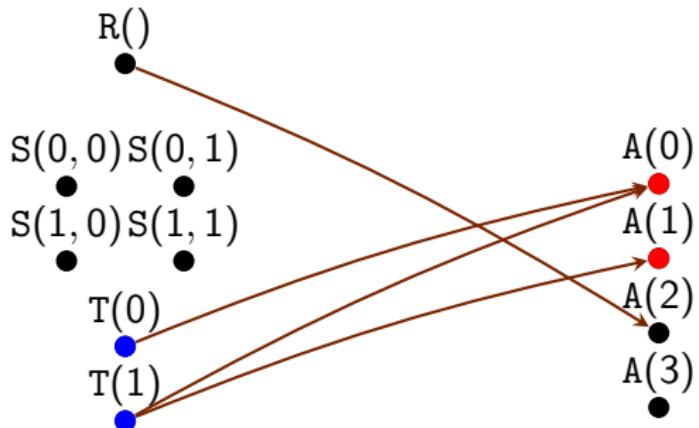


R : read access relation

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k \leq 2 \}$

S : instances of T statement $\{ T(k) : 0 \leq k \leq 2 \}$

Application



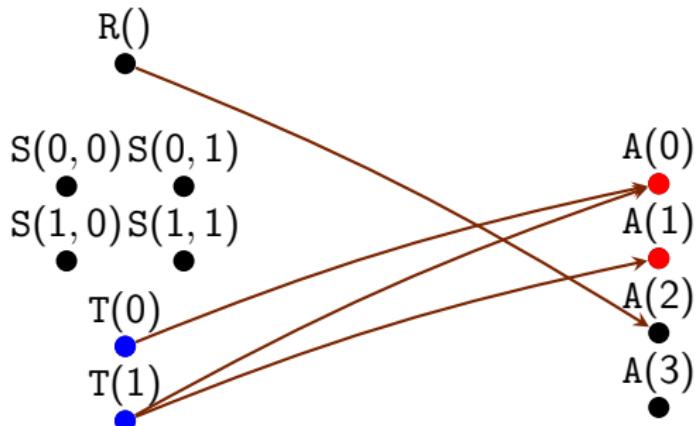
R : read access relation

$$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k \leq 2 \}$$

S : instances of T statement $\{ T(k) : 0 \leq k \leq 2 \}$

$$R(S) = \{ A(k) : 0 \leq k \leq 2 \}$$

Application



R : read access relation

$$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k \leq 2 \}$$

S : instances of T statement $\{ T(k) : 0 \leq k \leq 2 \}$

$$R(S) = \{ A(k) : 0 \leq k \leq 2 \}$$

$$A = \{ S_1(\mathbf{i}_1) : f(\mathbf{i}_1) \}$$

$$B = \{ S_2(\mathbf{i}_2) \rightarrow T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

$$B(A) = \begin{cases} \{ T_2(\mathbf{j}) : \exists \mathbf{i} : f(\mathbf{i}) \wedge g(\mathbf{i}, \mathbf{j}) \} & \text{if } S_1(\mathbf{i}_1) = S_2(\mathbf{i}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

Lexicographic Order — Sets

R()



S(0, 0) S(0, 1)



S(1, 0) S(1, 1)



T(0)



T(1)



A(0)



A(1)



A(2)



A(3)



I: iteration domain

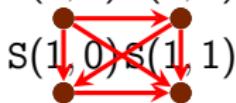
{ R(); S(i, j) : 0 ≤ i < 2 ∧ 0 ≤ j < 2; T(k) : 0 ≤ k < 2 }

Lexicographic Order — Sets

R()



S(0, 0) S(0, 1)



T(0)

T(1)

A(0)

A(1)

A(2)

A(3)

A(4)

I: iteration domain

$$\{ R(); S(i, j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2 \}$$

I \prec I: lexicographic order on pairs of statement instances

$$\{ S(i, j) \rightarrow S(i', j') : 0 \leq i, i', j, j' < 2 \wedge i, j \prec i', j'; T(0) \rightarrow T(1) \}$$

Lexicographic Order — Sets

$R()$



$S(0, 0) S(0, 1)$



$T(0)$

$T(1)$

$A(0)$

$A(1)$

$A(2)$

$A(3)$

I : iteration domain

$\{ R(); S(i, j) : 0 \leq i < 2 \wedge 0 \leq j < 2; T(k) : 0 \leq k < 2 \}$

$I \prec I$: lexicographic order on pairs of statement instances

$\{ S(i, j) \rightarrow S(i', j') : 0 \leq i, i', j, j' < 2 \wedge i, j \prec i', j'; T(0) \rightarrow T(1) \}$

$A = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$

$B = \{ T(\mathbf{j}) : g(\mathbf{j}) \}$

$A \prec B = \begin{cases} \{ S(\mathbf{i}) \rightarrow S(\mathbf{j}) : f(\mathbf{i}) \wedge g(\mathbf{j}) \wedge \mathbf{i} \prec \mathbf{j} \} & \text{if } S(\mathbf{i}) = T(\mathbf{j}) \\ \emptyset & \text{otherwise} \end{cases}$

Lexicographic Order — Relations

- Sets

$$A = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$$

$$B = \{ T(\mathbf{j}) : g(\mathbf{j}) \}$$

$$A \prec B = \begin{cases} \{ S(\mathbf{i}) \rightarrow S(\mathbf{j}) : f(\mathbf{i}) \wedge g(\mathbf{j}) \wedge \mathbf{i} \prec \mathbf{j} \} & \text{if } S(\mathbf{i}) = T(\mathbf{j}) \\ \emptyset & \text{otherwise} \end{cases}$$

Lexicographic Order — Relations

- Sets

$$A = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$$

$$B = \{ T(\mathbf{j}) : g(\mathbf{j}) \}$$

$$A \prec B = \begin{cases} \{ S(\mathbf{i}) \rightarrow S(\mathbf{j}) : f(\mathbf{i}) \wedge g(\mathbf{j}) \wedge \mathbf{i} \prec \mathbf{j} \} & \text{if } S(\mathbf{i}) = T(\mathbf{j}) \\ \emptyset & \text{otherwise} \end{cases}$$

- Relations

⇒ binary relation on domains reflecting lexicographic order of images

$$A = \{ S_1(\mathbf{i}_1) \rightarrow T_1(\mathbf{j}_1) : f(\mathbf{i}_1, \mathbf{j}_1) \}$$

$$B = \{ S_2(\mathbf{i}_2) \rightarrow T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

$$A \prec B = \begin{cases} \{ S_1(\mathbf{i}_1) \rightarrow S_2(\mathbf{i}_2) : \exists \mathbf{j}_1, \mathbf{j}_2 : f(\mathbf{i}_1, \mathbf{j}_1) \wedge g(\mathbf{i}_2, \mathbf{j}_2) \wedge \mathbf{j}_1 \prec \mathbf{j}_2 \} & \text{if } T_1(\mathbf{j}_1) = T_2(\mathbf{j}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

Application 1: Dependence Analysis

Recall:

Scheduling computes a new schedule: reorders statement instances

New schedule S' needs to respect dependences D

$$i \rightarrow j \in D \Rightarrow S'(i) \prec S'(j)$$

Application 1: Dependence Analysis

Recall:

Scheduling computes a new schedule: reorders statement instances

New schedule S' needs to respect dependences D

$$i \rightarrow j \in D \Rightarrow S'(i) \prec S'(j)$$

Iteration j depends on iteration i if

- i is executed before j (in original program)
- i and j (may) access the same memory location
- at least one of those two accesses is a write

Application 1: Dependence Analysis

Recall:

Scheduling computes a new schedule: reorders statement instances

New schedule S' needs to respect dependences D

$$i \rightarrow j \in D \Rightarrow S'(i) \prec S'(j)$$

Iteration j depends on iteration i if

- i is executed before j (in original program)
- i and j (may) access the same memory location
- at least one of those two accesses is a write

Computation:

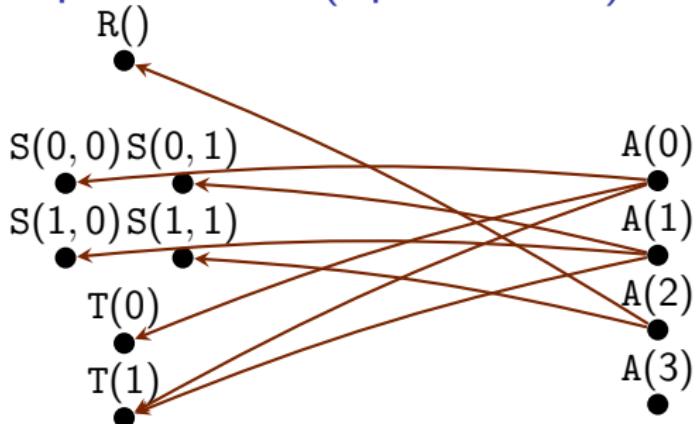
$$D = ((W^{-1} \circ R) \cup (W^{-1} \circ W) \cup (R^{-1} \circ W)) \cap (S \prec S)$$

W : write access relation

R : write access relation

S : original schedule

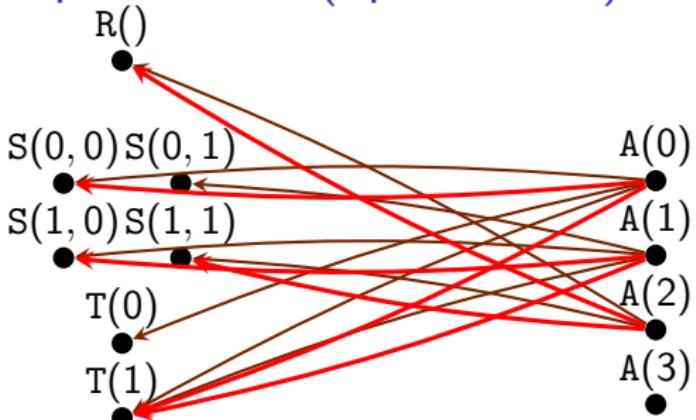
Lexicographic Optimization (Space Local)



$(R \cup W)^{-1}$: statement instances accessing array element

$\{ A(2) \rightarrow R(); A(a) \rightarrow S(i, j) : a = i + j \wedge 0 \leq i < 2 \wedge 0 \leq j < 2;$
 $A(0) \rightarrow T(k) : 0 \leq k < 2; A(k) \rightarrow T(k) : 0 \leq k < 2 \}$

Lexicographic Optimization (Space Local)



$(R \cup W)^{-1}$: statement instances accessing array element

$\{ A(2) \rightarrow R(); A(a) \rightarrow S(i,j) : a = i + j \wedge 0 \leq i < 2 \wedge 0 \leq j < 2;$

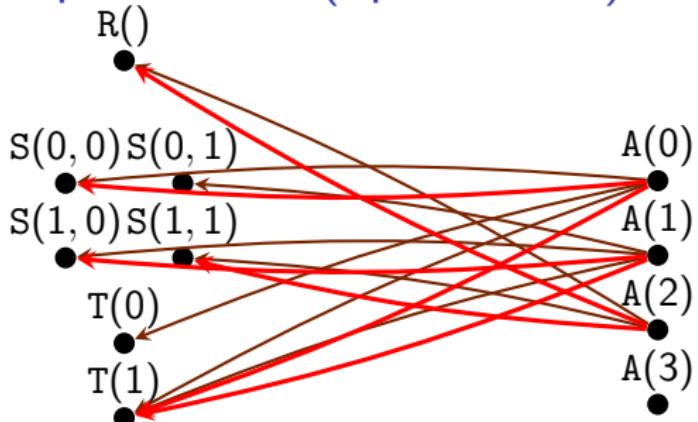
$A(0) \rightarrow T(k) : 0 \leq k < 2; A(k) \rightarrow T(k) : 0 \leq k < 2 \}$

$\text{lexmax}(R \cup W)^{-1}$: last instance of statement accessing array element

$\{ A(2) \rightarrow R(); A(a) \rightarrow S(a,0) : 0 \leq a < 2; A(2) \rightarrow S(1,1);$

$A(k) \rightarrow T(1) : 0 \leq k < 2 \}$

Lexicographic Optimization (Space Local)



$(R \cup W)^{-1}$: statement instances accessing array element

$\{ A(2) \rightarrow R(); A(a) \rightarrow S(i,j) : a = i + j \wedge 0 \leq i < 2 \wedge 0 \leq j < 2;$

$A(0) \rightarrow T(k) : 0 \leq k < 2; A(k) \rightarrow T(k) : 0 \leq k < 2 \}$

$\text{lexmax}(R \cup W)^{-1}$: last instance of statement accessing array element

$\{ A(2) \rightarrow R(); A(a) \rightarrow S(a,0) : 0 \leq a < 2; A(2) \rightarrow S(1,1);$

$A(k) \rightarrow T(1) : 0 \leq k < 2 \}$

$R = \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \}$

$\text{lexmax } R = \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \wedge \forall \mathbf{j}' : f(\mathbf{i}, \mathbf{j}') \Rightarrow \mathbf{j} \succcurlyeq \mathbf{j}' \}$

Application 2: Array Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);
```

Application 2: Array Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);
```

Application 2: Array Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);
```

Access relations:

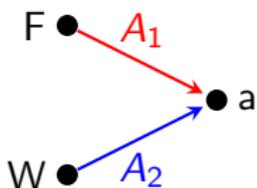
$$A_1 = \{ F(i, j) \rightarrow a(i + j) : 0 \leq i < N \wedge 0 \leq j < N - i \}$$

$$A_2 = \{ W(i) \rightarrow a(i) : 0 \leq i < N \}$$

Application 2: Array Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);
```



Access relations:

$$A_1 = \{ F(i, j) \rightarrow a(i + j) : 0 \leq i < N \wedge 0 \leq j < N - i \}$$

$$A_2 = \{ W(i) \rightarrow a(i) : 0 \leq i < N \}$$

Map to all writes:

$$R' = A_1^{-1} \circ A_2 = \{ W(i) \rightarrow F(i', i - i') : 0 \leq i' \leq i < N \}$$

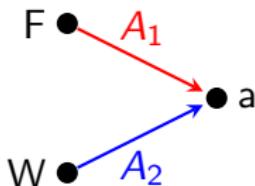
Application 2: Array Dataflow Analysis

Given a read from an array element, what was the *last* write to the same array element before the read?

```

for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);

```



Access relations:

$$A_1 = \{ F(i, j) \rightarrow a(i + j) : 0 \leq i < N \wedge 0 \leq j < N - i \}$$

$$A_2 = \{ W(i) \rightarrow a(i) : 0 \leq i < N \}$$

Map to all writes:

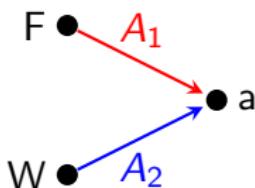
$$R' = A_1^{-1} \circ A_2 = \{ W(i) \rightarrow F(i', i - i') : 0 \leq i' \leq i < N \}$$

$$\text{Last write: } R = \text{lexmax } R' = \{ W(i) \rightarrow F(i, 0) : 0 \leq i < N \}$$

Application 2: Array Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);
```



Access relations:

$$A_1 = \{ F(i, j) \rightarrow a(i + j) : 0 \leq i < N \wedge 0 \leq j < N - i \}$$

$$A_2 = \{ W(i) \rightarrow a(i) : 0 \leq i < N \}$$

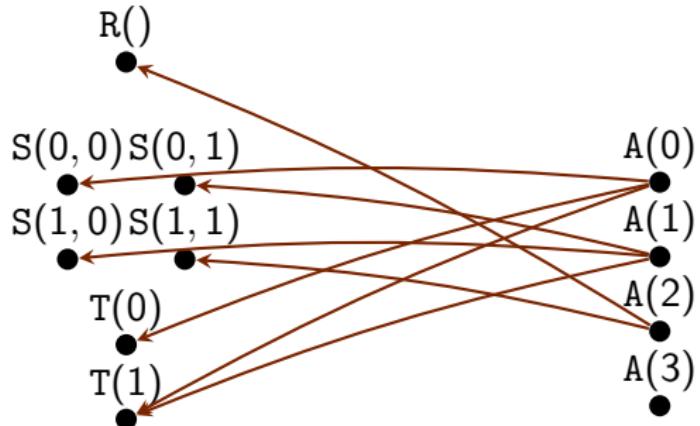
Map to all writes:

$$R' = A_1^{-1} \circ A_2 = \{ W(i) \rightarrow F(i', i - i') : 0 \leq i' \leq i < N \}$$

Last write: $R = \text{lexmax } R' = \{ W(i) \rightarrow F(i, 0) : 0 \leq i < N \}$

In general: impose lexicographical order on shared iterators

Cardinality

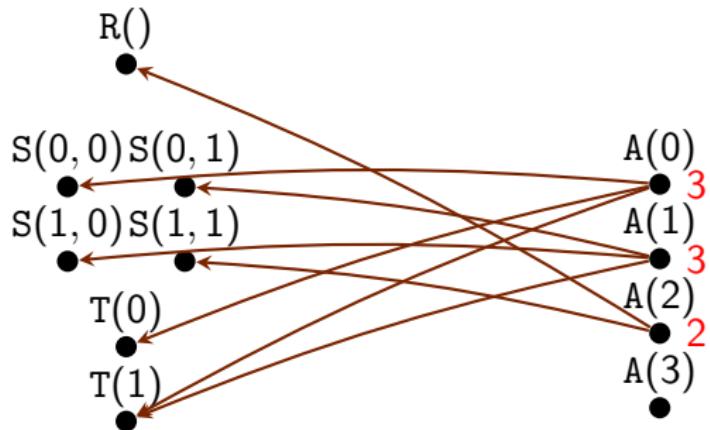


$(R \cup W)^{-1}$: statement instances accessing array element

$\{ A(2) \rightarrow R(); A(a) \rightarrow S(i, j) : a = i + j \wedge 0 \leq i < 2 \wedge 0 \leq j < 2;$

$A(0) \rightarrow T(k) : 0 \leq k < 2; A(k) \rightarrow T(k) : 0 \leq k < 2 \}$

Cardinality



$(R \cup W)^{-1}$: statement instances accessing array element

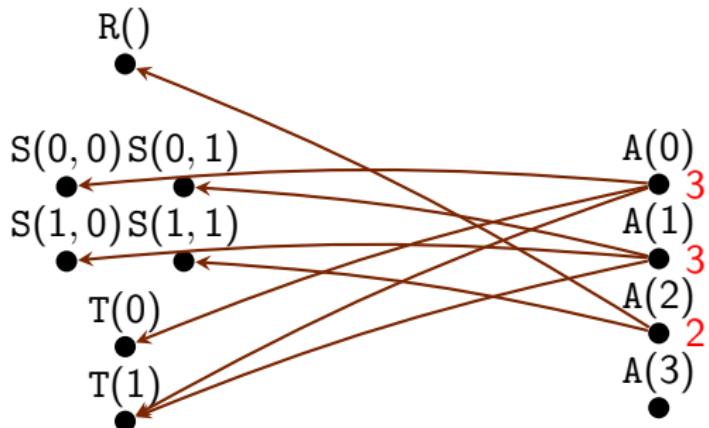
$\{ A(2) \rightarrow R(); A(a) \rightarrow S(i,j) : a = i + j \wedge 0 \leq i < 2 \wedge 0 \leq j < 2;$

$A(0) \rightarrow T(k) : 0 \leq k < 2; A(k) \rightarrow T(k) : 0 \leq k < 2 \}$

$\text{card}(R \cup W)^{-1}$: number of statement instances accessing array element

$\{ A(0) \rightarrow 3; A(1) \rightarrow 3; A(2) \rightarrow 2; \}$

Cardinality



$(R \cup W)^{-1}$: statement instances accessing array element

$\{ A(2) \rightarrow R(); A(a) \rightarrow S(i,j) : a = i + j \wedge 0 \leq i < 2 \wedge 0 \leq j < 2;$

$A(0) \rightarrow T(k) : 0 \leq k < 2; A(k) \rightarrow T(k) : 0 \leq k < 2 \}$

$\text{card}(R \cup W)^{-1}$: number of statement instances accessing array element

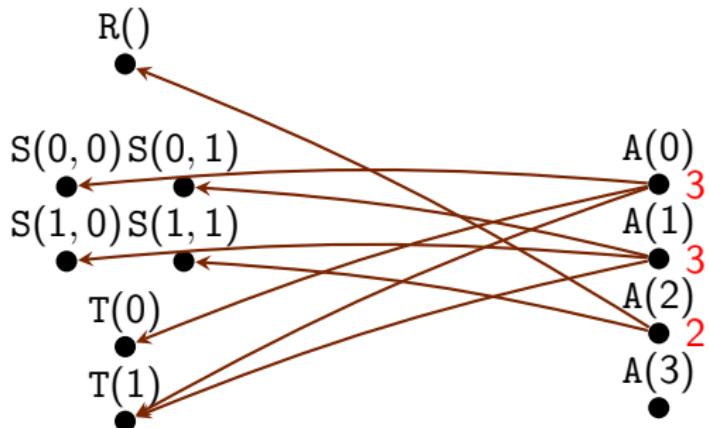
$\{ A(0) \rightarrow 3; A(1) \rightarrow 3; A(2) \rightarrow 2; \}$

$$R = \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \}$$

$$\text{card } R = \{ S(\mathbf{i}) \rightarrow n : n = \#\mathbf{j} : f(\mathbf{i}, \mathbf{j}) \}$$

$$\text{card } \bigcup_i R_i := \sum_i \text{card } R_i$$

Cardinality



$(R \cup W)^{-1}$: statement instances accessing array element

$\{ A(2) \rightarrow R(); A(a) \rightarrow S(i,j) : a = i + j \wedge 0 \leq i < 2 \wedge 0 \leq j < 2;$

$A(0) \rightarrow T(k) : 0 \leq k < 2; A(k) \rightarrow T(k) : 0 \leq k < 2 \}$

$\text{card}(R \cup W)^{-1}$: number of statement instances accessing array element

$\{ A(0) \rightarrow 3; A(1) \rightarrow 3; A(2) \rightarrow 2; \}$

$$R = \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \}$$

$$\text{card } R = \{ S(\mathbf{i}) \rightarrow n : n = \#\mathbf{j} : f(\mathbf{i}, \mathbf{j}) \}$$

\Rightarrow not a Presburger formula

$$\text{card } \bigcup_i R_i := \sum_i \text{card } R_i$$

Cardinality Examples

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

$$\text{card}\{(i, j) : 0 \leq i < N \wedge 0 \leq j < N - i\}$$

$$\Rightarrow \left\{ \frac{N+N^2}{2} : N \geq 1 \right\}$$

Cardinality Examples

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

$$\begin{aligned} \text{card}\{(i, j) : 0 \leq i < N \wedge 0 \leq j < N - i\} \\ \Rightarrow \left\{ \frac{N+N^2}{2} : N \geq 1 \right\} \end{aligned}$$

- How many times is a given array element written?

$$\begin{aligned} \text{card}(\{(i, j) \rightarrow a(i + j) : 0 \leq i < N \wedge 0 \leq j < N - i\})^{-1} \\ \Rightarrow \{a(a) \rightarrow 1 + a : 0 \leq a < N\} \end{aligned}$$

Cardinality Examples

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

$$\begin{aligned} \text{card}\{(i, j) : 0 \leq i < N \wedge 0 \leq j < N - i\} \\ \Rightarrow \left\{ \frac{N+N^2}{2} : N \geq 1 \right\} \end{aligned}$$

- How many times is a given array element written?

$$\begin{aligned} \text{card}(\{(i, j) \rightarrow a(i + j) : 0 \leq i < N \wedge 0 \leq j < N - i\})^{-1} \\ \Rightarrow \{a(a) \rightarrow 1 + a : 0 \leq a < N\} \end{aligned}$$

- How many array elements are written?

$$\begin{aligned} \text{card}(\text{ran}\{(i, j) \rightarrow a(i + j) : 0 \leq i < N \wedge 0 \leq j < N - i\}) \\ \Rightarrow \{N : N \geq 1\} \end{aligned}$$

Cardinality Examples (2)

How many times is S1 executed ?

```
for (i = max(0,N-M); i <= N-M+3; i++)
    for (j = 0; j <= N-2*i; j++)
        S1;
```

$$\text{card}\{(i, j) : 0, N - M \leq i \leq N - M + 3 \wedge 0 \leq j \leq N - 2i\}$$

$$\begin{cases} -4N + 8M - 8 & \text{if } M \leq N \leq 2M - 6 \\ MN - 2N - M^2 + 6M - 8 & \text{if } N \leq M \leq N + 3 \wedge N \leq 2M \\ \frac{N^2}{4} + \frac{3}{4}N + \frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor + 1 & \text{if } 0 \leq N \leq M \wedge 2M \leq N + 6 \\ \frac{N^2}{4} - MN - \frac{5}{4}N + M^2 + 2M + \frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor + 1 & \text{if } M \leq N \leq 2M \leq N + 6 \end{cases}$$

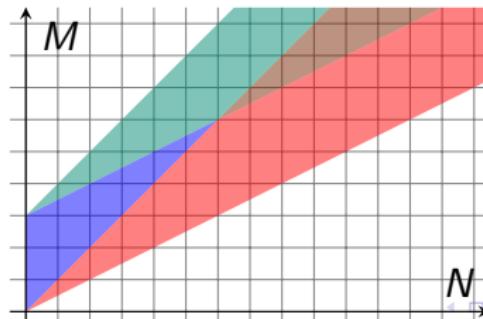
Cardinality Examples (2)

How many times is S1 executed ?

```
for (i = max(0,N-M); i <= N-M+3; i++)
    for (j = 0; j <= N-2*i; j++)
        S1;
```

$$\text{card}\{(i,j) : 0, N - M \leq i \leq N - M + 3 \wedge 0 \leq j \leq N - 2i\}$$

$$\begin{cases} -4N + 8M - 8 & \text{if } M \leq N \leq 2M - 6 \\ MN - 2N - M^2 + 6M - 8 & \text{if } N \leq M \leq N + 3 \wedge N \leq 2M \\ \frac{N^2}{4} + \frac{3}{4}N + \frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor + 1 & \text{if } 0 \leq N \leq M \wedge 2M \leq N + 6 \\ \frac{N^2}{4} - MN - \frac{5}{4}N + M^2 + 2M + \frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor + 1 & \text{if } M \leq N \leq 2M \leq N + 6 \end{cases}$$



Cardinality Representation

- Integer quasi affine expression

$$\lfloor x/2 \rfloor + 3N$$

⇒ Presburger term

That is, a term constructed from variables, symbolic constants, integer constants, addition (+), subtraction (-) and integer division by a constant ($\lfloor \cdot/d \rfloor$)

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
 - ⇒ Presburger term
That is, a term constructed from variables, symbolic constants, integer constants, addition (+), subtraction (-) and integer division by a constant ($\lfloor \cdot/d \rfloor$)
- Rational polynomial expression $x^2 - N/2$
 - ⇒ a term constructed from variables, symbolic constants, rational constants, addition (+), subtraction (-) and multiplication (\cdot)

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
 - ⇒ Presburger term
That is, a term constructed from variables, symbolic constants, integer constants, addition (+), subtraction (−) and integer division by a constant ($\lfloor \cdot/d \rfloor$)
- Rational polynomial expression $x^2 - N/2$
 - ⇒ a term constructed from variables, symbolic constants, rational constants, addition (+), subtraction (−) and multiplication (\cdot)
- Quasi polynomial expression $(\lfloor x/2 \rfloor + 3N)^2 - N/2$
 - ⇒ a rational polynomial expression with variables replaced by integer quasi affine expressions

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
 ⇒ Presburger term
 That is, a term constructed from variables, symbolic constants, integer constants, addition (+), subtraction (−) and integer division by a constant ($\lfloor \cdot/d \rfloor$)
- Rational polynomial expression $x^2 - N/2$
 ⇒ a term constructed from variables, symbolic constants, rational constants, addition (+), subtraction (−) and multiplication (\cdot)
- Quasi polynomial expression $(\lfloor x/2 \rfloor + 3N)^2 - N/2$
 ⇒ a rational polynomial expression with variables replaced by integer quasi affine expressions
- Piecewise quasi affine/polynomial expression
 ⇒ a list of pairs of pairs of Presburger sets and quasi affine/polynomial expressions $E = (S_i, e_i)_i$, with S_i disjoint

$$E(\mathbf{j}) = \begin{cases} e_i(\mathbf{j}) & \text{if } \mathbf{j} \in S_i \\ \perp/0 & \text{otherwise} \end{cases}$$

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
 ⇒ Presburger term
 That is, a term constructed from variables, symbolic constants, integer constants, addition (+), subtraction (−) and integer division by a constant ($\lfloor \cdot/d \rfloor$)
- Rational polynomial expression $x^2 - N/2$
 ⇒ a term constructed from variables, symbolic constants, rational constants, addition (+), subtraction (−) and multiplication (\cdot)
- Quasi polynomial expression $(\lfloor x/2 \rfloor + 3N)^2 - N/2$
 ⇒ a rational polynomial expression with variables replaced by integer quasi affine expressions
- Piecewise quasi affine/polynomial expression
 ⇒ a list of pairs of pairs of Presburger sets and quasi affine/polynomial expressions $E = (S_i, e_i)_i$, with S_i disjoint

$$E(\mathbf{j}) = \begin{cases} e_i(\mathbf{j}) & \text{if } \mathbf{j} \in S_i \\ \perp/0 & \text{otherwise} \end{cases}$$

Note: in practice, cardinality result does not contain nested integer divisions

Basic Operations on Piecewise Expressions

- Piecewise (rational) quasi affine expressions

- ▶ addition (+)
- ▶ subtraction (-)
- ▶ negation (-)
- ▶ minimum (min), maximum (max)
- ▶ multiplication by constant ($\cdot d$)
- ▶ division by constant ($/d$)
- ▶ remainder on integer division by constant ($\text{mod } d$)
- ▶ floor ($\lfloor \cdot \rfloor$)
- ▶ ceiling ($\lceil \cdot \rceil$)

Basic Operations on Piecewise Expressions

- Piecewise (rational) quasi affine expressions
 - ▶ addition (+)
 - ▶ subtraction (-)
 - ▶ negation (-)
 - ▶ minimum (min), maximum (max)
 - ▶ multiplication by constant ($\cdot d$)
 - ▶ division by constant ($/d$)
 - ▶ remainder on integer division by constant ($\text{mod } d$)
 - ▶ floor ($\lfloor \cdot \rfloor$)
 - ▶ ceiling ($\lceil \cdot \rceil$)
- Piecewise quasi polynomial expressions
 - ▶ addition (+)
 - ▶ subtraction (-)
 - ▶ negation (-)
 - ▶ multiplication (\cdot)
 - ▶ exponentiation by positive integer constant (\cdot^d)

Application 3: Reuse Distance Computation

Given an access to a cache line ℓ , how many distinct cache lines have been accessed since the previous access to ℓ ?

⇒ Is the cache line still in the cache?

Application 3: Reuse Distance Computation

Given an access to a cache line ℓ , how many distinct cache lines have been accessed since the previous access to ℓ ?

⇒ Is the cache line still in the cache?

```
for (i = 0; i <= 7; ++i) {  
    a: A[i];  
    b: A[7-i];  
    if (i <= 3)  
        c: A[2*i];  
}
```

Assume $A[i]$ in cache line $\lfloor i/3 \rfloor$

Application 3: Reuse Distance Computation

Given an access to a cache line ℓ , how many distinct cache lines have been accessed since the previous access to ℓ ?

⇒ Is the cache line still in the cache?

```
for (i = 0; i <= 7; ++i) {
    a: A[i];
    b: A[7-i];
    if (i <= 3)
        c: A[2*i];
}
```

Assume $A[i]$ in cache line $\lfloor i/3 \rfloor$

i	0	1	2	3	4	5	6	7
r	a	b	c	a	b	c	a	b
$r@i$	0	7	0	1	6	2	2	5
$\lfloor (r@i)/3 \rfloor$	0	2	0	0	2	0	0	1
distance	0	0	2	1	2	2	1	3

Reuse Distance Computation

```
for (i = 0; i <= 7; ++i) {  
    a: A[i];  
    b: A[7-i];  
    if (i <= 3)  
        c: A[2*i];  
}
```

Assume $A[i]$ in cache line $\lfloor i/3 \rfloor$

Reuse Distance Computation

```
for (i = 0; i <= 7; ++i) {  
    a: A[i];  
    b: A[7-i];  
    if (i <= 3)  
        c: A[2*i];  
}
```

Assume $A[i]$ in cache line $\lfloor i/3 \rfloor$

$$I = \{ a(i) : 0 \leq i \leq 7; b(i) : 0 \leq i \leq 7; c(i) : 0 \leq i \leq 3; \}$$

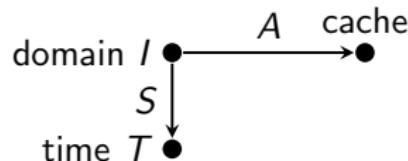
$$R = \{ a(i) \rightarrow A(i); b(i) \rightarrow A(7 - i); c(i) \rightarrow A(2i) \} \cap_{\text{dom}} I$$

$$S = \{ a(i) \rightarrow (i, 0); b(i) \rightarrow (i, 1); c(i) \rightarrow (i, 2) \}$$

$$C = \{ A(a) \rightarrow L(\lfloor a/3 \rfloor) \}$$

$$A = C \circ R$$

$$T = \text{ran } S$$



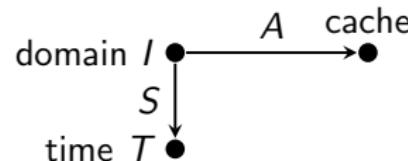
Reuse Distance Computation

```

for (i = 0; i <= 7; ++i) {
a: A[i];
b: A[7-i];
if (i <= 3)
c: A[2*i];
}

```

Assume $A[i]$ in cache line $\lfloor i/3 \rfloor$



$$I = \{ a(i) : 0 \leq i \leq 7; b(i) : 0 \leq i \leq 7; c(i) : 0 \leq i \leq 3; \}$$

$$R = \{ a(i) \rightarrow A(i); b(i) \rightarrow A(7-i); c(i) \rightarrow A(2i) \} \cap_{\text{dom}} I$$

$$S = \{ a(i) \rightarrow (i, 0); b(i) \rightarrow (i, 1); c(i) \rightarrow (i, 2) \}$$

$$C = \{ A(a) \rightarrow L(\lfloor a/3 \rfloor) \}$$

$$A = C \circ R$$

$$T = \text{ran } S$$

$$\text{next} = S^{-1} \circ (\text{lexmin} ((S \circ (A^{-1} \circ A) \circ S^{-1}) \cap (T \prec T))) \circ S$$

$$\text{before} = S^{-1} \circ (T \succcurlyeq T) \circ S$$

$$\text{after_prev} = \text{before}^{-1} \circ \text{next}^{-1}$$

$$\text{distance} = \text{card}(A \circ (\text{after_prev} \cap \text{before}))$$

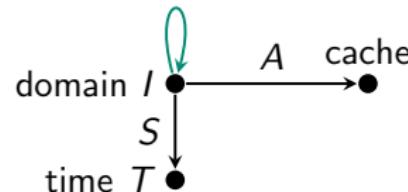
Reuse Distance Computation

```

for (i = 0; i <= 7; ++i) {
    a: A[i];
    b: A[7-i];
    if (i <= 3)
        c: A[2*i];
}

```

Assume $A[i]$ in cache line $\lfloor i/3 \rfloor$



$$I = \{ a(i) : 0 \leq i \leq 7; b(i) : 0 \leq i \leq 7; c(i) : 0 \leq i \leq 3; \}$$

$$R = \{ a(i) \rightarrow A(i); b(i) \rightarrow A(7-i); c(i) \rightarrow A(2i) \} \cap_{\text{dom}} I$$

$$S = \{ a(i) \rightarrow (i, 0); b(i) \rightarrow (i, 1); c(i) \rightarrow (i, 2) \}$$

$$C = \{ A(a) \rightarrow L(\lfloor a/3 \rfloor) \}$$

$$A = C \circ R$$

$$T = \text{ran } S$$

$$\text{next} = S^{-1} \circ (\text{lexmin} ((S \circ (A^{-1} \circ A) \circ S^{-1}) \cap (T \prec T))) \circ S$$

before = pair of statement instances that access same cache line

$$\text{after_prev} = \text{before}^{-1} \circ \text{next}^{-1}$$

$$\text{distance} = \text{card}(A \circ (\text{after_prev} \cap \text{before}))$$

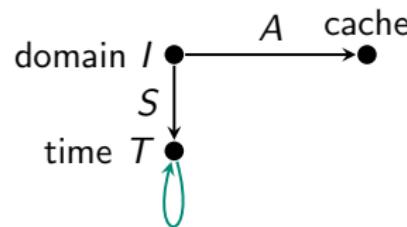
Reuse Distance Computation

```

for (i = 0; i <= 7; ++i) {
    a: A[i];
    b: A[7-i];
    if (i <= 3)
        c: A[2*i];
}

```

Assume $A[i]$ in cache line $\lfloor i/3 \rfloor$



$$I = \{ a(i) : 0 \leq i \leq 7; b(i) : 0 \leq i \leq 7; c(i) : 0 \leq i \leq 3; \}$$

$$R = \{ a(i) \rightarrow A(i); b(i) \rightarrow A(7-i); c(i) \rightarrow A(2i) \} \cap_{\text{dom}} I$$

$$S = \{ a(i) \rightarrow (i, 0); b(i) \rightarrow (i, 1); c(i) \rightarrow (i, 2) \}$$

$$C = \{ A(a) \rightarrow L(\lfloor a/3 \rfloor) \}$$

$$A = C \circ R$$

$$T = \text{ran } S$$

$$\text{next} = S^{-1} \circ (\text{lexmin} ((S \circ (A^{-1} \circ A) \circ S^{-1}) \cap (T \prec T))) \circ S$$

$\text{before}_{\text{line}}$ schedule times of pair of statement instances that access same cache

$$\text{after_prev} = \text{before}^{-1} \circ \text{next}^{-1}$$

$$\text{distance} = \text{card}(A \circ (\text{after_prev} \cap \text{before}))$$

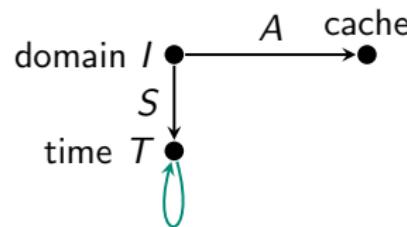
Reuse Distance Computation

```

for (i = 0; i <= 7; ++i) {
a: A[i];
b: A[7-i];
if (i <= 3)
c: A[2*i];
}

```

Assume $A[i]$ in cache line $\lfloor i/3 \rfloor$



$$I = \{ a(i) : 0 \leq i \leq 7; b(i) : 0 \leq i \leq 7; c(i) : 0 \leq i \leq 3; \}$$

$$R = \{ a(i) \rightarrow A(i); b(i) \rightarrow A(7-i); c(i) \rightarrow A(2i) \} \cap_{\text{dom}} I$$

$$S = \{ a(i) \rightarrow (i, 0); b(i) \rightarrow (i, 1); c(i) \rightarrow (i, 2) \}$$

$$C = \{ A(a) \rightarrow L(\lfloor a/3 \rfloor) \}$$

$$A = C \circ R$$

$$T = \text{ran } S$$

$$\text{next} = S^{-1} \circ (\text{lexmin} ((S \circ (A^{-1} \circ A) \circ S^{-1}) \cap (T \prec T))) \circ S$$

before schedule times of pair of statement instances that access same cache

after_prev line such that first is executed before second

$$\text{distance} = \text{card}(A \circ (\text{after_prev} \cap \text{before}))$$

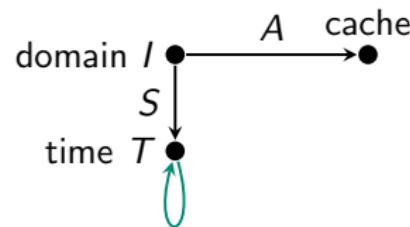
Reuse Distance Computation

```

for (i = 0; i <= 7; ++i) {
    a: A[i];
    b: A[7-i];
    if (i <= 3)
        c: A[2*i];
}

```

Assume $A[i]$ in cache line $\lfloor i/3 \rfloor$



$$I = \{ a(i) : 0 \leq i \leq 7; b(i) : 0 \leq i \leq 7; c(i) : 0 \leq i \leq 3; \}$$

$$R = \{ a(i) \rightarrow A(i); b(i) \rightarrow A(7-i); c(i) \rightarrow A(2i) \} \cap_{\text{dom}} I$$

$$S = \{ a(i) \rightarrow (i, 0); b(i) \rightarrow (i, 1); c(i) \rightarrow (i, 2) \}$$

$$C = \{ A(a) \rightarrow L(\lfloor a/3 \rfloor) \}$$

$$A = C \circ R$$

$$T = \text{ran } S$$

$$\text{next} = S^{-1} \circ \left(\text{lexmin} \left((S \circ (A^{-1} \circ A) \circ S^{-1}) \cap (T \prec T) \right) \right) \circ S$$

before schedule times of a statement instance and the next statement instance

after $_p$ that accesses same cache line

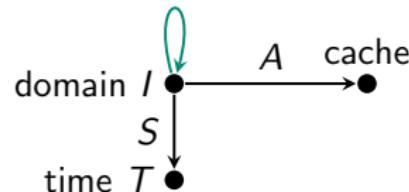
$$\text{distance} = \text{card}(A \circ (\text{after_prev} \cap \text{before}))$$

Reuse Distance Computation

```

for (i = 0; i <= 7; ++i) {
    a: A[i];
    b: A[7-i];
    if (i <= 3)
        c: A[2*i];
}

```



Assume $A[i]$ in cache line $\lfloor i/3 \rfloor$

$$I = \{ a(i) : 0 \leq i \leq 7; b(i) : 0 \leq i \leq 7; c(i) : 0 \leq i \leq 3; \}$$

$$R = \{ a(i) \rightarrow A(i); b(i) \rightarrow A(7-i); c(i) \rightarrow A(2i) \} \cap_{\text{dom}} I$$

$$S = \{ a(i) \rightarrow (i, 0); b(i) \rightarrow (i, 1); c(i) \rightarrow (i, 2) \}$$

$$C = \{ A(a) \rightarrow L(\lfloor a/3 \rfloor) \}$$

$$A = C \circ R$$

$$T = \text{ran } S$$

$$\text{next} = S^{-1} \circ (\text{lexmin} ((S \circ (A^{-1} \circ A) \circ S^{-1}) \cap (T \prec T))) \circ S$$

before (statement instance) and the next statement instance that accesses after same cache line $\circ \text{next}^{-1}$

$$\text{distance} = \text{card}(A \circ (\text{after_prev} \cap \text{before}))$$

Reuse Distance Computation

```

for (i = 0; i <= 7; ++i) {
    a: A[i];
    b: A[7-i];
    if (i <= 3)
        c: A[2*i];
}

```

Assume $A[i]$ in cache line $\lfloor i/3 \rfloor$

$$I = \{ a(i) : 0 \leq i \leq 7; b(i) : 0 \leq i \leq 7; c(i) : 0 \leq i \leq 3; \}$$

$$R = \{ a(i) \rightarrow A(i); b(i) \rightarrow A(7-i); c(i) \rightarrow A(2i) \} \cap_{\text{dom}} I$$

$$S = \{ a(i) \rightarrow (i, 0); b(i) \rightarrow (i, 1); c(i) \rightarrow (i, 2) \}$$

$$C = \{ A(a) \rightarrow L(\lfloor a/3 \rfloor) \}$$

$$A = C \circ R$$

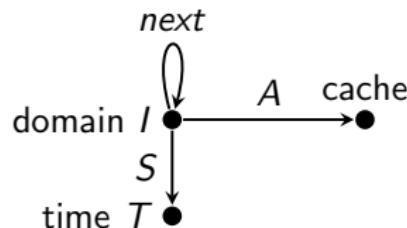
$$T = \text{ran } S$$

$$\text{next} = S^{-1} \circ (\text{lexmin} ((S \circ (A^{-1} \circ A) \circ S^{-1}) \cap (T \prec T))) \circ S$$

$$\text{before} = S^{-1} \circ (T \succcurlyeq T) \circ S$$

$$\text{after_prev} = \text{before}^{-1} \circ \text{next}^{-1}$$

$$\text{distance} = \text{card}(A \circ (\text{after_prev} \cap \text{before}))$$



Bounds on Piecewise Quasi Polynomials

$$m(N) = \max_{(x,y):x,y \geq 0 \wedge x+y \leq N} 4+x+y-(x-2)^2$$

Question

Can exact maximum be computed in general?

Bounds on Piecewise Quasi Polynomials

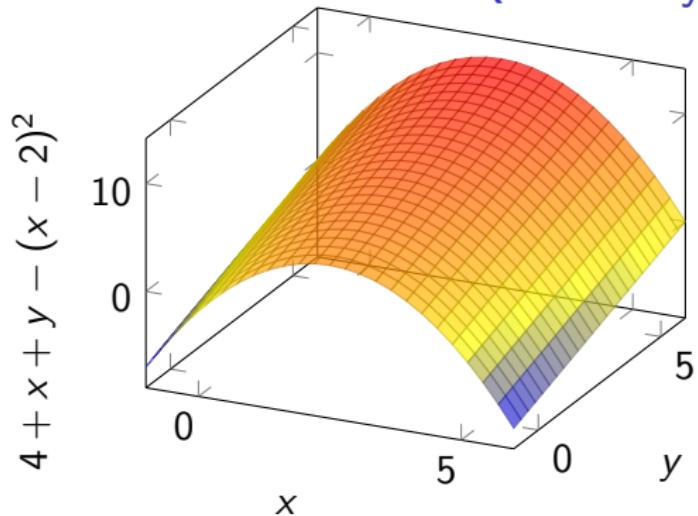
$$m(N) = \max_{(x,y):x,y \geq 0 \wedge x+y \leq N} 4+x+y-(x-2)^2$$

Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials



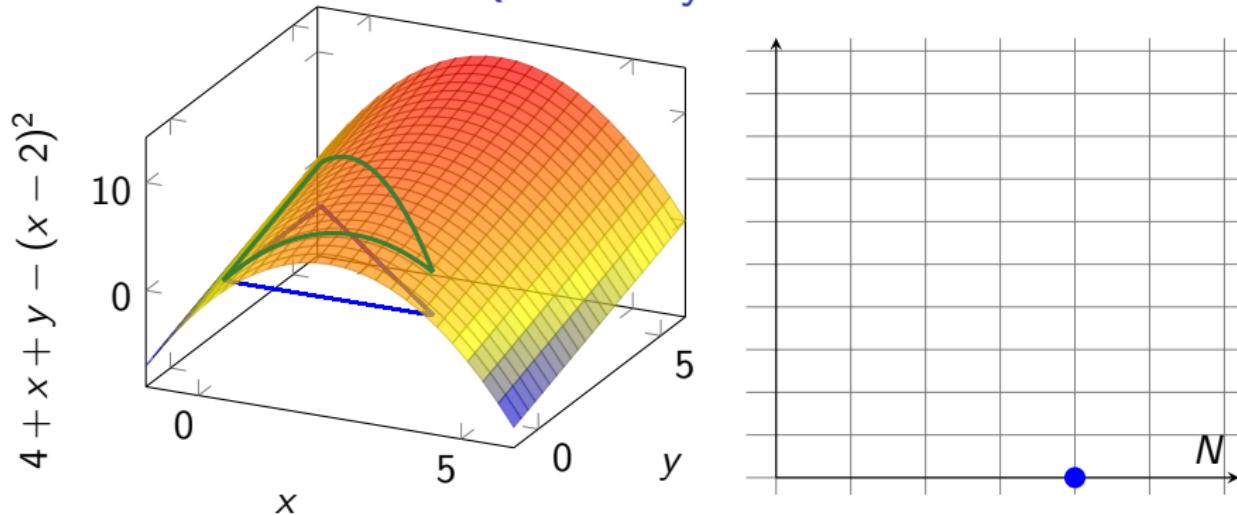
$$m(N) = \max_{(x,y):x,y \geq 0 \wedge x+y \leq N} 4+x+y-(x-2)^2$$

Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials



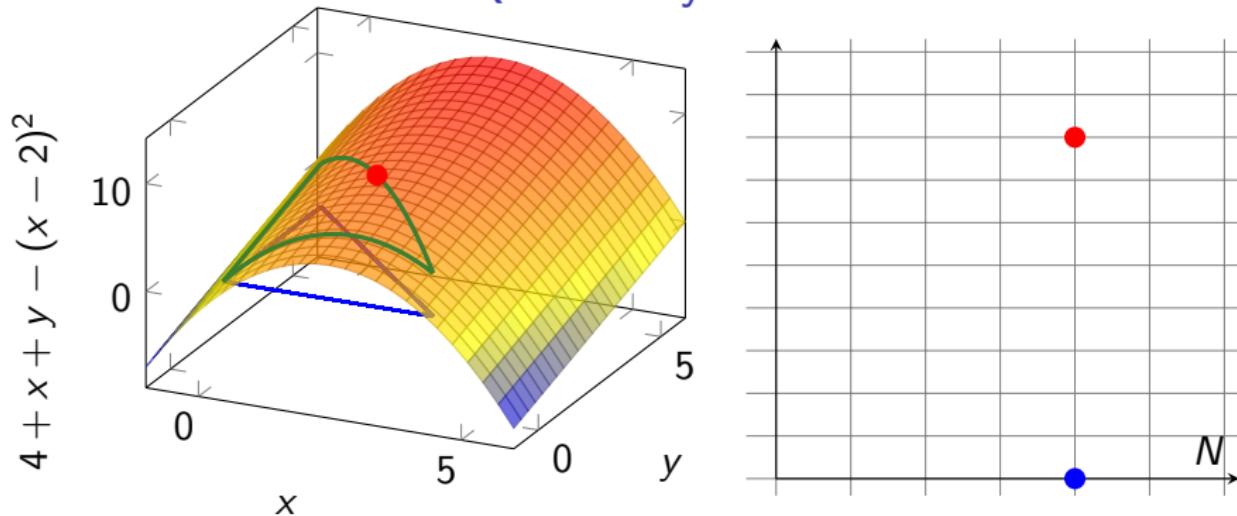
$$m(N) = \max_{(x,y): x,y \geq 0 \wedge x+y \leq N} 4+x+y-(x-2)^2$$

Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials



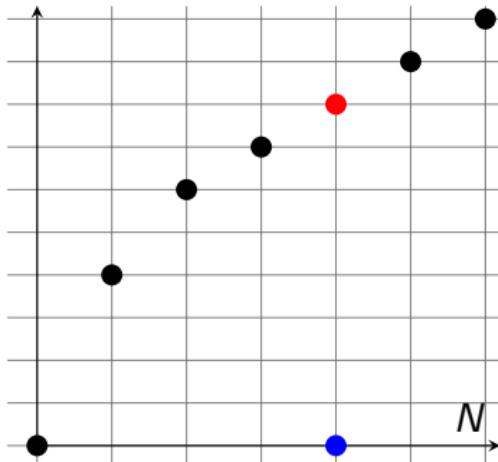
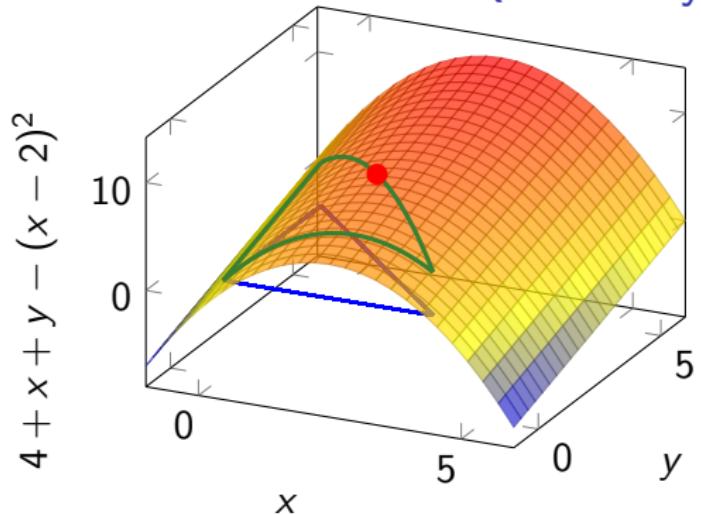
$$m(N) = \max_{(x,y): x,y \geq 0 \wedge x+y \leq N} 4+x+y-(x-2)^2$$

Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials



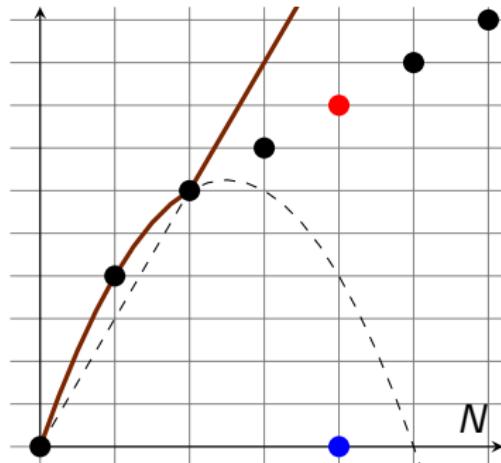
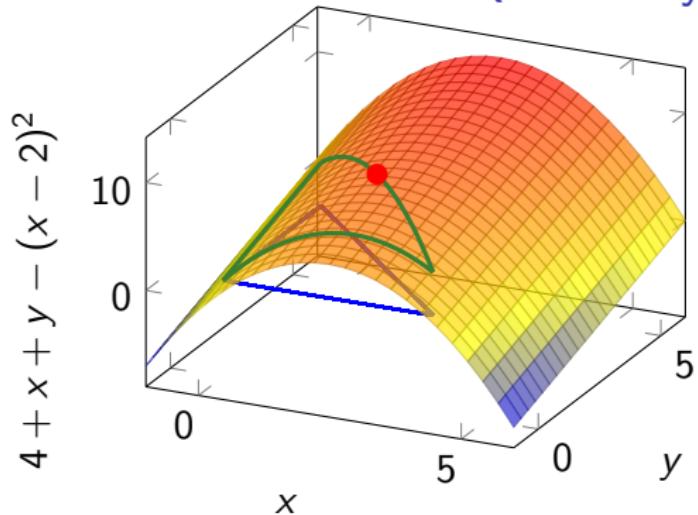
$$m(N) = \max_{(x,y):x,y \geq 0 \wedge x+y \leq N} 4+x+y-(x-2)^2$$

Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials



$$m(N) = \max_{(x,y):x,y \geq 0 \wedge x+y \leq N} 4+x+y-(x-2)^2 \leq u(N) = \max(3N, 5N - N^2)$$

Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials — Example

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

Bounds on Piecewise Quasi Polynomials — Example

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

$$\text{ub} \left\{ ij + i - N + 1 \quad \text{if } 0 \leq i < N \wedge i \leq j < N \right.$$

Bounds on Piecewise Quasi Polynomials — Example

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

$$\text{ub} \left\{ ij + i - N + 1 \quad \text{if } 0 \leq i < N \wedge i \leq j < N \right.$$

Result:

$$\left\{ \max(1 - 2N + N^2) \quad \text{if } N \geq 1 \right.$$

(exact maximum)

Application 4: Maximal Number of Live Memory elements

- Assume each statement instance writes to at most one array element
 - ⇒ Each live element can be identified by write instance
- Compute dataflow relation D
- For each write instance compute last read
$$L = S^{-1} \circ \text{lexmax}(S \circ D)$$
- For each statement instance i , count **write** instances that precede i such that **corresponding last read follows** i
 - ⇒ Number of live elements at i
- Compute upper bound
$$U = \text{ub } N$$

Maximal Number of Live Memory elements — Example

```
for (i = 0; i < N; ++i)
S1:      t[i] = f(a[i]);
for (i = 0; i < N; ++i)
S2:      b[i] = g(t[N-i-1]);
```

$$I = \{ S1(i) : 0 \leq i < N; S2(i) : 0 \leq i < N \}$$

$$S = \{ S1(i) \rightarrow (0, i); S2(i) \rightarrow (1, i) \}$$

$$D = \{ S1(i) \rightarrow S2(N - 1 - i) : 0 \leq i < N \}$$

Maximal Number of Live Memory elements — Example

```
for (i = 0; i < N; ++i)
```

```
S1:      t[i] = f(a[i]);
```

```
for (i = 0; i < N; ++i)
```

```
S2:      b[i] = g(t[N-i-1]);
```

$$I = \{ S1(i) : 0 \leq i < N; S2(i) : 0 \leq i < N \}$$

$$S = \{ S1(i) \rightarrow (0, i); S2(i) \rightarrow (1, i) \}$$

$$D = \{ S1(i) \rightarrow S2(N - 1 - i) : 0 \leq i < N \}$$

$$L = S^{-1} \circ \text{lexmax}(S \circ D)$$

$$= \{ S1(i) \rightarrow S2(N - 1 - i) : 0 \leq i < N \}$$

$$N = \text{card}(((S \succ S) \cap (I \rightarrow (\text{dom } L))) \cap (L^{-1} \circ ((S \preccurlyeq S) \cap (I \rightarrow (\text{ran } L)))))$$

$$= \{ S1(i) \rightarrow i : 1 \leq i < N; S2(i) \rightarrow N - i : 0 \leq i < N \}$$

$$U = \text{ub } N$$

$$= \{ \max(N) : N \geq 1 \}$$

Nested Relations

Assume that we want to count the number of **statement instances** between any given **pair of statement instances**

- ⇒ we need to create a relation between a **pair of statement instances** and a third **statement instance**
- ⇒ nested relations

Nested Relations

Assume that we want to count the number of **statement instances** between any given **pair of statement instances**

- ⇒ we need to create a relation between a **pair of statement instances** and a third **statement instance**
- ⇒ nested relations

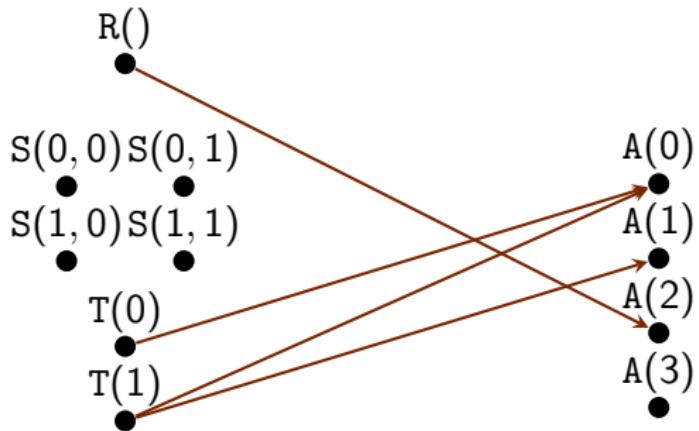
Example:

$$\{ (S(i) \rightarrow S(j)) \rightarrow S(k) : i \prec k \prec j \}$$

1D:

$$\begin{aligned} & \text{card}\{ ((i) \rightarrow (j)) \rightarrow (k) : 0 \leq i < k < j \leq n \} \\ & = \{ ((i) \rightarrow (j)) \rightarrow j - i - 1 : 0 \leq i \wedge i + 2 \leq j \leq n \} \end{aligned}$$

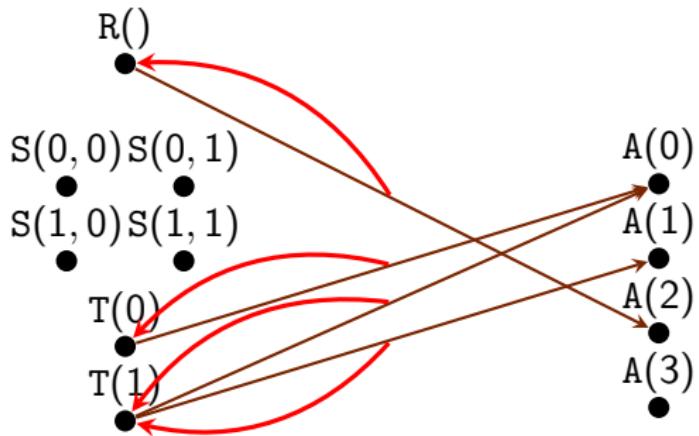
Domain Map



R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

Domain Map



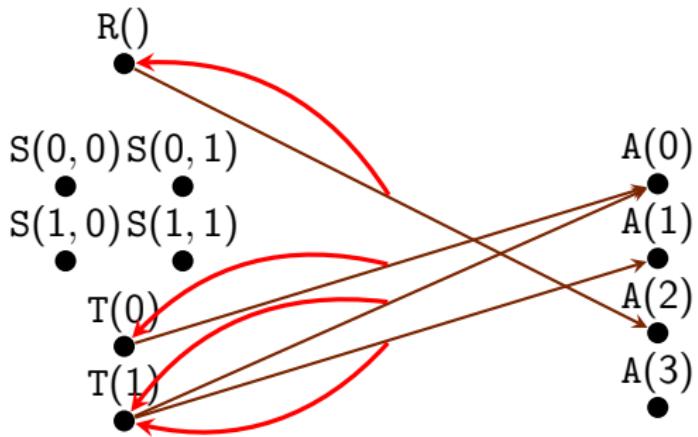
R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

dom R : statement instance of access

$\{ (R() \rightarrow A(2)) \rightarrow R(); (T(k) \rightarrow A(0)) \rightarrow T(k) : 0 \leq k < 2;$
 $(T(k) \rightarrow A(k)) \rightarrow T(k) : 0 \leq k < 2 \}$

Domain Map



R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

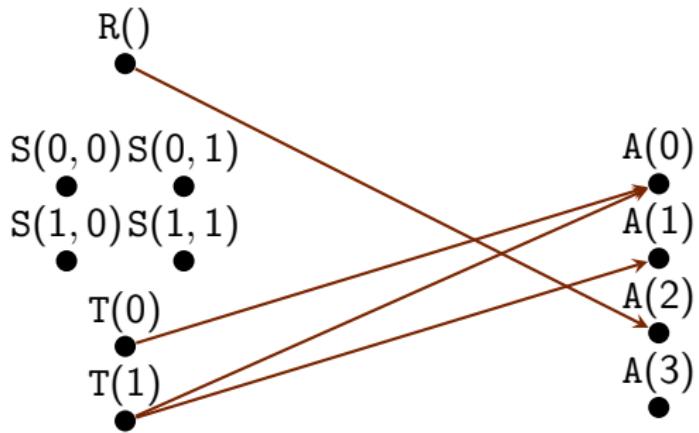
dom R : statement instance of access

$\{ (R() \rightarrow A(2)) \rightarrow R(); (T(k) \rightarrow A(0)) \rightarrow T(k) : 0 \leq k < 2;$
 $(T(k) \rightarrow A(k)) \rightarrow T(k) : 0 \leq k < 2 \}$

$R = \{ S(i) \rightarrow T(j) : f(i,j) \}$

dom $R = \{ (S(i) \rightarrow T(j)) \rightarrow S(i) : f(i,j) \}$

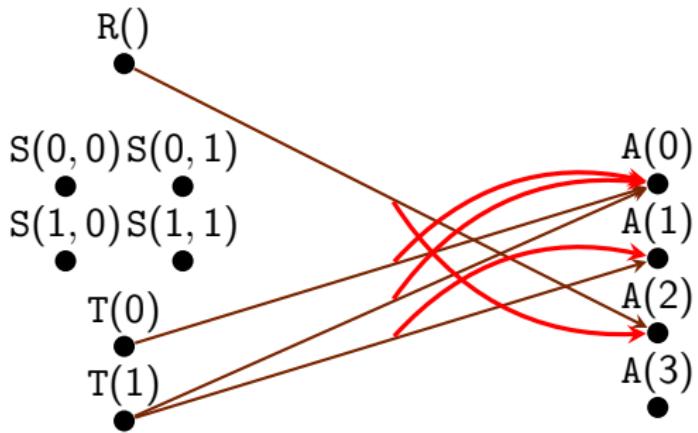
Range Map



R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

Range Map



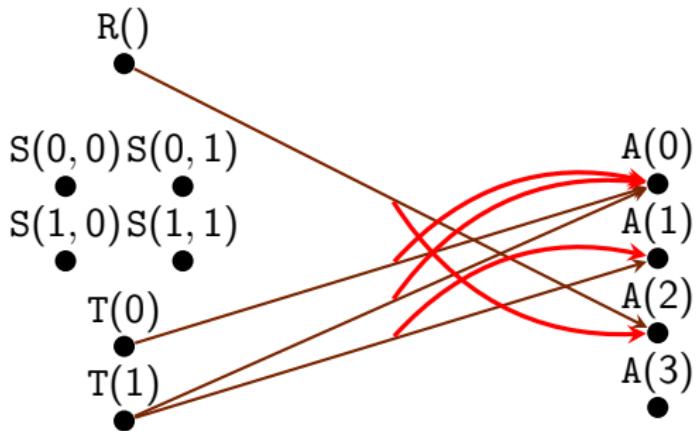
R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

$\overrightarrow{\text{ran } R}$: array element of access

$\{ (R() \rightarrow A(2)) \rightarrow A(2); (T(k) \rightarrow A(0)) \rightarrow A(0) : 0 \leq k < 2;$
 $(T(k) \rightarrow A(k)) \rightarrow A(k) : 0 \leq k < 2 \}$

Range Map



R : array elements read by statement instance

$\{ R() \rightarrow A(2); T(k) \rightarrow A(0) : 0 \leq k < 2; T(k) \rightarrow A(k) : 0 \leq k < 2 \}$

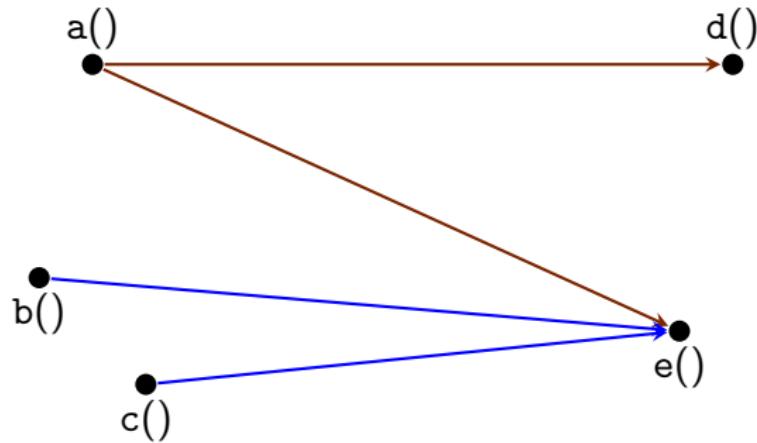
ran R : array element of access

$\{ (R() \rightarrow A(2)) \rightarrow A(2); (T(k) \rightarrow A(0)) \rightarrow A(0) : 0 \leq k < 2;$
 $(T(k) \rightarrow A(k)) \rightarrow A(k) : 0 \leq k < 2 \}$

$R = \{ S(i) \rightarrow T(j) : f(i,j) \}$

ran $R = \{ (S(i) \rightarrow T(j)) \rightarrow T(j) : f(i,j) \}$

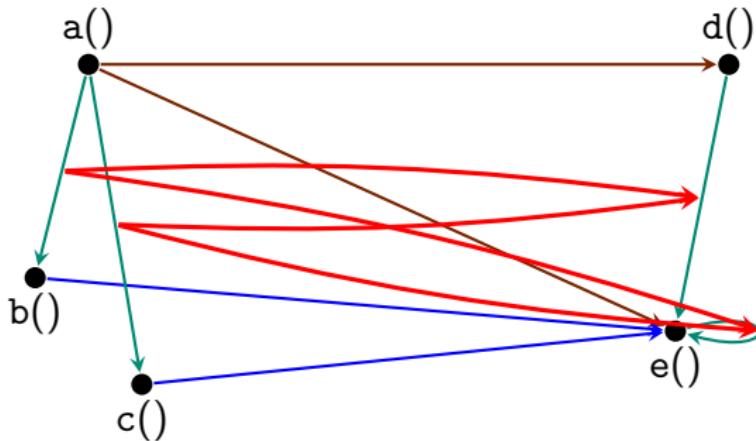
Product



$$A = \{ a() \rightarrow d(); a() \rightarrow e() \}$$

$$B = \{ b() \rightarrow e(); c() \rightarrow e() \}$$

Product

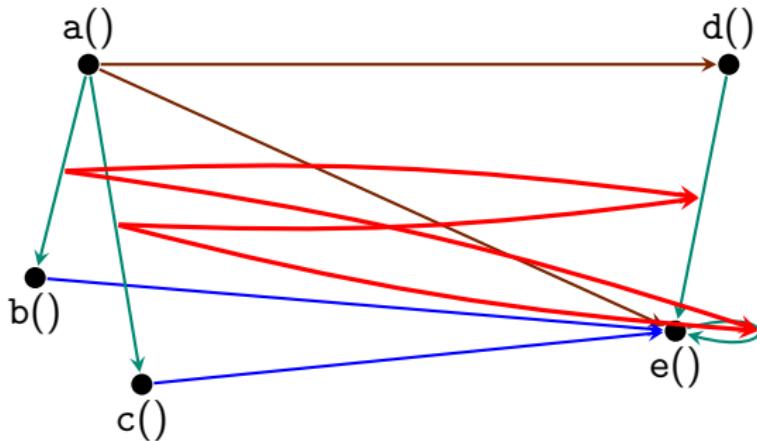


$$A = \{ a() \rightarrow d(); a() \rightarrow e() \}$$

$$B = \{ b() \rightarrow e(); c() \rightarrow e() \}$$

$$A \times B = \{ (a() \rightarrow b()) \rightarrow (d \rightarrow e()); (a() \rightarrow c()) \rightarrow (d \rightarrow e()); (a() \rightarrow b()) \rightarrow (e \rightarrow e()); (a() \rightarrow c()) \rightarrow (e \rightarrow e()) \}$$

Product



$$A = \{ a() \rightarrow d(); a() \rightarrow e() \}$$

$$B = \{ b() \rightarrow e(); c() \rightarrow e() \}$$

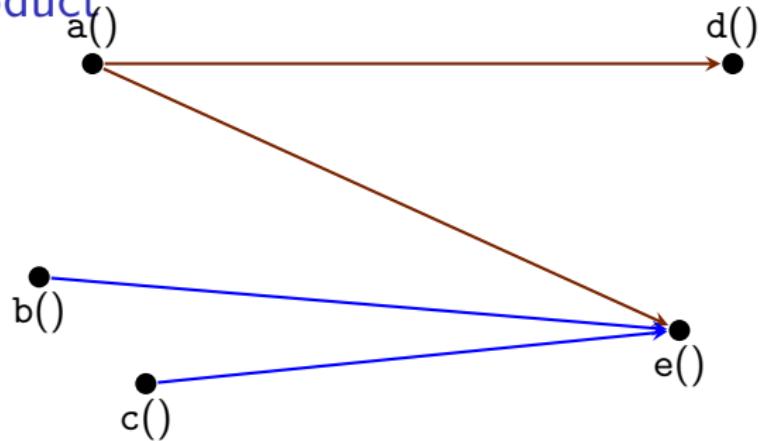
$$A \times B = \{ (a() \rightarrow b()) \rightarrow (d \rightarrow e()); (a() \rightarrow c()) \rightarrow (d \rightarrow e()); \\ (a() \rightarrow b()) \rightarrow (e \rightarrow e()); (a() \rightarrow c()) \rightarrow (e \rightarrow e()) \}$$

$$A = \{ S_1(\mathbf{i}_1) \rightarrow T_1(\mathbf{j}_1) : f(\mathbf{i}_1, \mathbf{j}_1) \}$$

$$B = \{ S_2(\mathbf{i}_2) \rightarrow T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

$$A \times B = \{ (S_1(\mathbf{i}_1) \rightarrow S_2(\mathbf{i}_2)) \rightarrow (T_1(\mathbf{j}_1) \rightarrow T_2(\mathbf{j}_2)) : f(\mathbf{i}_1, \mathbf{j}_1) \wedge g(\mathbf{i}_2, \mathbf{j}_2) \}$$

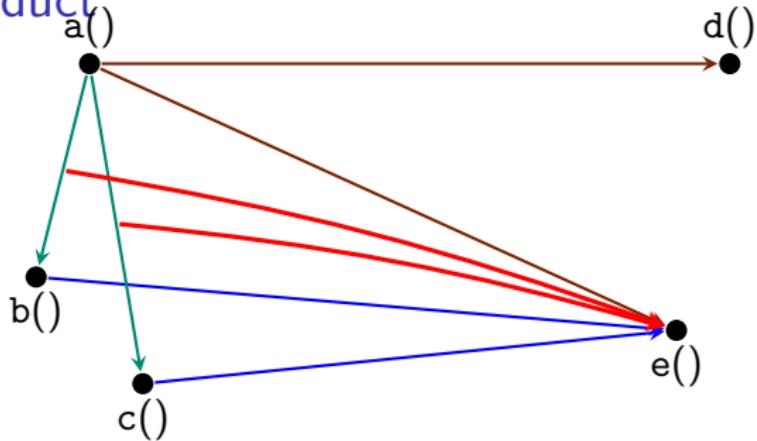
Domain Product



$$A = \{ a() \rightarrow d(); a() \rightarrow e() \}$$

$$B = \{ b() \rightarrow e(); c() \rightarrow e() \}$$

Domain Product

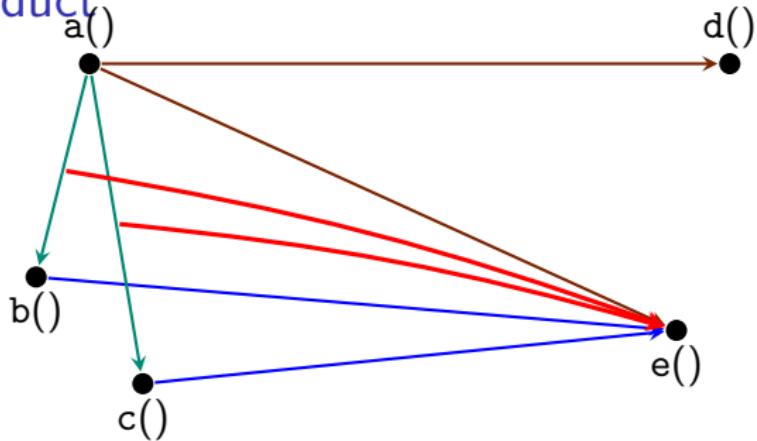


$$A = \{ a() \rightarrow d(); a() \rightarrow e() \}$$

$$B = \{ b() \rightarrow e(); c() \rightarrow e() \}$$

$$A \bowtie B = \{ (a() \rightarrow b()) \rightarrow e(); (a() \rightarrow c()) \rightarrow e() \}$$

Domain Product



$$A = \{ \text{ a}() \rightarrow \text{ d}(); \text{ a}() \rightarrow \text{ e}() \}$$

$$B = \{ \text{b}() \rightarrow \text{e}(); \text{c}() \rightarrow \text{e}() \}$$

$$A \bowtie B = \{ (a() \rightarrow b()) \rightarrow e(); (a() \rightarrow c()) \rightarrow e() \}$$

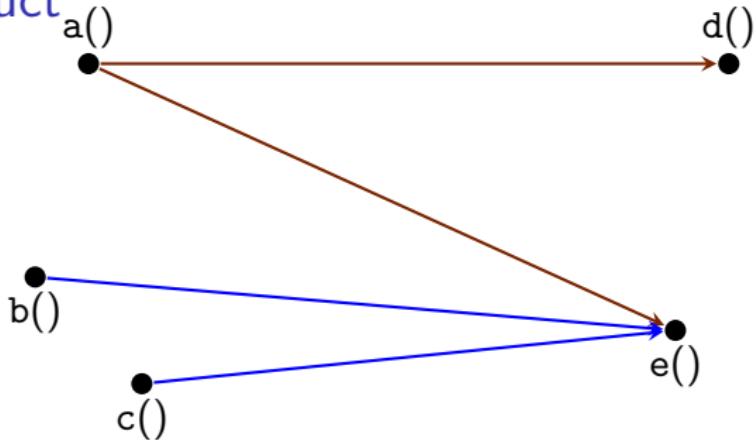
$$A = \{ S_1(\mathbf{i}_1) \rightarrow T_1(\mathbf{j}_1) : f(\mathbf{i}_1, \mathbf{j}_1) \}$$

$$B = \{ S_2(\mathbf{i}_2) \rightarrow T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

$$A \times B =$$

$$\begin{cases} \{ (S_1(\mathbf{i}_1) \rightarrow S_2(\mathbf{i}_2)) \rightarrow T_1(\mathbf{j}) : f(\mathbf{i}_1, \mathbf{j}) \wedge g(\mathbf{i}_2, \mathbf{j}) \} & T_1(\mathbf{j}_1) = T_2(\mathbf{j}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

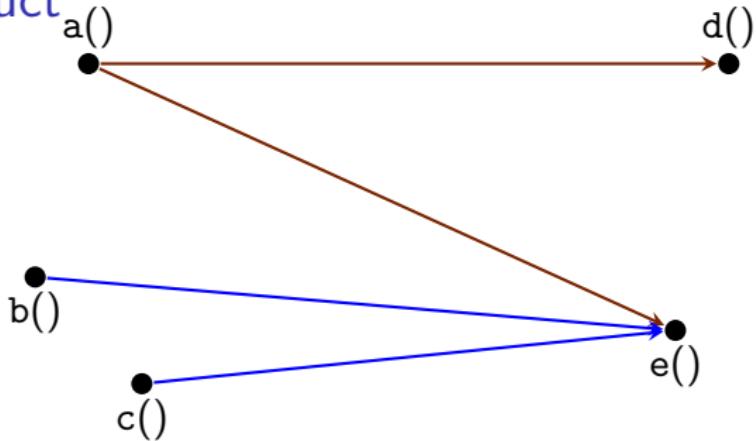
Range Product



$$A = \{ a() \rightarrow d(); a() \rightarrow e() \}$$

$$B = \{ b() \rightarrow e(); c() \rightarrow e() \}$$

Range Product

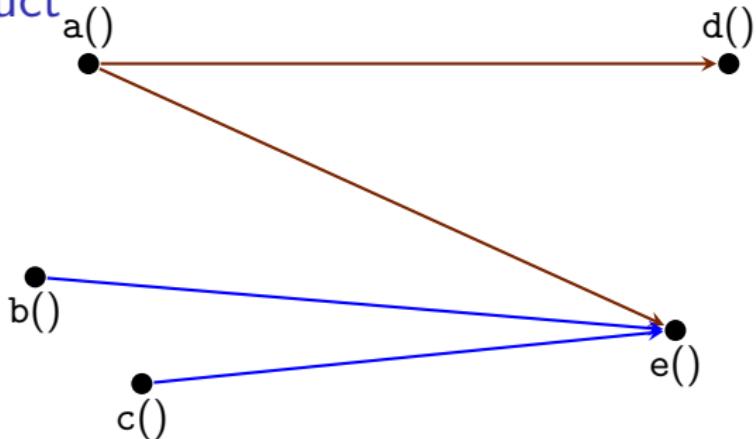


$$A = \{ a() \rightarrow d(); a() \rightarrow e() \}$$

$$B = \{ b() \rightarrow e(); c() \rightarrow e() \}$$

$$A \ltimes B = \{ \}$$

Range Product



$$A = \{ a() \rightarrow d(); a() \rightarrow e() \}$$

$$B = \{ b() \rightarrow e(); c() \rightarrow e() \}$$

$$A \bowtie B = \{ \}$$

$$A = \{ S_1(\mathbf{i}_1) \rightarrow T_1(\mathbf{j}_1) : f(\mathbf{i}_1, \mathbf{j}_1) \}$$

$$B = \{ S_2(\mathbf{i}_2) \rightarrow T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

$$A \bowtie B =$$

$$\begin{cases} \{ S_1(\mathbf{i}) \rightarrow (T_1(\mathbf{j}_1) \rightarrow T_2(\mathbf{j}_2)) : f(\mathbf{i}, \mathbf{j}_1) \wedge g(\mathbf{i}, \mathbf{j}_2) \} & S_1(\mathbf{i}_1) = S_2(\mathbf{i}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

Application 5: Representing Dynamic Conditions

```
N1: n = f();  
    for (int k = 0; k < 100; ++k) {  
M:        m = g();  
        for (int i = 0; i < m; ++i)  
            for (int j = 0; j < n; ++j)  
A:                a[j][i] = g();  
N2:        n = f();  
    }
```

What is iteration domain (restricted to A statement)?

Application 5: Representing Dynamic Conditions

```
N1: n = f();  
    for (int k = 0; k < 100; ++k) {  
M:        m = g();  
        for (int i = 0; i < m; ++i)  
            for (int j = 0; j < n; ++j)  
A:                a[j][i] = g();  
N2:        n = f();  
    }
```

What is iteration domain (restricted to A statement)?

$\{ A(k, i, j) : 0 \leq k < 100 \wedge 0 \leq i < m \wedge 0 \leq j < n \}$?

Application 5: Representing Dynamic Conditions

```
N1: n = f();  
    for (int k = 0; k < 100; ++k) {  
M:     m = g();  
        for (int i = 0; i < m; ++i)  
            for (int j = 0; j < n; ++j)  
A:                a[j][i] = g();  
N2:    n = f();  
}
```

What is iteration domain (restricted to A statement)?

$\{ A(k, i, j) : 0 \leq k < 100 \wedge 0 \leq i < m \wedge 0 \leq j < n \}$?

⇒ no, m and n cannot be treated as symbolic constants
(they are modified inside k-loop)

Application 5: Representing Dynamic Conditions

```
N1: n = f();
    for (int k = 0; k < 100; ++k) {
M:     m = g();
        for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j)
A:                 a[j][i] = g();
N2:         n = f();
    }
```

What is iteration domain (restricted to A statement)?

$\{A(k, i, j) : 0 \leq k < 100 \wedge 0 \leq i < m \wedge 0 \leq j < n\}$?

⇒ no, m and n cannot be treated as symbolic constants
(they are modified inside k-loop)

$\{A(k, i, j) : 0 \leq k < 100 \wedge 0 \leq i < \text{valueOf_m}(k) \wedge 0 \leq j < \text{valueOf_n}(k)\}$?

Application 5: Representing Dynamic Conditions

```
N1: n = f();
    for (int k = 0; k < 100; ++k) {
M:     m = g();
        for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j)
A:                 a[j][i] = g();
N2:         n = f();
    }
```

What is iteration domain (restricted to A statement)?

$\{A(k, i, j) : 0 \leq k < 100 \wedge 0 \leq i < m \wedge 0 \leq j < n\}$?

⇒ no, m and n cannot be treated as symbolic constants
(they are modified inside k-loop)

$\{A(k, i, j) : 0 \leq k < 100 \wedge 0 \leq i < \text{valueOf_m}(k) \wedge 0 \leq j < \text{valueOf_n}(k)\}$?

⇒ requires uninterpreted functions (of arity > 0)

Application 5: Representing Dynamic Conditions

```
N1: n = f();  
    for (int k = 0; k < 100; ++k) {  
M:     m = g();  
        for (int i = 0; i < m; ++i)  
            for (int j = 0; j < n; ++j)  
A:                a[j][i] = g();  
N2:    n = f();  
}
```

What is iteration domain (restricted to A statement)?

$\{A(k, i, j) : 0 \leq k < 100 \wedge 0 \leq i < m \wedge 0 \leq j < n\}$?

⇒ no, m and n cannot be treated as symbolic constants
(they are modified inside k-loop)

$\{A(k, i, j) : 0 \leq k < 100 \wedge 0 \leq i < \text{valueOf_m}(k) \wedge 0 \leq j < \text{valueOf_n}(k)\}$?

⇒ requires uninterpreted functions (of arity > 0)

Alternative: use overapproximation of iteration domain and keep track of which elements are executed

Application 5: Representing Dynamic Conditions

```

N1: n = f();
    for (int k = 0; k < 100; ++k) {
M:     m = g();
        for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j)
A:                 a[j][i] = g();
N2:         n = f();
    }

```

- Iteration domain: $\{ A(k, i, j) : 0 \leq k < 100 \wedge 0 \leq i \wedge 0 \leq j \}$
- Filter:

► Filter access relations: **reader** \rightarrow (**writer** \rightarrow array element)

$$\star F_1^A = \{ A(k, i, j) \rightarrow (M(k) \rightarrow m()) \}$$

$$\star F_2^A = \{ A(0, i, j) \rightarrow (N1() \rightarrow n()); A(k, i, j) \rightarrow (N2(k - 1) \rightarrow n()): k \geq 1 \}$$

► Filter value relation:

$$V^A = \{ A(k, i, j) \rightarrow (m, n) : 0 \leq k \leq 99 \wedge 0 \leq i < m \wedge 0 \leq j < n \}$$

Iteration domain element is executed iff values written by corresponding write accesses (through filter access relations) satisfy filter value relation

Application 5: Representing Dynamic Conditions

```

N1: n = f();
    for (int k = 0; k < 100; ++k) {
M:     m = g();
        for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j)
A:                 a[j][i] = g();
N2:         n = f();
    }

```

- Iteration domain: $\{ A(k, i, j) : 0 \leq k < 100 \wedge 0 \leq i \wedge 0 \leq j \}$
- Filter:

► Filter access relations: **reader** \rightarrow (**writer** \rightarrow array element)

$$\star F_1^A = \{ A(k, i, j) \rightarrow (M(k) \rightarrow m()) \}$$

$$\star F_2^A = \{ A(0, i, j) \rightarrow (N1() \rightarrow n()); A(k, i, j) \rightarrow (N2(k - 1) \rightarrow n()): k \geq 1 \}$$

► Filter value relation:

$$V^A = \{ A(k, i, j) \rightarrow (m, n) : 0 \leq k \leq 99 \wedge 0 \leq i < m \wedge 0 \leq j < n \}$$

Iteration domain element is executed iff values written by corresponding write accesses (through filter access relations) satisfy filter value relation

Parametric Array Dataflow Analysis

```
while (1) { potential source
```

```
N:   n = f();  
      a = g();  
      if (n < 100)  
H:          a = h();  
          if (n > 200)  
T:          t(a);  
}  
               sink
```

Is there any dataflow between **potential source** and **sink** at inner level?

Parametric Array Dataflow Analysis

```
while (1) { potential source
```

```
N:   n = f();  
      a = g();  
      if (n < 100)  
H:          a = h();  
          if (n > 200)  
T:            t(a);  
}  
                                sink
```

$$I = \{ H(i) : i \geq 0; T(i) : i \geq 0 \}$$

$$F^H = \{ H(i) \rightarrow (N(i) \rightarrow n()) \}$$

$$V^H = \{ H(i) \rightarrow (n) : i \geq 0 \wedge n < 100 \}$$

$$F^T = \{ T(i) \rightarrow (N(i) \rightarrow n()) \}$$

$$V^T = \{ T(i) \rightarrow (n) : i \geq 0 \wedge n > 200 \}$$

Is there any dataflow between **potential source** and **sink** at inner level?

Parametric Array Dataflow Analysis

```
while (1) { potential source
```

```
N:   n = f();  
      a = g();  
      if (n < 100)  
H:          a = h();  
          if (n > 200)  
T:            t(a);  
}  
                                sink
```

$$I = \{ H(i) : i \geq 0; T(i) : i \geq 0 \}$$

$$F^H = \{ H(i) \rightarrow (N(i) \rightarrow n()) \}$$

$$V^H = \{ H(i) \rightarrow (n) : i \geq 0 \wedge n < 100 \}$$

$$F^T = \{ T(i) \rightarrow (N(i) \rightarrow n()) \}$$

$$V^T = \{ T(i) \rightarrow (n) : i \geq 0 \wedge n > 200 \}$$

Is there any dataflow between **potential source** and **sink** at **inner level**?

- $M = \{ T(i) \rightarrow H(i) \}$

Parametric Array Dataflow Analysis

```
while (1) { potential source
```

```
N:   n = f();  
      a = g();  
      if (n < 100)  
H:          a = h();  
          if (n > 200)  
T:            t(a);  
}  
                                sink
```

$$I = \{ H(i) : i \geq 0; T(i) : i \geq 0 \}$$

$$F^H = \{ H(i) \rightarrow (N(i) \rightarrow n()) \}$$

$$V^H = \{ H(i) \rightarrow (n) : i \geq 0 \wedge n < 100 \}$$

$$F^T = \{ T(i) \rightarrow (N(i) \rightarrow n()) \}$$

$$V^T = \{ T(i) \rightarrow (n) : i \geq 0 \wedge n > 200 \}$$

Is there any dataflow between **potential source** and **sink** at **inner level**?

- $M = \{ T(i) \rightarrow H(i) \}$
- $F^H \circ M \subseteq F^T$

\Rightarrow filter elements accessed by any **potential source** instance associated to **sink instance** forms subset of filter elements accessed by **sink** instance

Parametric Array Dataflow Analysis

`while (1) { potential source`

```
N:   n = f();
      a = g();
      if (n < 100)
H:      a = h();
      if (n > 200)
T:      t(a);
}
               sink
```

$$I = \{ H(i) : i \geq 0; T(i) : i \geq 0 \}$$

$$F^H = \{ H(i) \rightarrow (N(i) \rightarrow n()) \}$$

$$V^H = \{ H(i) \rightarrow (n) : i \geq 0 \wedge n < 100 \}$$

$$F^T = \{ T(i) \rightarrow (N(i) \rightarrow n()) \}$$

$$V^T = \{ T(i) \rightarrow (n) : i \geq 0 \wedge n > 200 \}$$

Is there any dataflow between **potential source** and **sink** at **inner level**?

- $M = \{ T(i) \rightarrow H(i) \}$
- $F^H \circ M \subseteq F^T$

- ⇒ filter elements accessed by any **potential source** instance associated to **sink instance** forms subset of filter elements accessed by **sink** instance
- ⇒ constraints on filter values at sink also apply at corresponding potential source: $V^T \circ M^{-1} = \{ H(i) \rightarrow (n) : i \geq 0 \wedge n > 200 \}$

Parametric Array Dataflow Analysis

```
while (1) { potential source
```

```
N:   n = f();  
      a = g();  
      if (n < 100)  
H:          a = h();  
          if (n > 200)  
T:            t(a);  
}  
                                sink
```

$$I = \{ H(i) : i \geq 0; T(i) : i \geq 0 \}$$

$$F^H = \{ H(i) \rightarrow (N(i) \rightarrow n()) \}$$

$$V^H = \{ H(i) \rightarrow (n) : i \geq 0 \wedge n < 100 \}$$

$$F^T = \{ T(i) \rightarrow (N(i) \rightarrow n()) \}$$

$$V^T = \{ T(i) \rightarrow (n) : i \geq 0 \wedge n > 200 \}$$

Is there any dataflow between **potential source** and **sink** at **inner level**?

- $M = \{ T(i) \rightarrow H(i) \}$

- $F^H \circ M \subseteq F^T$

\Rightarrow filter elements accessed by any **potential source** instance associated to **sink instance** forms subset of filter elements accessed by **sink** instance

\Rightarrow constraints on filter values at sink also apply at corresponding potential source: $V^T \circ M^{-1} = \{ H(i) \rightarrow (n) : i \geq 0 \wedge n > 200 \}$

- $(V^T \circ M^{-1}) \cap V^H = \emptyset$

\Rightarrow there can be no dataflow at inner level

Weighted Counting

$$\textcolor{red}{G} = \textcolor{blue}{F} \circ R$$

with $\textcolor{blue}{F}$ a piecewise quasi polynomial and $\textcolor{brown}{R}$ a Presburger relation
is a piecewise quasi polynomial $\textcolor{red}{G}$ such that

$$\textcolor{red}{G}(\mathbf{i}) = \sum_{\mathbf{j}: \textcolor{brown}{R}(\mathbf{i}, \mathbf{j})} \textcolor{blue}{F}(\mathbf{j})$$

Weighted Counting

$$\textcolor{red}{G} = \textcolor{blue}{F} \circ \textcolor{brown}{R} = \left\{ (x, y) \rightarrow \frac{x^2 + y^2}{4} : 1 \leq x, y \leq 2 \right\} \circ \{ (x) \rightarrow (x, y) \}$$

with $\textcolor{blue}{F}$ a piecewise quasi polynomial and $\textcolor{brown}{R}$ a Presburger relation
is a piecewise quasi polynomial $\textcolor{red}{G}$ such that

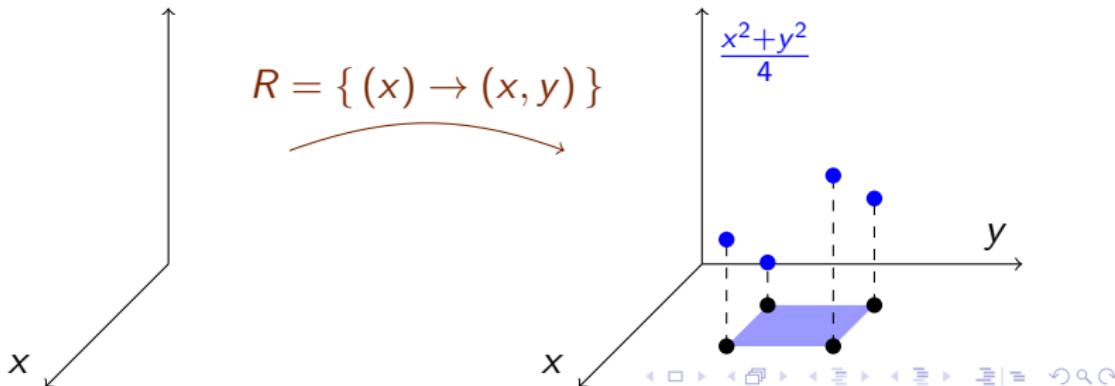
$$\textcolor{red}{G}(\mathbf{i}) = \sum_{\mathbf{j}: \textcolor{brown}{R}(\mathbf{i}, \mathbf{j})} \textcolor{blue}{F}(\mathbf{j})$$

Weighted Counting

$$G = F \circ R = \left\{ (x, y) \rightarrow \frac{x^2 + y^2}{4} : 1 \leq x, y \leq 2 \right\} \circ \{ (x) \rightarrow (x, y) \}$$

with F a piecewise quasi polynomial and R a Presburger relation
is a piecewise quasi polynomial G such that

$$G(i) = \sum_{j: R(i,j)} F(j)$$

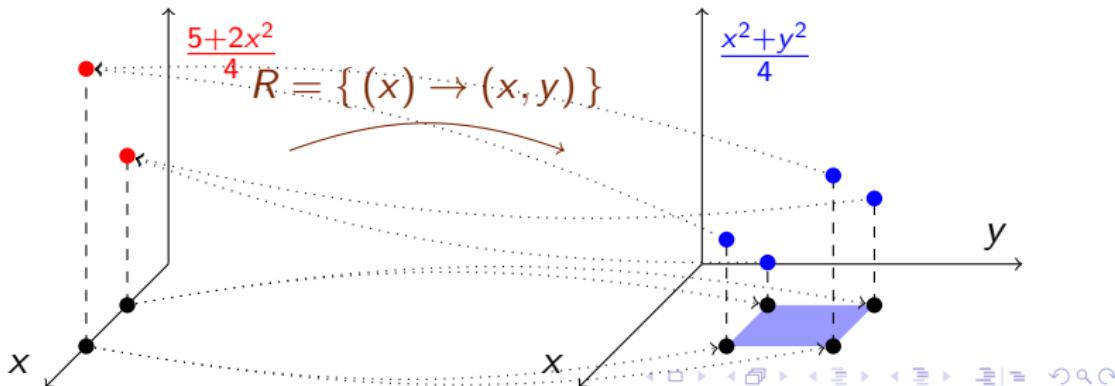


Weighted Counting

$$\begin{aligned} G &= F \circ R = \left\{ (x, y) \rightarrow \frac{x^2 + y^2}{4} : 1 \leq x, y \leq 2 \right\} \circ \{ (x) \rightarrow (x, y) \} \\ &= \left\{ (x) \rightarrow \frac{5 + 2x^2}{2} : 1 \leq x \leq 2 \right\} \end{aligned}$$

with F a piecewise quasi polynomial and R a Presburger relation
is a piecewise quasi polynomial G such that

$$G(\mathbf{i}) = \sum_{\mathbf{j}: R(\mathbf{i}, \mathbf{j})} F(\mathbf{j})$$



Compositions/Application with Piecewise (Folds of) Quasi polynomials

$$F \circ R$$

- $R: D_1 \rightarrow D_2$ is a Presburger relation
- $F: D_2 \rightarrow \mathbb{Q}$ may be
 - ▶ piecewise quasi polynomial
(result of counting problem)
 - ⇒ take sum over $(\text{ran } R) \cap (\text{dom } F)$
 - ▶ piecewise fold of quasi polynomials
(result of upper bound computation)
 - ⇒ compute bound over $(\text{ran } R) \cap (\text{dom } F)$
- $(F \circ R): D_1 \rightarrow \mathbb{Q}$ of same type as F

if R is single-valued, then sum/bound is computed over a single point

Compositions/Application with Piecewise (Folds of) Quasi polynomials

$$F \circ R \quad \text{or} \quad F(S)$$

- $R: D_1 \rightarrow D_2$ is a Presburger relation
- $S \subseteq D_2$ is a Presburger set
- $F: D_2 \rightarrow \mathbb{Q}$ may be

- ▶ piecewise quasi polynomial
(result of counting problem)

⇒ take sum over $(\text{ran } R) \cap (\text{dom } F)$ or $S \cap (\text{dom } F)$

- ▶ piecewise fold of quasi polynomials
(result of upper bound computation)

⇒ compute bound over $(\text{ran } R) \cap (\text{dom } F)$ or $S \cap (\text{dom } F)$

- $(F \circ R): D_1 \rightarrow \mathbb{Q}$ of same type as F
- $F(S): \mathbb{Q}$ of same type as F

if R is single-valued, then sum/bound is computed over a single point

Example: Total Memory Allocation

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        p[i][j] = malloc(i * j + i - N + 1);
/* ... */
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        free(p[i][j]);
```

How much memory allocated in total?

Example: Total Memory Allocation

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        p[i][j] = malloc(i * j + i - N + 1);
/* ... */
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        free(p[i][j]);
```

How much memory allocated in total?

$$F = \{(i, j) \rightarrow ij + i - N + 1\}$$

$$I = \{(i, j) : 0 \leq i < N \wedge i \leq j < N\}$$

$$F(I) = \left\{ \frac{5}{12}N - \frac{1}{8}N^2 - \frac{5}{12}N^3 + \frac{1}{8}N^4 : N \geq 1 \right\}$$

Pointer Conversion

```
p = a;  
for (i = 0; i < N; ++i)  
    for (j = i; j < N; ++j) {  
        p += j * ((j-i)/4);  
        *p = hard_work(i, j);  
    }
```

Can we parallelize this code?

Pointer Conversion

```
p = a;  
for (i = 0; i < N; ++i)  
    for (j = i; j < N; ++j) {  
        p += j * ((j-i)/4);  
        *p = hard_work(i, j);  
    }
```

Can we parallelize this code?

⇒ No, (false) dependency through p

⇒ Compute closed formula for p

$$p = a + \sum_{\substack{(i', j') \in I \\ (i', j') \preccurlyeq (i, j)}} j' \left\lfloor \frac{j' - i'}{4} \right\rfloor$$

$$I = \{ (i, j) : 0 \leq i < N \wedge i \leq j < N \}$$

$$F = \left\{ (i, j) \rightarrow j \left\lfloor \frac{j - i}{4} \right\rfloor \right\}$$

$$F \circ (I \succcurlyeq I) = \dots$$

Application 6: Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c); /*S1*/
        B[] m2Arr = m2(2*m-c); /*S2*/
    }
}

void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A(); /*S3*/
        B[] dummyArr = m2(i); /*S4*/
    }
}

B[] m2(int n) {
    B[] arrB = new B[n]; /*S5*/
    for (j = 1; j <= n; j++)
        B b = new B(); /*S6*/
    return arrB;
}
```

Application 6: Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c);                                /*S1*/
        B[] m2Arr = m2(2*m-c); /*S2*/
    }
}

void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                      /*S3*/
        B[] dummyArr = m2(i); /*S4*/
    }
}

B[] m2(int n) {
    B[] arrB = new B[n];      /*S5*/
    for (j = 1; j <= n; j++)
        B b = new B();          /*S6*/
    return arrB;
}
```

$I = \{m0(m) \rightarrow S1(c) : 0 \leq c < m;$
 $m0(m) \rightarrow S2(c) : 0 \leq c < m;$
 $m1(k) \rightarrow S3(i) : 1 \leq i \leq k;$
 $m1(k) \rightarrow S4(i) : 1 \leq i \leq k;$
 $m2(n) \rightarrow S5();$
 $m2(n) \rightarrow S6(j) : 1 \leq j \leq n\}$

Dynamic Memory Requirement Estimation

How much (scoped) memory is needed?

⇒ compute for each method

ret_m size of memory returned by m

cap_m size of memory “captured” (not returned) by m

memRq_m total memory requirements of m

$$\text{ret}_m + \text{cap}_m = \sum_{p \text{ called by } m} \text{ret}_p$$

$$\text{memRq}_m = \text{cap}_m + \max_{p \text{ called by } m} \text{memRq}_p$$

Dynamic Memory Requirement Estimation

How much (scoped) memory is needed?

⇒ compute for each method

ret_m size of memory returned by m

cap_m size of memory “captured” (not returned) by m

memRq_m total memory requirements of m

$$\text{ret}_m + \text{cap}_m = \sum_{\substack{\text{p called by } m}} \text{ret}_p$$

$$\text{memRq}_m = \text{cap}_m + \max_{\substack{\text{p called by } m}} \text{memRq}_p$$

⇒ summarize over iteration domain, i.e., compose with $M = \underline{\text{dom }} I^{-1}$

$$M = \{ m0(m) \rightarrow (\underline{m0(m)} \rightarrow S1(c)) : 0 \leq c < m; \}$$

$$m0(m) \rightarrow (\underline{m0(m)} \rightarrow S2(c)) : 0 \leq c < m;$$

$$m1(k) \rightarrow (\underline{m1(k)} \rightarrow S3(i)) : 1 \leq i \leq k;$$

$$m1(k) \rightarrow (\underline{m1(k)} \rightarrow S4(i)) : 1 \leq i \leq k;$$

$$m2(n) \rightarrow (\underline{m2(n)} \rightarrow S5()); m2(n) \rightarrow (\underline{m2(n)} \rightarrow S6(j)) : 1 \leq j \leq n \}$$

Dynamic Memory Requirement Estimation

$$\text{ret}_m + \text{cap}_m = \sum_{p \text{ called by } m} \text{ret}_p$$

$$\text{memRq}_m = \text{cap}_m + \max_{p \text{ called by } m} \text{memRq}_p$$

Dynamic Memory Requirement Estimation

$$\text{ret}_m + \text{cap}_m = \sum_{p \text{ called by } m} \text{ret}_p$$

$$\text{memRq}_m = \text{cap}_m + \max_{p \text{ called by } m} \text{memRq}_p$$

```
B [] m2(int n) {
    B [] arrB = new B [n];           /*S5*/
    for (j = 1; j <= n; j++)
        B b = new B();                /*S6*/
    return arrB;
}
```

Dynamic Memory Requirement Estimation

$$\text{ret}_m + \text{cap}_m = \sum_{p \text{ called by } m} \text{ret}_p$$

$$\text{memRq}_m = \text{cap}_m + \max_{p \text{ called by } m} \text{memRq}_p$$

```
B [] m2(int n) {
    B [] arrB = new B [n]; /*S5*/
    for (j = 1; j <= n; j++)
        B b = new B(); /*S6*/
    return arrB;
}
```

$$\text{ret}_{m2} = \{ (m2(n) \rightarrow S5()) \rightarrow n : n \geq 0 \} \circ M$$

$$\text{cap}_{m2} = \{ (m2(n) \rightarrow S6(j)) \rightarrow 1 \} \circ M$$

$$\text{memRq}_{m2} = \text{cap}_{m2} + \{ m2(n) \rightarrow \max(0) \}$$

Dynamic Memory Requirement Estimation

$$\text{ret}_m + \text{cap}_m = \sum_{p \text{ called by } m} \text{ret}_p$$

$$\text{memRq}_m = \text{cap}_m + \max_{p \text{ called by } m} \text{memRq}_p$$

```
B [] m2(int n) {
    B [] arrB = new B [n]; /*S5*/
    for (j = 1; j <= n; j++)
        B b = new B(); /*S6*/
    return arrB;
}
```

$$\text{ret}_{m2} = \{ (m2(n) \rightarrow S5()) \rightarrow n : n \geq 0 \} \circ M = \{ m2(n) \rightarrow n : n \geq 0 \}$$

$$\text{cap}_{m2} = \{ (m2(n) \rightarrow S6(j)) \rightarrow 1 \} \circ M = \{ m2(n) \rightarrow n : n \geq 1 \}$$

$$\text{memRq}_{m2} = \text{cap}_{m2} + \{ m2(n) \rightarrow \max(0) \} = \{ m2(n) \rightarrow \max(n) : n \geq 1 \}$$

Dynamic Memory Requirement Estimation

```
void m1(int k) {  
    for (i = 1; i <= k; i++) {  
        A a = new A(); /* S3 */  
        B [] dummyArr = m2(i); /* S4 */  
    }  
}
```

$$\text{cap}_{m1}(k) = \sum_{1 \leq i \leq k} (1 + \text{ret}_{m2}(i))$$

ret_{m2} is a function of the arguments of $m2$

We want to use it as a function of the arguments and local variables of $m1$

Dynamic Memory Requirement Estimation

```
void m1(int k) {  
    for (i = 1; i <= k; i++) {  
        A a = new A(); /* S3 */  
        B[] dummyArr = m2(i); /* S4 */  
    }  
}
```

$$\text{cap}_{m1}(k) = \sum_{1 \leq i \leq k} (1 + \text{ret}_{m2}(i))$$

ret_{m2} is a function of the arguments of $m2$

We want to use it as a function of the arguments and local variables of $m1$

⇒ define parameter binding

Application 6: Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c);                                /*S1*/
        B[] m2Arr = m2(2*m-c); /*S2*/
    }
}

void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                      /*S3*/
        B[] dummyArr = m2(i); /*S4*/
    }
}

B[] m2(int n) {
    B[] arrB = new B[n];      /*S5*/
    for (j = 1; j <= n; j++)
        B b = new B();          /*S6*/
    return arrB;
}
```

$I = \{m0(m) \rightarrow S1(c) : 0 \leq c < m;$
 $m0(m) \rightarrow S2(c) : 0 \leq c < m;$
 $m1(k) \rightarrow S3(i) : 1 \leq i \leq k;$
 $m1(k) \rightarrow S4(i) : 1 \leq i \leq k;$
 $m2(n) \rightarrow S5();$
 $m2(n) \rightarrow S6(j) : 1 \leq j \leq n\}$

Application 6: Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c); /*S1*/
        B[] m2Arr = m2(2*m-c); /*S2*/
    }
}

void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A(); /*S3*/
        B[] dummyArr = m2(i); /*S4*/
    }
}

B[] m2(int n) {
    B[] arrB = new B[n]; /*S5*/
    for (j = 1; j <= n; j++)
        B b = new B(); /*S6*/
    return arrB;
}
```

$I = \{ m0(m) \rightarrow S1(c) : 0 \leq c < m; m0(m) \rightarrow S2(c) : 0 \leq c < m; m1(k) \rightarrow S3(i) : 1 \leq i \leq k; m1(k) \rightarrow S4(i) : 1 \leq i \leq k; m2(n) \rightarrow S5(); m2(n) \rightarrow S6(j) : 1 \leq j \leq n \}$

$B^{m0} = \{ (m0(m) \rightarrow S1(c)) \rightarrow m1(c); (m0(m) \rightarrow S2(c)) \rightarrow m2(2m - c) \}$
 $B^{m1} = \{ (m1(k) \rightarrow S4(i)) \rightarrow m2(i) \}$

Dynamic Memory Requirement Estimation

```
void m1(int k) {  
    for (i = 1; i <= k; i++) {  
        A a = new A(); /* S3 */  
        B[] dummyArr = m2(i); /* S4 */  
    }  
}
```

$$\text{cap}_{m1}(k) = \sum_{1 \leq i \leq k} (1 + \text{ret}_{m2}(i))$$

ret_{m2} is a function of the arguments of $m2$

We want to use it as a function of the arguments and local variables of $m1$

⇒ define parameter binding

Dynamic Memory Requirement Estimation

```
void m1(int k) {  
    for (i = 1; i <= k; i++) {  
        A a = new A(); /* S3 */  
        B[] dummyArr = m2(i); /* S4 */  
    }  
}
```

$$\text{cap}_{\text{m1}}(k) = \sum_{1 \leq i \leq k} (1 + \text{ret}_{\text{m2}}(i))$$

$$\text{ret}_{\text{m1}} = \{ \text{m1}(k) \rightarrow 0 \}$$

$$\text{cap}_{\text{m1}} = (\{ (\text{m1}(k) \rightarrow \text{S3}(i)) \rightarrow 1 \} + \text{ret}_{\text{m2}} \circ B^{\text{m1}}) \circ M$$

$$\text{memRq}_{\text{m1}} = \text{cap}_{\text{m1}} + (\text{memRq}_{\text{m2}} \circ B^{\text{m1}} \circ M)$$

Dynamic Memory Requirement Estimation

```
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A(); /* S3 */
        B[] dummyArr = m2(i); /* S4 */
    }
}
```

$$\text{cap}_{\text{m1}}(k) = \sum_{1 \leq i \leq k} (1 + \text{ret}_{\text{m2}}(i))$$

$$\text{ret}_{\text{m1}} = \{ \text{m1}(k) \rightarrow 0 \}$$

$$\begin{aligned} \text{cap}_{\text{m1}} &= (\{ (\text{m1}(k) \rightarrow \text{S3}(i)) \rightarrow 1 \} + \text{ret}_{\text{m2}} \circ B^{\text{m1}}) \circ M \\ &= \left\{ \text{m1}(k) \rightarrow \frac{3}{2}k + \frac{1}{2}k^2 : k \geq 1 \right\} \end{aligned}$$

$$\begin{aligned} \text{memRq}_{\text{m1}} &= \text{cap}_{\text{m1}} + (\text{memRq}_{\text{m2}} \circ B^{\text{m1}} \circ M) \\ &= \left\{ \text{m1}(k) \rightarrow \max \left(\frac{5}{2}k + \frac{1}{2}k^2 \right) : k \geq 1 \right\} \end{aligned}$$

Dynamic Memory Requirement Estimation

```
void m0(int m) {  
    for (c = 0; c < m; c++) {  
        m1(c); /* S1 */  
        B [] m2Arr = m2(2 * m - c); /* S2 */  
    }  
}
```

$$B^{m^0} = \{ (m0(k) \rightarrow S1(c)) \rightarrow m1(c); (m0(k) \rightarrow S2(c)) \rightarrow m2(2m - c) \}$$

$$\text{ret}_{m0} = \{ m0(m) \rightarrow 0 \}$$

$$\text{cap}_{m0} = (\text{ret}_{m1} + \text{ret}_{m2}) \circ B^{m^0} \circ M$$

$$\text{memRq}_{m0} = \text{cap}_{m0} + ((\text{memRq}_{m1} + \text{memRq}_{m2}) \circ B^{m^0} \circ M)$$

Dynamic Memory Requirement Estimation

```

void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c);                                /* S1 */
        B [] m2Arr = m2(2 * m - c); /* S2 */
    }
}

```

$$B^{m^0} = \{ (m0(k) \rightarrow S1(c)) \rightarrow m1(c); (m0(k) \rightarrow S2(c)) \rightarrow m2(2m - c) \}$$

$$\text{ret}_{m0} = \{ m0(m) \rightarrow 0 \}$$

$$\text{cap}_{m0} = (\text{ret}_{m1} + \text{ret}_{m2}) \circ B^{m^0} \circ M$$

$$= \left\{ m0(m) \rightarrow \frac{1}{2}m + \frac{3}{2}m^2 : m \geq 1 \right\}$$

$$\text{memRq}_{m0} = \text{cap}_{m0} + ((\text{memRq}_{m1} + \text{memRq}_{m2}) \circ B^{m^0} \circ M)$$

$$= \left\{ m0(m) \rightarrow \max \left(-2 + 2m + 2m^2, \frac{5}{2}m + \frac{3}{2}m^2 \right) : m \geq 1 \right\}$$

Positive Powers

Definition (Power of a Relation)

Let R be a Presburger relation and k a positive integer, then power k of relation R is defined as

$$R^k := \begin{cases} R & \text{if } k = 1 \\ R \circ R^{k-1} & \text{if } k \geq 2. \end{cases}$$

Positive Powers

Definition (Power of a Relation)

Let R be a Presburger relation and k a positive integer, then power k of relation R is defined as

$$R^k := \begin{cases} R & \text{if } k = 1 \\ R \circ R^{k-1} & \text{if } k \geq 2. \end{cases}$$

Example

$$R = \{ (x) \rightarrow (x + 1) \}$$

$$R^k = \{ (x) \rightarrow (x + k) : k \geq 1 \}$$

Transitive Closures

Definition (Transitive Closure of a Relation)

Let R be a Presburger relation, then the transitive closure R^+ of R is the union of all positive powers of R ,

$$R^+ := \bigcup_{k \geq 1} R^k.$$

Transitive Closures

Definition (Transitive Closure of a Relation)

Let R be a Presburger relation, then the transitive closure R^+ of R is the union of all positive powers of R ,

$$R^+ := \bigcup_{k \geq 1} R^k.$$

Example

$$R = \{ (x) \rightarrow (x + 1) \}$$

$$R^k = \{ (x) \rightarrow (x + k) : k \geq 1 \}$$

$$R^+ = \{ (x) \rightarrow (y) : \exists k \geq 1 : y = x + k \} = \{ (x) \rightarrow (y) : y \geq x + 1 \}$$

Transitive Closures

Definition (Transitive Closure of a Relation)

Let R be a Presburger relation, then the transitive closure R^+ of R is the union of all positive powers of R ,

$$R^+ := \bigcup_{k \geq 1} R^k.$$

Example

$$R = \{ (x) \rightarrow (x + 1) \}$$

$$R^k = \{ (x) \rightarrow (x + k) : k \geq 1 \}$$

$$R^+ = \{ (x) \rightarrow (y) : \exists k \geq 1 : y = x + k \} = \{ (x) \rightarrow (y) : y \geq x + 1 \}$$

Definition (Transitive Closure of a Relation, Alternative)

Inductive definition:

$$R^+ := R \cup (R \circ R^+)$$

Transitive Closures — Approximation

Fact

Given a Presburger relation R , the power R^k (with k a parameter) and the transitive closure R^+ may not be Presburger relations.

Example

$$R = \{ (x) \rightarrow (2x) \}$$

$$R^k = \{ (x) \rightarrow (2^k x) \}$$

Transitive Closures — Approximation

Fact

Given a Presburger relation R , the power R^k (with k a parameter) and the transitive closure R^+ may not be Presburger relations.

Example

$$R = \{ (x) \rightarrow (2x) \}$$

$$R^k = \{ (x) \rightarrow (2^k x) \}$$

⇒ need for approximation

- ▶ overapproximation $\overline{R^+}$
- ▶ underapproximation R^\pm

Transitive Closures — Approximation

Fact

Given a Presburger relation R , the power R^k (with k a parameter) and the transitive closure R^+ may not be Presburger relations.

Example

$$R = \{ (x) \rightarrow (2x) \}$$

$$R^k = \{ (x) \rightarrow (2^k x) \}$$

⇒ need for approximation

- ▶ overapproximation $R^{\overline{+}}$
- ▶ underapproximation $R^{\underline{-}}$

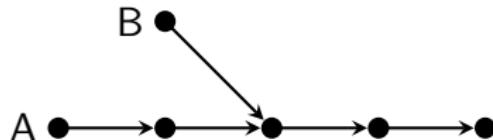
Note

Do not use transitive closures if there is an alternative.

Transitive Closures — Graph Example

Given a graph (represented as a Presburger relation)

$$M = \{ A(i) \rightarrow A(i + 1) : 0 \leq i \leq 3; B() \rightarrow A(2) \}$$

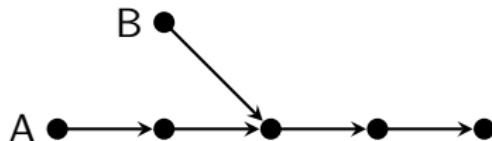


What is the transitive closure?

Transitive Closures — Graph Example

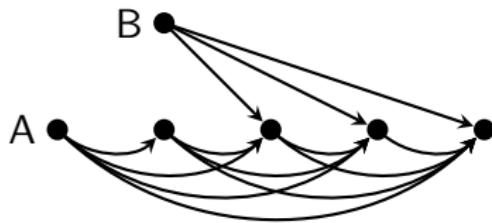
Given a graph (represented as a Presburger relation)

$$M = \{ A(i) \rightarrow A(i+1) : 0 \leq i \leq 3; B() \rightarrow A(2) \}$$



What is the transitive closure?

$$\Rightarrow M^+ = \{ A(i) \rightarrow A(i') : 0 \leq i < i' \leq 4; B() \rightarrow A(i) : 2 \leq i \leq 4 \}$$



Application 7: Reachability Analysis

```
double x[2][10];
int old = 0, new = 1, i, t;
for (t = 0; t<1000; t++) {
    for (i = 0; i<10; i++)
        x[new][i] = g(x[old][i]);
    new = (new+1) %2; old = (old+1) %2;
}
```

- State: “overapproximation of values of integer variables”

- ▶ initial state

$$S_0 = \{ \text{st}(old, new, i, t) : old = 0 \wedge new = 1 \wedge t = 0 \}$$

- ▶ transformer for outer for loop

$$T = \{ \text{st}(old, new, i, t) \rightarrow \text{st}((old + 1) \bmod 2, (new + 1) \bmod 2, 9, t + 1) \}$$

Application 7: Reachability Analysis

```
double x[2][10];
int old = 0, new = 1, i, t;
for (t = 0; t<1000; t++) {
    for (i = 0; i<10; i++)
        x[new][i] = g(x[old][i]);
    new = (new+1) %2; old = (old+1) %2;
}
```

- State: “overapproximation of values of integer variables”

- ▶ initial state

$$S_0 = \{ \text{st}(old, new, i, t) : old = 0 \wedge new = 1 \wedge t = 0 \}$$

- ▶ transformer for outer for loop

$$T = \{ \text{st}(old, new, i, t) \rightarrow \text{st}((old + 1) \bmod 2, (new + 1) \bmod 2, 9, t + 1) \}$$

- ▶ invariant of outer for loop $I = T^{\top}(S_0) \cup S_0$

$$I = \{ \text{st}(old, 1 - old, 9, t) : 0 \leq old \leq 1 \wedge t \geq 1 \wedge old = t \bmod 2; \\ \text{st}(0, 1, i, 0) \}$$

Application 7: Reachability Analysis/Array Region Analysis

```
double x[2][10];
int old = 0, new = 1, i, t;
for (t = 0; t < 1000; t++) {
    for (i = 0; i < 10; i++)
        x[new][i] = g(x[old][i]);
    new = (new + 1) % 2; old = (old + 1) % 2;
}
```

- State: “overapproximation of values of integer variables”

- ▶ initial state

$$S_0 = \{ \text{st}(old, new, i, t) : old = 0 \wedge new = 1 \wedge t = 0 \}$$

- ▶ transformer for outer for loop

$$T = \{ \text{st}(old, new, i, t) \rightarrow \text{st}((old + 1) \bmod 2, (new + 1) \bmod 2, 9, t + 1) \}$$

- ▶ invariant of outer for loop $I = T^{\top}(S_0) \cup S_0$

$$I = \{ \text{st}(old, 1 - old, 9, t) : 0 \leq old \leq 1 \wedge t \geq 1 \wedge old = t \bmod 2; \\ \text{st}(0, 1, i, 0) \}$$

- Array region: “access relation in terms of state”

$$\{ \text{st}(old, new, i, t) \rightarrow x(new, i) \}$$

Outline

- 1 Introduction
- 2 Representation
- 3 Operations
 - Basic Operations
 - Application 1: Dependence Analysis
 - Application 2: Array Dataflow Analysis
 - Cardinality
 - Application 3: Reuse Distance Computation
 - Application 4: Maximal Number of Live Memory elements
 - Nested Relations
 - Application 5: Representing Dynamic Conditions
 - Weighted Counting
 - Application 6: Dynamic Memory Requirement Estimation
 - Transitive Closures
 - Application 7: Reachability Analysis
- 4 Availability

Availability — Representation

$$\{ A(i) : 0 \leq i \leq n; B(i,j) : \exists \alpha : i = 2\alpha \}$$

- Named (and nested) spaces: `isl`

```
[n] -> { A[i]: 0 <= i <= n; B[i,j]: exists a: i = 2 a }
```

In `omega(+)`:

Availability — Representation

$$\{ A(i) : 0 \leq i \leq n; B(i,j) : \exists \alpha : i = 2\alpha \}$$

- Named (and nested) spaces: `isl`

```
[n] -> { A[i]: 0 <= i <= n; B[i,j]: exists a: i = 2 a }
```

In omega(+):

A padding **B**

~~symbolic n;~~

{ [0, i, 0]: 0 <= i <= n } union { [1, i, j]: exists a: i = 2 a }

Availability — Representation

$$\{ A(i) : 0 \leq i \leq n; B(i,j) : \exists \alpha : i = 2\alpha \}$$

- Named (and nested) spaces: `isl`

```
[n] -> { A[i]: 0 <= i <= n; B[i,j]: exists a: i = 2 a }
```

In `omega(+):`

`A` `padding` `B`

~~symbolic n;~~
~~{ [0, i, 0]: 0 <= i <= n } union { [1, i, j]: exists a: i = 2 a }~~

- Presburger sets and relations: `isl`, `omega(+)`

Availability — Representation

$$\{ A(i) : 0 \leq i \leq n; B(i,j) : \exists \alpha : i = 2\alpha \}$$

- Named (and nested) spaces: `isl`

```
[n] -> { A[i]: 0 <= i <= n; B[i,j]: exists a: i = 2 a }
```

In `omega(+)`:

```

symbolic n;
{ [0, i, 0]: 0 <= i <= n } union { [1, i, j]: exists a: i = 2 a }

```

- Presburger sets and relations: `isl`, `omega(+)`

In PolyLib:

2	4	6				
0	-1	0	0	0	0	n
0	0	0	-1	0	0	
1	0	1	0	0	0	
1	0	-1	0	1	0	
2	7					
0	-1	0	0	0	0	
0	0	-1	0	2	0	

Moreover: PolyLib deals with **rational** sets (polyhedra)

isl/iscc syntax

() (tuple)	[]
+, -	+, -
=, \leq , $<$, \geq , $>$	=, \leq , $<$, \geq , $>$
true	true
false	false
\wedge	and
\vee	or
\neg	not
$\exists v :$	exists v :
$\forall v :$	not exists v : not
\asymp , \prec , \succ , \succcurlyeq	(not available yet; write out explicitly)

Availability — Representation (2)

- Uninterpreted functions: `omega(+)`
 - ⇒ arity can be greater than 0
 - ⇒ not available in `isl` (yet)

Note: support in `omega(+)` for uninterpreted functions is very restrictive

- ▶ arguments need to be prefix of input/output dimensions
 - ⇒ essentially symbolic constants

Availability — Operations

	union	intersection	difference	emptiness	application	domain	range	inverse
PolyLib	✓	✓	in \mathbb{Z}					
PPL	✓	✓			✓			
PIP					lexmin			
omega(+)	✓	✓	✓	✓				✓
isl	✓	✓	✓	✓				✓
barvinok					card			
bernstein								
LattE					card			

incomplete and biased

Availability — Operations (2)

	lexmin	cardinality	bounds on polynomials	weighted counting	transitive closure
PolyLib		✓			
PPL	✓				
PIP	✓				
omega(+)	Presburger				" R^\pm "
isl	✓		✓		R^+
barvinok	partial	✓		✓	
bernstein			✓		
LattE		✓		✓	

incomplete and biased

Part II

Algorithms and Implementation

Outline

5 Tools

6 Internal Representation

7 Weighted Counting

- Introduction
- Nested Sums
- Unweighted Counting
- Derivation
- Local Euler-Maclaurin Formulas
- Laurent Series Expansion
- Interpolation
- Set Operations

Outline

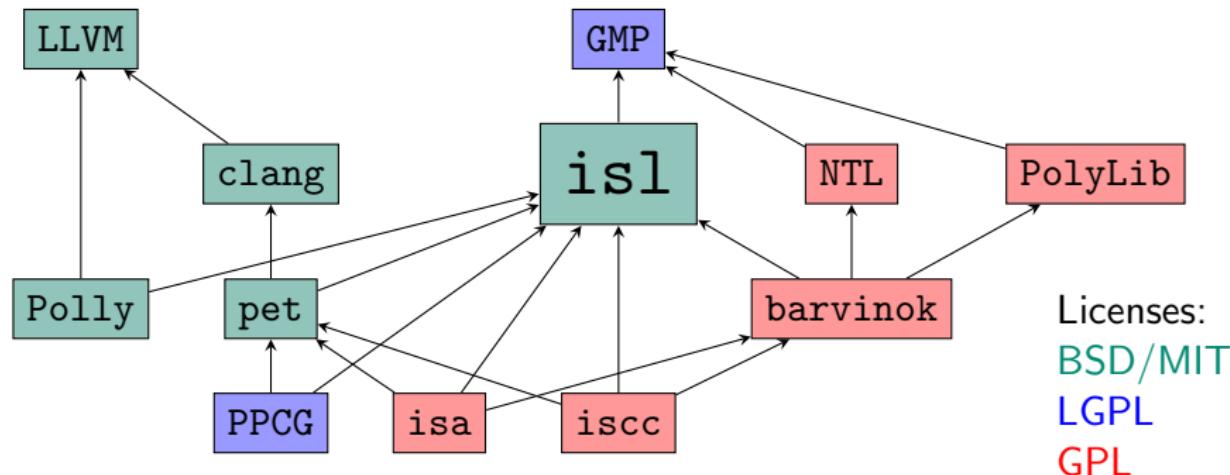
5 Tools

6 Internal Representation

7 Weighted Counting

- Introduction
- Nested Sums
- Unweighted Counting
- Derivation
- Local Euler-Maclaurin Formulas
- Laurent Series Expansion
- Interpolation
- Set Operations

isl and Related Libraries and Tools



isl: manipulates Presburger sets and relations

barvinok: counts elements in Presburger sets and relations

pet: extracts polyhedral model from clang AST

PPCG: Polyhedral Parallel Code Generator

iscc: interactive calculator

isa: prototype tool set including derivation of process networks and equivalence checker

Overview of isl

isl is a thread-safe C library for manipulating

- *Presburger sets and binary relations*
- *piecewise quasi affine expressions*
- *piecewise quasi polynomials*

Supported operations by core library include

- *intersection*
- *union*
- *set difference*
- *integer projection*
- *coalescing*
- *closed convex hull*
- *sampling, scanning*
- *integer affine hull*
- *lexicographic optimization*
- *transitive closure* (approx.)
- *parametric vertex enumeration*
- *bounds on quasi polynomials*

Polyhedral compilation library

- *dataflow analysis*
- *scheduling*
- *AST generation*

Dataflow analysis

- Input:
- access relations
 - schedule relation

- Output:
- dataflow relation
 - uninitialized reads

Dataflow analysis

- Input:
- access relations
 - schedule relation

- Output:
- dataflow relation
 - uninitialized reads

Note

isl performs dataflow analysis **after** access relations and schedule have been extracted as Presburger relations.

It would be better to perform dataflow analysis on the AST
(before or during the extraction).

- more efficient
- more easily extensible

See, e.g., Maslov (1994) [25] or array region analysis

Scheduling

[10, 3]

- Input:
- iteration domain
 - dependence relation(s)

- Output:
- schedule tree

Scheduling

[10, 3]

- Input:
- iteration domain
 - dependence relation(s)

- Output:
- schedule tree

```
A:    for (int i = 0; i < N; ++i)
        a[i] = f(i);
B:    for (int i = 0; i < N; ++i)
        b[i] = a[N-1-i];
C:    for (int i = 0; i < N; ++i)
        c[i] = 0;
```

$$\{ A(i) \rightarrow (0); B(i) \rightarrow (0); C(i) \rightarrow (1) \}$$

$$\{ A(i) \rightarrow (-i); B(i) \rightarrow (1 - N + i) \}$$

$$\{ C(i) \rightarrow (i) \}$$

$$\{ A(i) \rightarrow (0); B(i) \rightarrow (1) \}$$

Scheduling

[10, 3]

- Input:
- iteration domain
 - dependence relation(s)

- Output:
- schedule tree

```
A:    for (int i = 0; i < N; ++i)
      a[i] = f(i);
B:    for (int i = 0; i < N; ++i)
      b[i] = a[N-1-i];
C:    for (int i = 0; i < N; ++i)
      c[i] = 0;
```

$$\{ A(i) \rightarrow (0); B(i) \rightarrow (0); C(i) \rightarrow (1) \}$$

$$\{ A(i) \rightarrow (-i); B(i) \rightarrow (1 - N + i) \}$$

$$\{ C(i) \rightarrow (i) \}$$

$$\{ A(i) \rightarrow (0); B(i) \rightarrow (1) \}$$

More advanced alternatives: Pluto, PoCC, ...

AST Generation

[8, 11]

Input: • schedule relation

Output: • code that visits each domain element according to lexicographic order of corresponding image
⇒ single output space

AST Generation

[8, 11]

Input: • schedule relation

Output: • code that visits each domain element according to lexicographic order of corresponding image
⇒ single output space

$$\{ A(i) \rightarrow (0, -i, 0) : 0 \leq i < N; B(i) \rightarrow (0, 1 - N + i, 1) : 0 \leq i < N; \\ C(i) \rightarrow (1, i, 0) : 0 \leq i < N \}$$

```
{  
    for (int c1 = -N + 1; c1 <= 0; c1 += 1) {  
        A(-c1);  
        B(N + c1 - 1);  
    }  
    for (int c1 = 0; c1 < N; c1 += 1)  
        C(c1);  
}
```

AST Generation

[8, 11]

Input: • schedule relation

Output: • code that visits each domain element according to lexicographic order of corresponding image
⇒ single output space

$$\{ A(i) \rightarrow (0, -i, 0) : 0 \leq i < N; B(i) \rightarrow (0, 1 - N + i, 1) : 0 \leq i < N; \\ C(i) \rightarrow (1, i, 0) : 0 \leq i < N \}$$

```
{  
    for (int c1 = -N + 1; c1 <= 0; c1 += 1) {  
        A(-c1);  
        B(N + c1 - 1);  
    }  
    for (int c1 = 0; c1 < N; c1 += 1)  
        C(c1);  
}
```

Alternatives: CLooG, codegen+

AST Generation

[8, 11]

- Input:
- schedule relation
 - future work: schedule tree

- Output:
- code that visits each domain element according to lexicographic order of corresponding image
⇒ single output space

$$\{ A(i) \rightarrow (0, -i, 0) : 0 \leq i < N; B(i) \rightarrow (0, 1 - N + i, 1) : 0 \leq i < N;$$
$$C(i) \rightarrow (1, i, 0) : 0 \leq i < N \}$$

```
{  
    for (int c1 = -N + 1; c1 <= 0; c1 += 1) {  
        A(-c1);  
        B(N + c1 - 1);  
    }  
    for (int c1 = 0; c1 < N; c1 += 1)  
        C(c1);  
}
```

Alternatives: CLooG, codegen+

PPCG

PPCG (<http://freecode.com/projects/ppcg>)

- Input: C code
- Output: CUDA code for GPGPUs (or OpenMP code for CPUs)

PPCG

PPCG (<http://freecode.com/projects/ppcg>)

- Input: C code
- Output: CUDA code for GPGPUs (or OpenMP code for CPUs)

Steps:

- extract polyhedral model from C code (pet)
- dependence analysis (isl)
- scheduling
 - ▶ expose parallelism and tiling opportunities (isl)
 - ▶ perform tiling (isl)
 - ▶ separate into parts mapped to host, GPU blocks and GPU threads
- memory management
 - ▶ add transfers of data to/from GPU
 - ▶ detect array reference groups
 - ▶ allocate groups to registers and shared memory
- generate AST (isl)

PPCG

PPCG (<http://freecode.com/projects/ppcg>)

- Input: C code
- Output: CUDA code for GPGPUs (or OpenMP code for CPUs)

Steps:

- extract polyhedral model from C code (pet)
- dependence analysis (isl)
- scheduling
 - ▶ expose parallelism and tiling opportunities (isl)
 - ▶ perform tiling (isl)
 - ▶ separate into parts mapped to host, GPU blocks and GPU threads
- memory management
 - ▶ add transfers of data to/from GPU
 - ▶ detect array reference groups
 - ▶ allocate groups to registers and shared memory
- generate AST (isl)

PPCG Example — Input

```
void matmul(int M, int N, int K,
            float A[static const restrict M][K],
            float B[static const restrict K][N],
            float C[static const restrict M][N])
{
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++) {
S1:    C[i][j] = 0;
        for (int k = 0; k < K; k++)
S2:        C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
}
```

Options:

```
--ctx="[M,N,K] -> { : M = N = K = 256 }"
--sizes="{ kernel[i] -> tile[16,16,16];
          kernel[i] -> block[8,16] }"
--pet-autodetect
```

PPCG Example — Output

```
long b0 = blockIdx.y, b1 = blockIdx.x;
long t0 = threadIdx.y, t1 = threadIdx.x;
__shared__ float s_A[16][16];
float p_C[2][1];
__shared__ float s_B[16][16];

for (long g9 = 0; g9 <= 15; g9 += 1) {
    for (long c0 = t0; c0 <= 15; c0 += 8)
        s_B[c0][t1] = B[(16 * g9 + c0) * (256) + 16 * b1 + t1];
    for (long c0 = t0; c0 <= 15; c0 += 8)
        s_A[c0][t1] = A[(16 * b0 + c0) * (256) + t1 + 16 * g9];
    __syncthreads();
    if (g9 == 0) {
        p_C[0][0] = (0);
        p_C[1][0] = (0);
    }
    for (long c3 = 0; c3 <= 15; c3 += 1) {
        p_C[0][0] = (p_C[0][0] + (s_A[t0][c3] * s_B[c3][t1]));
        p_C[1][0] = (p_C[1][0] + (s_A[t0 + 8][c3] * s_B[c3][t1]));
    }
    __syncthreads();
}
C[(16 * b0 + t0) * (256) + 16 * b1 + t1] = p_C[0][0];
C[(16 * b0 + t0 + 8) * (256) + 16 * b1 + t1] = p_C[1][0];
```

CARP Project

Design tools and techniques to aid **Correct and Efficient Accelerator Programming**

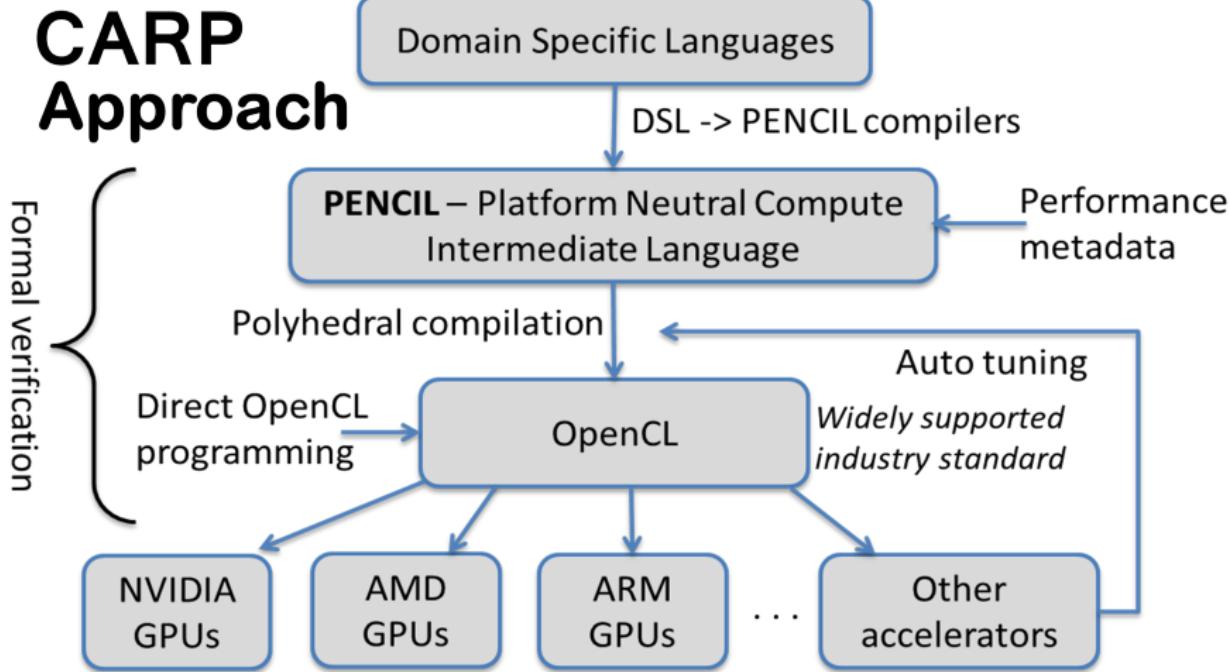
Key areas:

- High level programming models
- Advanced compilation techniques
- Formal verification

Partners:

- Imperial College London (UK)
- ENS (FR)
- ARM (UK)
- Realeyes (ES)
- RWTH Aachen University (DE)
- Monoidics (UK)
- University of Twente (NL)
- Rightware (FI)

CARP Approach



Outline

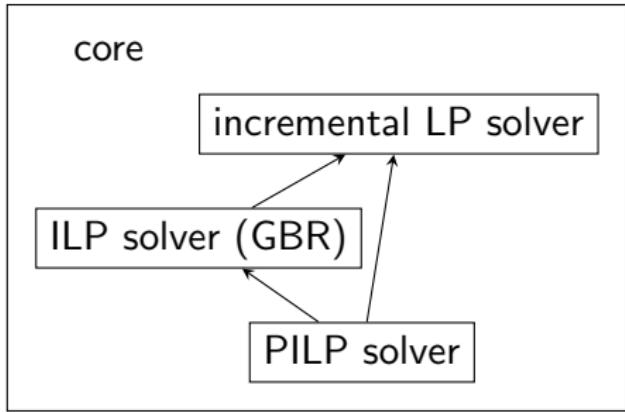
5 Tools

6 Internal Representation

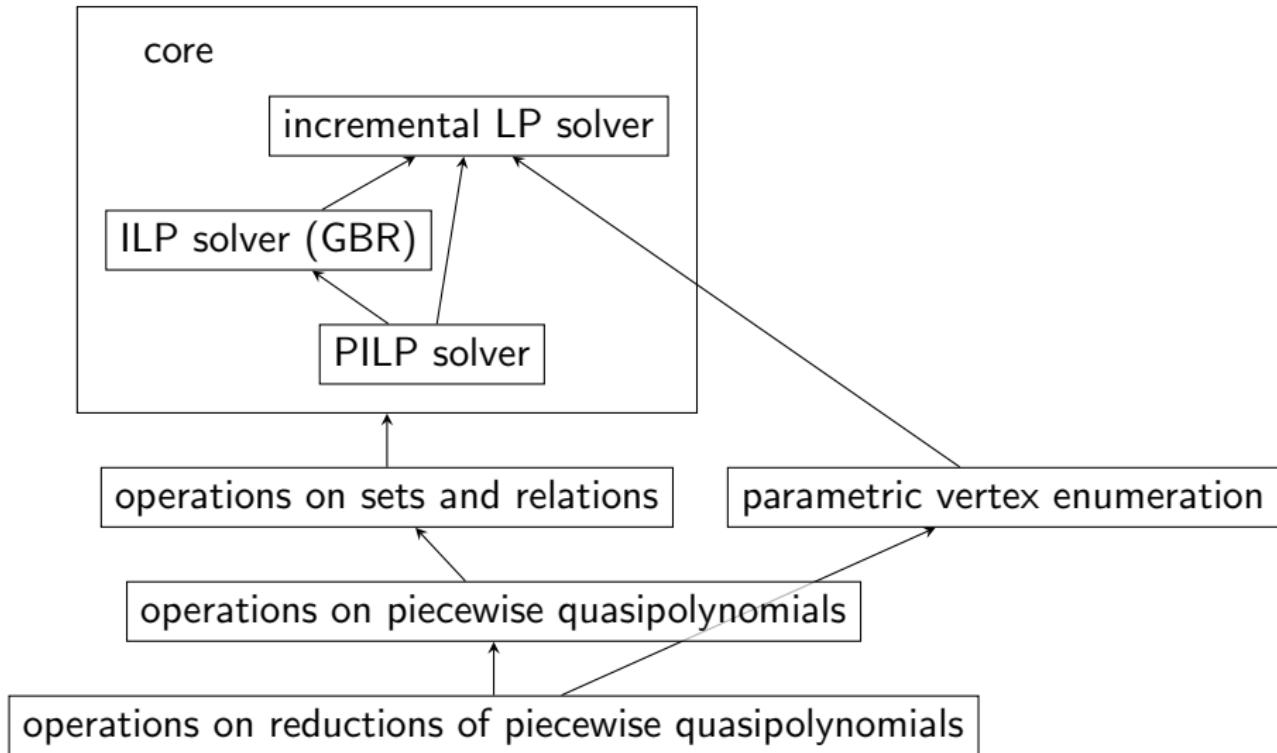
7 Weighted Counting

- Introduction
- Nested Sums
- Unweighted Counting
- Derivation
- Local Euler-Maclaurin Formulas
- Laurent Series Expansion
- Interpolation
- Set Operations

Internal Structure of isl



Internal Structure of isl



Internal Representation of Sets and Relations

Each set or relation is stored in disjunctive normal form

$$R = \bigcup_i R_i$$

$$R_i = \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : \exists \mathbf{k} : A_0 \mathbf{c} + A_1 \mathbf{i} + A_2 \mathbf{j} + A_3 \mathbf{k} \geq \mathbf{a} \}$$

Each disjunct consists of

- affine equalities and inequalities
 - symbolic constants **c**
 - existentially quantified variables **k**
 - ▶ optionally explicitly represented as integer division $k_i = \lfloor e_i/d_i \rfloor$
 - ▶ integer division also encoded in constraints $d_i k \leq e_i \leq d_i k + d - 1$
- ⇒ can be represented as the integer points in a rational polyhedron

Internal Representation of Sets and Relations

Each set or relation is stored in disjunctive normal form

$$R = \bigcup_i R_i$$

$$R_i = \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : \exists \mathbf{k} : A_0 \mathbf{c} + A_1 \mathbf{i} + A_2 \mathbf{j} + A_3 \mathbf{k} \geq \mathbf{a} \}$$

Each disjunct consists of

- affine equalities and inequalities
- symbolic constants \mathbf{c}
- existentially quantified variables \mathbf{k}
 - ▶ optionally explicitly represented as integer division $k_i = \lfloor e_i/d_i \rfloor$
 - ▶ integer division also encoded in constraints $d_i k \leq e_i \leq d_i k + d_i - 1$

⇒ can be represented as the integer points in a rational polyhedron

Question

What is the best internal representation?

Conversion to Disjunctive Normal Form

Disjunctive Normal Form: $\bigvee_i \exists \mathbf{x}_i \bigwedge_j$

$(\exists \mathbf{a} : f(\mathbf{x}, \mathbf{a})) \wedge (\exists \mathbf{b} : g(\mathbf{x}, \mathbf{b}))$	\rightarrow	$\exists \mathbf{a}, \mathbf{b} : f(\mathbf{x}, \mathbf{a}) \wedge g(\mathbf{x}, \mathbf{b})$
$\exists \mathbf{a} : f(\mathbf{x}, \mathbf{a}) \vee g(\mathbf{x}, \mathbf{a})$	\rightarrow	$(\exists \mathbf{a} : f(\mathbf{x}, \mathbf{a})) \vee (\exists \mathbf{a} : g(\mathbf{x}, \mathbf{a}))$
$f(\mathbf{x}) \wedge (g_1(\mathbf{x}) \vee g_2(\mathbf{x}))$	\rightarrow	$(f(\mathbf{x}) \wedge g_1(\mathbf{c})) \vee (f(\mathbf{x}) \wedge g_2(\mathbf{c}))$
$\neg(f_1 \vee f_2)$	\rightarrow	$\neg f_1 \wedge \neg f_2$
$\neg(f_1 \wedge f_2)$	\rightarrow	$\neg f_1 \vee \neg f_2$
$\neg(e \geq 0)$	\rightarrow	$e \leq -1$
$\neg(e = 0)$	\rightarrow	$(e \geq 1) \vee (e \leq -1)$
$\neg(\exists \mathbf{a} : f(\mathbf{x}, \mathbf{a}))$	\rightarrow	$\neg f(\mathbf{x}, \mathbf{g}(\mathbf{x}))$

Conversion to Disjunctive Normal Form

Disjunctive Normal Form: $\bigvee_i \exists \mathbf{x}_i \bigwedge_j$

$(\exists \mathbf{a} : f(\mathbf{x}, \mathbf{a})) \wedge (\exists \mathbf{b} : g(\mathbf{x}, \mathbf{b}))$	\rightarrow	$\exists \mathbf{a}, \mathbf{b} : f(\mathbf{x}, \mathbf{a}) \wedge g(\mathbf{x}, \mathbf{b})$
$\exists \mathbf{a} : f(\mathbf{x}, \mathbf{a}) \vee g(\mathbf{x}, \mathbf{a})$	\rightarrow	$(\exists \mathbf{a} : f(\mathbf{x}, \mathbf{a})) \vee (\exists \mathbf{a} : g(\mathbf{x}, \mathbf{a}))$
$f(\mathbf{x}) \wedge (g_1(\mathbf{x}) \vee g_2(\mathbf{x}))$	\rightarrow	$(f(\mathbf{x}) \wedge g_1(\mathbf{c})) \vee (f(\mathbf{x}) \wedge g_2(\mathbf{c}))$
$\neg(f_1 \vee f_2)$	\rightarrow	$\neg f_1 \wedge \neg f_2$
$\neg(f_1 \wedge f_2)$	\rightarrow	$\neg f_1 \vee \neg f_2$
$\neg(e \geq 0)$	\rightarrow	$e \leq -1$
$\neg(e = 0)$	\rightarrow	$(e \geq 1) \vee (e \leq -1)$
$\neg(\exists \mathbf{a} : f(\mathbf{x}, \mathbf{a}))$	\rightarrow	$\neg f(\mathbf{x}, g(\mathbf{x}))$

- ⇒ determine unique value of \mathbf{a} satisfying $f(\mathbf{x}, \mathbf{a})$ and write it as an explicit piecewise quasi affine expression $g(\mathbf{x})$ of \mathbf{x}
- ⇒ using parametric integer linear programming

Parametric Integer Linear Programming

[17]

$$R = \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : \exists \mathbf{k} : A_0 \mathbf{c} + A_1 \mathbf{i} + A_2 \mathbf{j} + A_3 \mathbf{k} \geq \mathbf{a} \}$$

Parametric integer linear programming lexicographic minimum of R as piecewise quasi affine expression

$$\begin{aligned}\text{lexmin } R &= \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) \in R : \forall \mathbf{j}' : S(\mathbf{i}) \rightarrow T(\mathbf{j}') \in R \Rightarrow \mathbf{j} \prec \mathbf{j}' \} \\ &= \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : \mathbf{j} = \mathbf{g}(\mathbf{i}) \}\end{aligned}$$

with \mathbf{g} a piecewise quasi affine expression

Technique: dual simplex + Gomory cuts

Parametric Integer Linear Programming

[17]

$$R = \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : \exists \mathbf{k} : A_0 \mathbf{c} + A_1 \mathbf{i} + A_2 \mathbf{j} + A_3 \mathbf{k} \geq \mathbf{a} \}$$

Parametric integer linear programming lexicographic minimum of R as piecewise quasi affine expression

$$\begin{aligned}\text{lexmin } R &= \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) \in R : \forall \mathbf{j}' : S(\mathbf{i}) \rightarrow T(\mathbf{j}') \in R \Rightarrow \mathbf{j} \prec \mathbf{j}' \} \\ &= \{ S(\mathbf{i}) \rightarrow T(\mathbf{j}) : \mathbf{j} = \mathbf{g}(\mathbf{i}) \}\end{aligned}$$

with \mathbf{g} a piecewise quasi affine expression

- ⇒ compute lexicographic minimum of
 $\{ (S(\mathbf{i}) \rightarrow T(\mathbf{j})) \rightarrow (\mathbf{k}) : A_0 \mathbf{c} + A_1 \mathbf{i} + A_2 \mathbf{j} + A_3 \mathbf{k} \geq \mathbf{a} \}$
- ⇒ explicit representation of \mathbf{k} in terms of \mathbf{i} and \mathbf{j} (+ symbolic constants \mathbf{c})

Technique: dual simplex + Gomory cuts

Quantifier Elimination

- Our Presburger language allows quantifier elimination because of the $\lfloor \cdot/d \rfloor$ function symbols
- More traditional Presburger language has divisibility predicate symbols “ $d \mid \cdot$ ” instead

Quantifier Elimination

- Our Presburger language allows quantifier elimination because of the $\lfloor \cdot/d \rfloor$ function symbols
- More traditional Presburger language has divisibility predicate symbols “ $d \mid \cdot$ ” instead

Quantifier elimination in `omega(+)` is based on these divisibility predicates [38, Section 3.1.1]

- ⇒ integer projection is split into
- ▶ “dark shadow”
 - ▶ “splinters” (expressed using divisibility predicates)

Quantifier Elimination

- Our Presburger language allows quantifier elimination because of the $\lfloor \cdot/d \rfloor$ function symbols
- More traditional Presburger language has divisibility predicate symbols “ $d \mid \cdot$ ” instead

Quantifier elimination in omega(+) is based on these divisibility predicates [38, Section 3.1.1]

- ⇒ integer projection is split into
- ▶ “dark shadow”
 - ▶ “splinters” (expressed using divisibility predicates)

Question

Which elimination is best?

Outline

5 Tools

6 Internal Representation

7 Weighted Counting

- Introduction
- Nested Sums
- Unweighted Counting
- Derivation
- Local Euler-Maclaurin Formulas
- Laurent Series Expansion
- Interpolation
- Set Operations

Weighted Counting — Reformulation

Recall

$$G = F \circ R$$

- R is a Presburger relation

$$R = \{ (\mathbf{s}) \rightarrow (\mathbf{t}) : f(\mathbf{s}, \mathbf{t}) \}$$

- F is a piecewise quasi polynomial

$$F = \{ (\mathbf{t}) \rightarrow n : n = f(\mathbf{t}) \}$$

$$f(\mathbf{t}) = \begin{cases} q_i(\mathbf{t}) & \text{if } f_i(\mathbf{t}) \\ \end{cases}$$

- G is a piecewise quasi polynomial

$$G(\mathbf{s}) = \sum_{\mathbf{t}: f(\mathbf{s}, \mathbf{t})} F(\mathbf{t})$$

Weighted Counting — Reformulation

Recall

$$G = F \circ R$$

- R is a Presburger relation

$$R = \{ (\mathbf{s}) \rightarrow (\mathbf{t}) : f(\mathbf{s}, \mathbf{t}) \}$$

- F is a piecewise quasi polynomial

$$F = \{ (\mathbf{t}) \rightarrow n : n = f(\mathbf{t}) \}$$

$$f(\mathbf{t}) = \begin{cases} q_i(\mathbf{t}) & \text{if } f_i(\mathbf{t}) \\ & \dots \end{cases}$$

- G is a piecewise quasi polynomial

$$G(\mathbf{s}) = \sum_{\mathbf{t}: f(\mathbf{s}, \mathbf{t})} F(\mathbf{t})$$

$$= \sum_i \sum_{\mathbf{t}: f(\mathbf{s}, \mathbf{t}) \wedge f_i(\mathbf{t})} q_i(\mathbf{t})$$

Weighted Counting — Reformulation

Recall

$$G = F \circ R$$

- R is a Presburger relation

$$R = \{ (\mathbf{s}) \rightarrow (\mathbf{t}) : f(\mathbf{s}, \mathbf{t}) \}$$

- F is a piecewise quasi polynomial

$$F = \{ (\mathbf{t}) \rightarrow n : n = f(\mathbf{t}) \}$$

$$f(\mathbf{t}) = \begin{cases} q_i(\mathbf{t}) & \text{if } f_i(\mathbf{t}) \end{cases}$$

- G is a piecewise quasi polynomial

$$G(\mathbf{s}) = \sum_{\mathbf{t}: f(\mathbf{s}, \mathbf{t})} F(\mathbf{t})$$

$$= \sum_i \sum_{\mathbf{t}: f(\mathbf{s}, \mathbf{t}) \wedge f_i(\mathbf{t})} q_i(\mathbf{t})$$

$$= \sum_j \sum_{\mathbf{t} \in P_j(\mathbf{s}) \cap \mathbb{Z}^d} q_i(\mathbf{t})$$

DNF + make disjoint + PILP

the integer points in a parametric polytope

Weighted Counting — Example

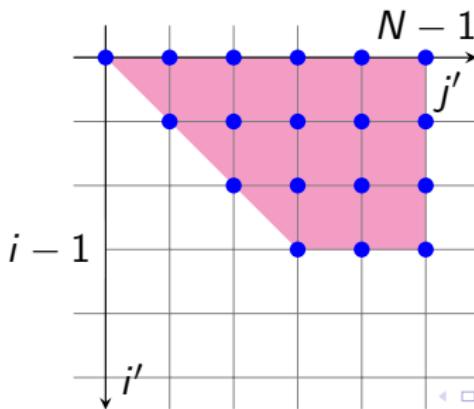
$$G(\mathbf{s}) = \sum_j \sum_{\mathbf{t} \in P_j(\mathbf{s}) \cap \mathbb{Z}^d} q_i(\mathbf{t})$$

Example:

$$\sum_{i', j' \in P(N, i, j)} j' \left\lfloor \frac{j' - i'}{4} \right\rfloor$$

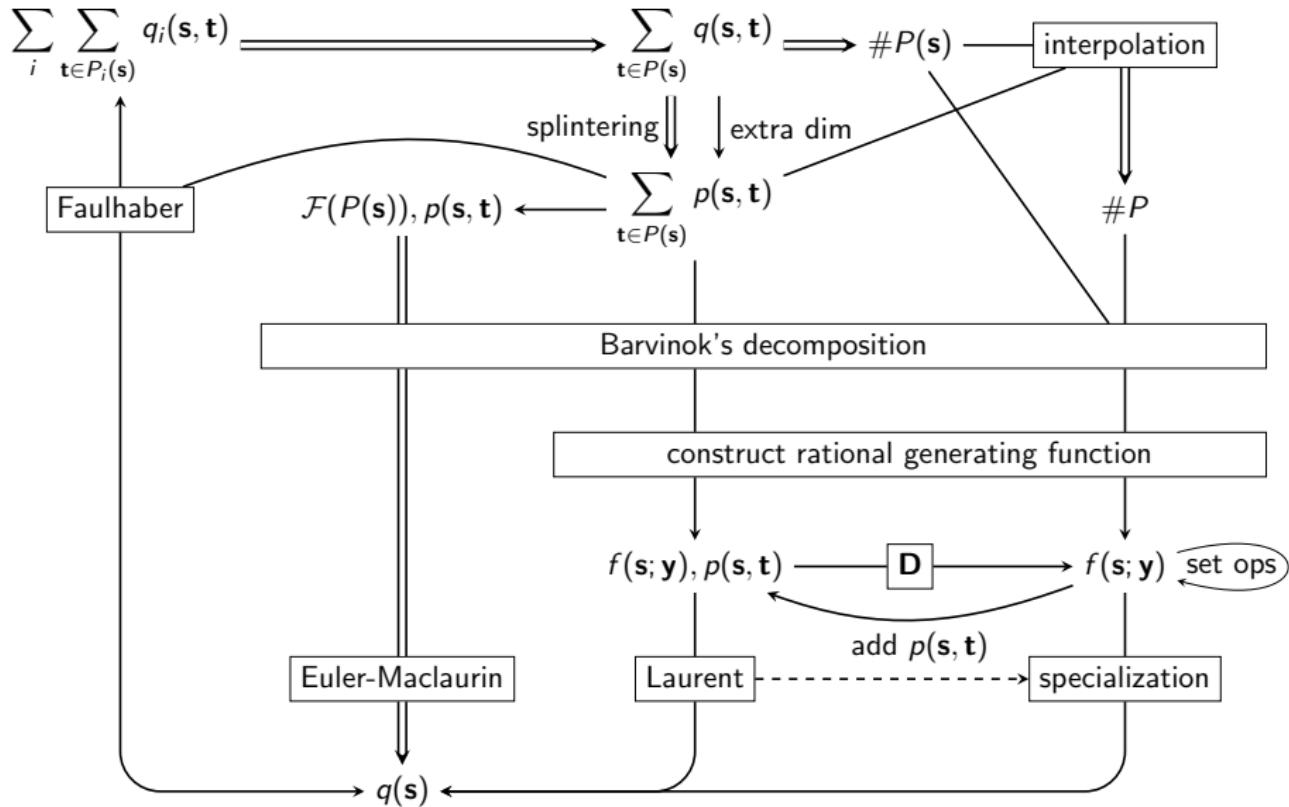
with

$$P(N, i, j) = \{ (i', j') : 0 \leq i' < i \wedge i' \leq j' < N \}$$



Weighted Counting Overview

[32, 29]



Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{\mathbf{t} \in P_i(\mathbf{s})} q_i(\mathbf{s}, \mathbf{t}) \xrightarrow{\hspace{1cm}} \sum_{\mathbf{t} \in P(\mathbf{s})} q(\mathbf{s}, \mathbf{t})$$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{\mathbf{t} \in P_i(\mathbf{s})} q_i(\mathbf{s}, \mathbf{t}) \xrightarrow{\text{splintering}} \sum_{\mathbf{t} \in P(\mathbf{s})} q(\mathbf{s}, \mathbf{t})$$

↓

$$\sum_{\mathbf{t} \in P(\mathbf{s})} p(\mathbf{s}, \mathbf{t})$$

From Quasi Polynomial to Polynomial

$$\text{Quasi monomial } \prod_i \left[\frac{a_{i0}t + \sum_j a_{ij}s_j}{D_i} \right]^{m_i} j' \left[\frac{j' - i'}{4} \right]$$

- Splintering (on single dimension t)

⇒ Split domain into D parts (t replaced by t')

$$t = Dt' + k \quad \text{for } 0 \leq k < D, D = \text{lcm}_i D_i$$

$$i' = 4t' + k \quad \text{for } 0 \leq k < 4$$

From Quasi Polynomial to Polynomial

$$\text{Quasi monomial } \prod_i \left[\frac{a_{i0}t + \sum_j a_{ij}s_j}{D_i} \right]^{m_i} \quad j' \left[\frac{j' - i'}{4} \right]$$

- Splintering (on single dimension t)

⇒ Split domain into D parts (t replaced by t') $j' \left(-t' + \left[\frac{j' - k}{4} \right] \right)$

$$t = Dt' + k \quad \text{for } 0 \leq k < D, D = \text{lcm}_i D_i$$

$$i' = 4t' + k \quad \text{for } 0 \leq k < 4$$

From Quasi Polynomial to Polynomial

$$\text{Quasi monomial } \prod_i \left[\frac{a_{i0}t + \sum_j a_{ij}s_j}{D_i} \right]^{m_i} \quad j' \left[\frac{j' - i'}{4} \right]$$

- Splintering (on single dimension t)

$$\Rightarrow \text{Split domain into } D \text{ parts (} t \text{ replaced by } t' \text{)} \quad j' \left(-t' + \left[\frac{j' - k}{4} \right] \right)$$

$$t = Dt' + k \quad \text{for } 0 \leq k < D, D = \text{lcm}_i D_i$$

$$i' = 4t' + k \quad \text{for } 0 \leq k < 4$$

Domain: $P(N, i, j) = \{(i', j') : 0 \leq i' < i \wedge i' \leq j' < N\}$

Split:

$$\begin{array}{lll} k=0 & \{(t, j') : 0 \leq 4t < i \wedge 4t \leq j' < N\} & j'(-t' + \lfloor j'/4 \rfloor) \\ k=1 & \{(t, j') : 0 \leq 4t+1 < i \wedge 4t+1 \leq j' < N\} & j'(-t' + \lfloor (j'-1)/4 \rfloor) \\ k=2 & \{(t, j') : 0 \leq 4t+2 < i \wedge 4t+2 \leq j' < N\} & j'(-t' + \lfloor (j'-2)/4 \rfloor) \\ k=3 & \{(t, j') : 0 \leq 4t+3 < i \wedge 4t+3 \leq j' < N\} & j'(-t' + \lfloor (j'-3)/4 \rfloor) \end{array}$$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{\mathbf{t} \in P_i(\mathbf{s})} q_i(\mathbf{s}, \mathbf{t}) \xrightarrow{\text{splintering}} \sum_{\mathbf{t} \in P(\mathbf{s})} q(\mathbf{s}, \mathbf{t})$$

↓

$$\sum_{\mathbf{t} \in P(\mathbf{s})} p(\mathbf{s}, \mathbf{t})$$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{\mathbf{t} \in P_i(\mathbf{s})} q_i(\mathbf{s}, \mathbf{t}) \longrightarrow \sum_{\mathbf{t} \in P(\mathbf{s})} q(\mathbf{s}, \mathbf{t})$$

splintering

$$\sum_{\mathbf{t} \in P(\mathbf{s})} p(\mathbf{s}, \mathbf{t})$$

Faulhaber

$$q(\mathbf{s})$$

Nested Sums

[26, 28]

Consider a single variable t

$$\sum_{t=0}^{q_{-1}(\mathbf{s})} \sum_{k=0}^m q_k(\mathbf{s}) t^k = \sum_{t'=0}^{\lfloor \frac{i-1}{4} \rfloor} \left(j' \left\lfloor \frac{j'}{4} \right\rfloor - j' t' \right)$$

Nested Sums

[26, 28]

Consider a single variable t

$$\sum_{t=0}^{q_{-1}(\mathbf{s})} \sum_{k=0}^m q_k(\mathbf{s}) t^k = \sum_{t'=0}^{\lfloor \frac{i-1}{4} \rfloor} \left(j' \left\lfloor \frac{j'}{4} \right\rfloor - j' t' \right)$$

⇒ use Faulhaber's formula ($k > 0$)

$$\sum_{t=0}^{m-1} t^k = \frac{1}{k+1} \sum_{n=0}^k \binom{k+1}{n} B_n m^{k+1-n}$$

with B_n the Bernoulli numbers ($B_0 = 1, B_1 = -1/2, B_2 = 1/6, \dots$)

⇒ special case for constant term

$$\sum_{t=0}^{m-1} c = c m$$

Nested Sums

[26, 28]

Consider a single variable t

$$\sum_{t=0}^{q_{-1}(s)} \sum_{k=0}^m q_k(s) t^k \quad \sum_{t'=0}^{\left\lfloor \frac{i-1}{4} \right\rfloor} \left(j' \left\lfloor \frac{j'}{4} \right\rfloor - j' t' \right)^{m-1}$$

\Rightarrow use Faulhaber's formula ($k > 0$)

$$\sum_{t=0}^{m-1} t^k = \frac{1}{k+1} \sum_{n=0}^k \binom{k+1}{n} B_n m^{k+1-n}$$

 $k = 1$

with B_n the Bernoulli numbers ($B_0 = 1, B_1 = -1/2, B_2 = 1/6, \dots$)

\Rightarrow special case for constant term

$$\sum_{t=0}^{m-1} c = c m$$

$$\overbrace{j' \left\lfloor \frac{j'}{4} \right\rfloor \left(\left\lfloor \frac{i-1}{4} \right\rfloor + 1 \right)} - \overbrace{j' \left(\left(\left\lfloor \frac{i-1}{4} \right\rfloor + 1 \right)^2 + 2 \left(\frac{-1}{2} \right) \left(\left\lfloor \frac{i-1}{4} \right\rfloor + 1 \right) \right)}$$

Nested Sums (2)

[26, 28]

Consider a single variable t

$$\sum_{t=0}^{q_{-1}(\mathbf{s})} \sum_{k=0}^m q_k(\mathbf{s}) t^k$$

- assumes single upper bound and lower bound equal to 0

Nested Sums (2)

[26, 28]

Consider a single variable t

$$\sum_{t=0}^{q_{-1}(\mathbf{s})} \sum_{k=0}^m q_k(\mathbf{s}) t^k$$

- assumes single upper bound and lower bound equal to 0
 - ▶ n upper bounds u_i $u_1 = N$ and $u_2 = M$ ($t \leq N \wedge t \leq M$)
 - ⇒ split into n parts with a unique upper bound
$$u_j \leq u_{j'} \quad \text{for } j' < j$$
$$u_j < u_{j'} \quad \text{for } j' > j$$
 - ★ $j = 1$: $N < M$, upper bound: $t \leq N$
 - ★ $j = 2$: $M \leq N$, upper bound: $t \leq M$

Nested Sums (2)

[26, 28]

Consider a single variable t

$$\sum_{t=0}^{q_{-1}(\mathbf{s})} \sum_{k=0}^m q_k(\mathbf{s}) t^k$$

- assumes single upper bound and lower bound equal to 0
 - ▶ n upper bounds u_i $u_1 = N$ and $u_2 = M$ ($t \leq N \wedge t \leq M$)
 - ⇒ split into n parts with a unique upper bound
$$u_j \leq u_{j'} \quad \text{for } j' < j$$
$$u_j < u_{j'} \quad \text{for } j' > j$$
 - ★ $j = 1$: $N < M$, upper bound: $t \leq N$
 - ★ $j = 2$: $M \leq N$, upper bound: $t \leq M$
 - ▶ lower bound not equal to 0
 - ⇒ shift, split and/or shift

Nested Sums (2)

[26, 28]

Consider a single variable t

$$\sum_{t=0}^{q_{-1}(\mathbf{s})} \sum_{k=0}^m q_k(\mathbf{s}) t^k$$

- assumes single upper bound and lower bound equal to 0
 - ▶ n upper bounds u_i $u_1 = N$ and $u_2 = M$ ($t \leq N \wedge t \leq M$)
 - ⇒ split into n parts with a unique upper bound
$$u_j \leq u_{j'} \quad \text{for } j' < j$$
$$u_j < u_{j'} \quad \text{for } j' > j$$
 - ★ $j = 1$: $N < M$, upper bound: $t \leq N$
 - ★ $j = 2$: $M \leq N$, upper bound: $t \leq M$
 - ▶ lower bound not equal to 0
 - ⇒ shift, split and/or shift
- single variable
 - ⇒ recurse over all variables

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{\mathbf{t} \in P_i(\mathbf{s})} q_i(\mathbf{s}, \mathbf{t}) \longrightarrow \sum_{\mathbf{t} \in P(\mathbf{s})} q(\mathbf{s}, \mathbf{t})$$

splintering

$$\sum_{\mathbf{t} \in P(\mathbf{s})} p(\mathbf{s}, \mathbf{t})$$

Faulhaber

$$q(\mathbf{s})$$

Weighted Counting Overview: Nested Sums

[32, 29]

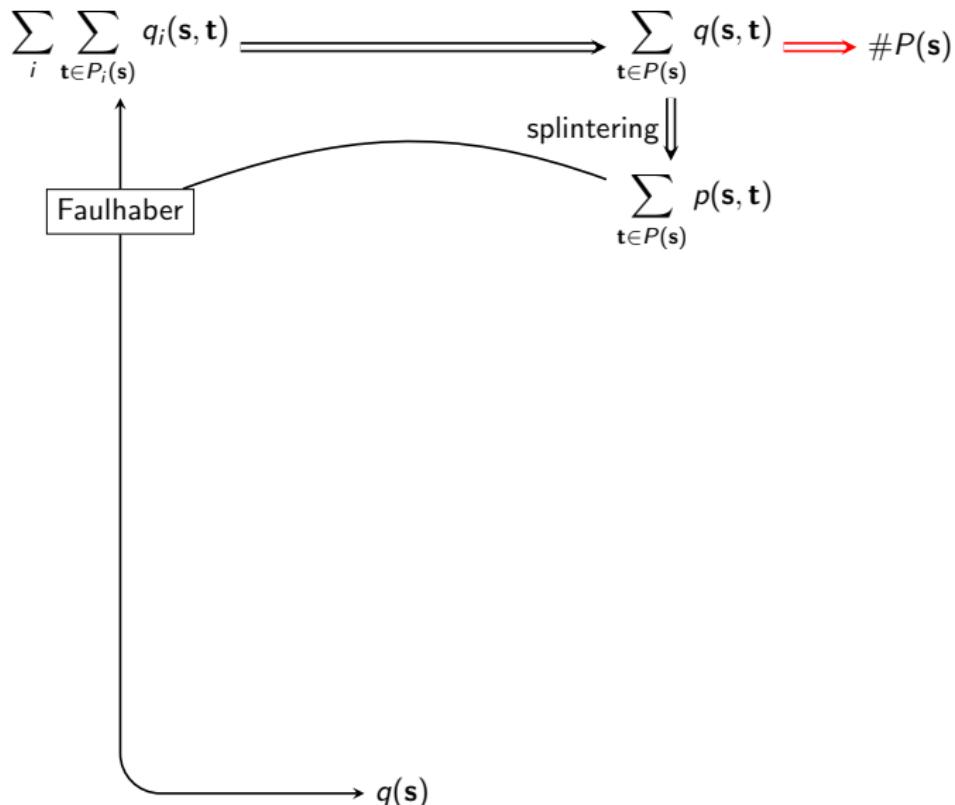
$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t)$$

↑
Faulhaber ↗
splintering ↘
↓
 $\sum_{t \in P(s)} p(s, t)$

$$q(s) \rightarrow$$

Weighted Counting Overview

[32, 29]



From Weighted Counting to Unweighted Counting

Quasi monomial $\prod_{i=1}^m \left\lfloor \frac{\sum_j a_{ij} s_j + \sum_j b_{ij} t_j}{D_i} \right\rfloor^{j' \left\lfloor \frac{j' - i'}{4} \right\rfloor}$

From Weighted Counting to Unweighted Counting

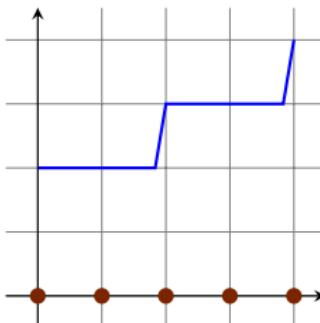
Quasi monomial $\prod_{i=1}^m \left\lfloor \frac{\sum_j a_{ij} s_j + \sum_j b_{ij} t_j}{D_i} \right\rfloor^{j' \left\lfloor \frac{j' - i'}{4} \right\rfloor}$

- ⇒ volume of (parametric) m -dimensional box (provided factors ≥ 0)
- ⇒ add extra variables u_i for each factor

$$1 \leq u_i \quad \text{and} \quad D_i u_i \leq \sum_j a_{ij} s_j + \sum_j b_{ij} t_j$$

$$P = \{ (x) : 0 \leq x \leq 4 \}$$

$$f = \left\lfloor \frac{x+4}{2} \right\rfloor$$



From Weighted Counting to Unweighted Counting

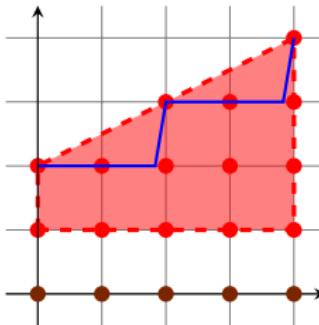
$$\text{Quasi monomial } \prod_{i=1}^m \left\lfloor \frac{\sum_j a_{ij} s_j + \sum_j b_{ij} t_j}{D_i} \right\rfloor^{j' \left\lfloor \frac{j' - i'}{4} \right\rfloor}$$

- ⇒ volume of (parametric) m -dimensional box (provided factors ≥ 0)
- ⇒ add extra variables u_i for each factor

$$1 \leq u_i \quad \text{and} \quad D_i u_i \leq \sum_j a_{ij} s_j + \sum_j b_{ij} t_j$$

$$P = \{ (x) : 0 \leq x \leq 4 \}$$

$$f = \left\lfloor \frac{x+4}{2} \right\rfloor$$



$$P' = \{ (x, u) : 0 \leq x \leq 4 \wedge 1 \leq u \wedge 2u \leq x + 4 \}$$

From Weighted Counting to Unweighted Counting

$$\text{Quasi monomial } \prod_{i=1}^m \left\lfloor \frac{\sum_j a_{ij} s_j + \sum_j b_{ij} t_j}{D_i} \right\rfloor^{j' \left\lfloor \frac{j' - i'}{4} \right\rfloor}$$

- ⇒ volume of (parametric) m -dimensional box (provided factors ≥ 0)
- ⇒ add extra variables u_i for each factor

$$1 \leq u_i \quad \text{and} \quad D_i u_i \leq \sum_j a_{ij} s_j + \sum_j b_{ij} t_j$$

$$\sum_{(i', j') \in P} j' \left\lfloor \frac{j' - i'}{4} \right\rfloor = \#P'$$

$$P = \{(i', j') : 0 \leq i' < i \wedge i' \leq j' < N\}$$

$$P' = \{(i', j', u_1, u_2) : 1 \leq u_1 \leq j' \wedge 1 \leq u_2 \leq j' - i' \wedge (i', j') \in P\}$$

From Weighted Counting to Unweighted Counting

$$\text{Quasi monomial } \prod_{i=1}^m \left\lfloor \frac{\sum_j a_{ij} s_j + \sum_j b_{ij} t_j}{D_i} \right\rfloor^{j' \left\lfloor \frac{j' - i'}{4} \right\rfloor}$$

- ⇒ volume of (parametric) m -dimensional box (provided factors ≥ 0)
- ⇒ add extra variables u_i for each factor

$$1 \leq u_i \quad \text{and} \quad D_i u_i \leq \sum_j a_{ij} s_j + \sum_j b_{ij} t_j$$

$$\sum_{(i', j') \in P} j' \left\lfloor \frac{j' - i'}{4} \right\rfloor = \#P'$$

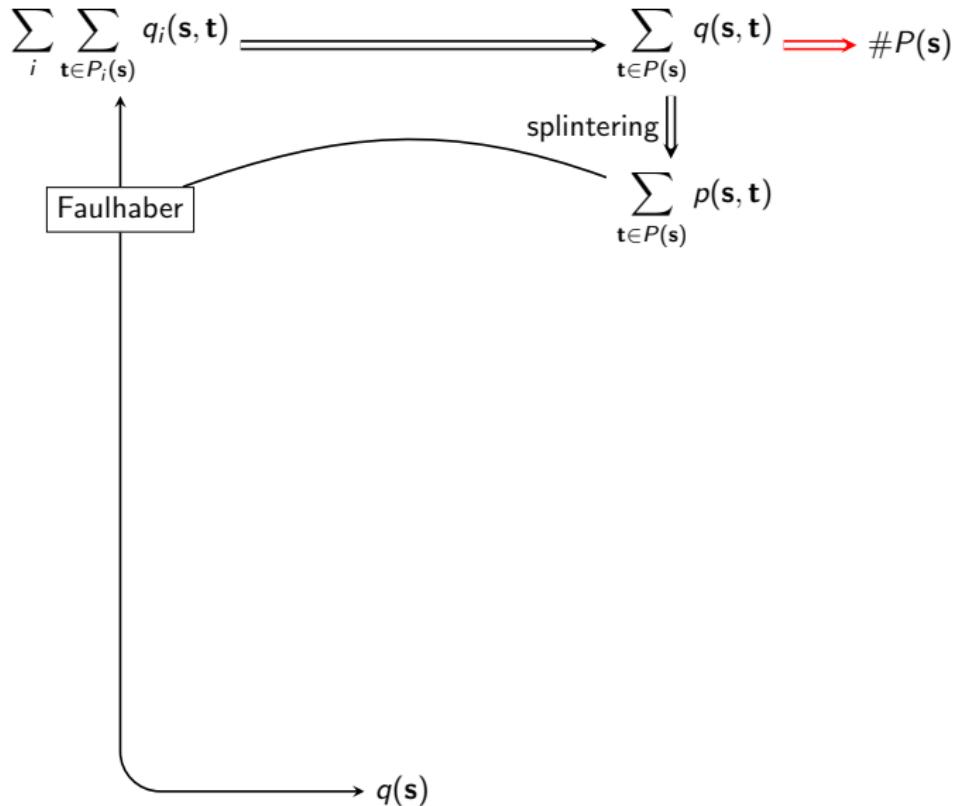
$$P = \{(i', j') : 0 \leq i' < i \wedge i' \leq j' < N\}$$

$$P' = \{(i', j', u_1, u_2) : 1 \leq u_1 \leq j' \wedge 1 \leq u_2 \wedge 4u_2 \leq j' - i' \wedge (i', j') \in P\}$$

+ split into orthants and shift floor arguments

Weighted Counting Overview

[32, 29]



Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$



$$\sum_{t \in P(s)} p(s, t)$$

Barvinok's decomposition

construct rational generating function

$$f(s; y)$$

specialization

$$q(s)$$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$

Faulhaber

splintering

$$\sum_{t \in P(s)} p(s, t)$$

Barvinok's decomposition

construct rational generating function

 $f(s; y)$

specialization

 $q(s)$

Generating Functions

[6, 15, 36]

- Iverson bracket

$$[P] = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{if } P \text{ is false} \end{cases}$$

Generating Functions

[6, 15, 36]

- Iverson bracket

$$[P] = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{if } P \text{ is false} \end{cases}$$

- Indicator function of a set $S \subseteq \mathbb{Z}^d$

$$[S] : \mathbb{Z}^d \rightarrow \{0, 1\} : s \mapsto [s \in S]$$

Generating Functions

[6, 15, 36]

- Iverson bracket

$$[P] = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{if } P \text{ is false} \end{cases}$$

- Indicator function of a set $S \subseteq \mathbb{Z}^d$

$$[S] : \mathbb{Z}^d \rightarrow \{0, 1\} : s \mapsto [s \in S]$$

- Generating function of a set $S \subseteq \mathbb{Z}^d$ is the generating function of its indicator function

$$\sum_{s \in \mathbb{Z}^d} [s \in S] x^s$$

Generating Functions

[6, 15, 36]

- Iverson bracket

$$[P] = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{if } P \text{ is false} \end{cases}$$

- Indicator function of a set $S \subseteq \mathbb{Z}^d$

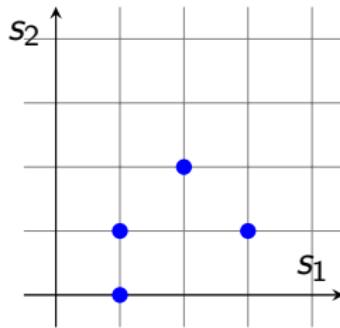
$$[S] : \mathbb{Z}^d \rightarrow \{0, 1\} : s \mapsto [s \in S]$$

- Generating function of a set $S \subseteq \mathbb{Z}^d$ is the generating function of its indicator function

$$\sum_{s \in \mathbb{Z}^d} [s \in S] x^s$$

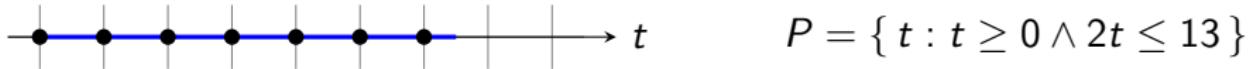
$$S = \{(3, 1), (2, 2), (1, 1), (1, 0)\}$$

$$\begin{aligned} f(S; x) &= x^{(3,1)} + x^{(2,2)} + x^{(1,1)} + x^{(1,0)} \\ &= x_1^3 x_2^1 + x_1^2 x_2^2 + x_1^1 x_2^1 + x_1^1 x_2^0 \end{aligned}$$



The Use of Generating Functions for Counting

[6, 15, 36]

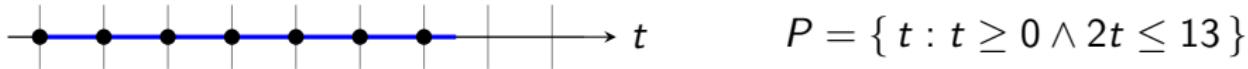


Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P
⇒ 0 or 1

The Use of Generating Functions for Counting

[6, 15, 36]



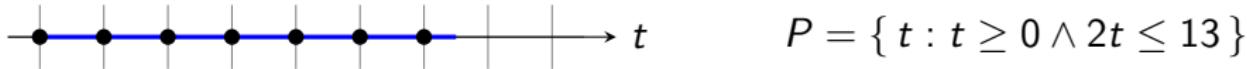
Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P
⇒ 0 or 1

$$[t \in P] = \begin{cases} 1 & \text{if } 0 \leq t \leq 6 \\ 0 & \text{otherwise} \end{cases}$$

The Use of Generating Functions for Counting

[6, 15, 36]



Counting the integer points in polytope P

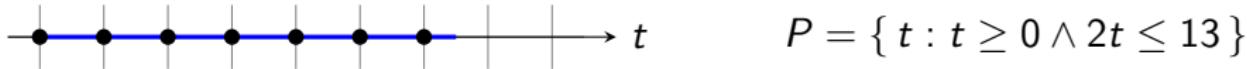
- For each point in \mathbb{Z} count the number of occurrences in P
⇒ 0 or 1

$$[t \in P] = \begin{cases} 1 & \text{if } 0 \leq t \leq 6 \\ 0 & \text{otherwise} \end{cases}$$

- Sum over all points

The Use of Generating Functions for Counting

[6, 15, 36]



Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P
 $\Rightarrow 0$ or 1

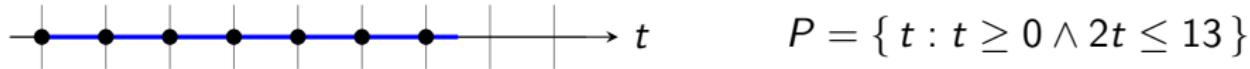
$$[t \in P] = \begin{cases} 1 & \text{if } 0 \leq t \leq 6 \\ 0 & \text{otherwise} \end{cases}$$

- Sum over all points

$$\#P = \sum_{t \in \mathbb{Z}} [t \in P] = 7$$

The Use of Generating Functions for Counting

[6, 15, 36]



Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P
⇒ 0 or 1

$$[t \in P] = \begin{cases} 1 & \text{if } 0 \leq t \leq 6 \\ 0 & \text{otherwise} \end{cases}$$

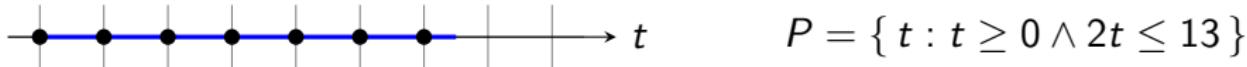
- Sum over all points

$$\#P = \sum_{t \in \mathbb{Z}} [t \in P] = 7$$

- Using generating functions $f(P; x) = \sum_{t \in \mathbb{Z}} [t \in P] x^t$

The Use of Generating Functions for Counting

[6, 15, 36]



$$P = \{ t : t \geq 0 \wedge 2t \leq 13 \}$$

Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P
 $\Rightarrow 0$ or 1

$$[t \in P] = \begin{cases} 1 & \text{if } 0 \leq t \leq 6 \\ 0 & \text{otherwise} \end{cases}$$

$$f(P; x) = x^0 + x^1 + x^2 + x^3 + x^4 + x^5 + x^6$$

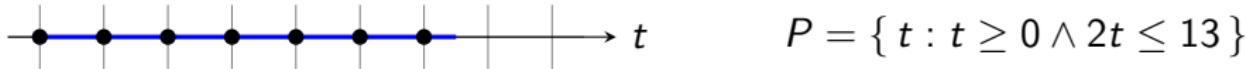
- Sum over all points

$$\#P = \sum_{t \in \mathbb{Z}} [t \in P] = 7$$

- Using generating functions $f(P; x) = \sum_{t \in \mathbb{Z}} [t \in P] x^t$

The Use of Generating Functions for Counting

[6, 15, 36]



Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P
⇒ 0 or 1

$$[t \in P] = \begin{cases} 1 & \text{if } 0 \leq t \leq 6 \\ 0 & \text{otherwise} \end{cases}$$

$$f(P; x) = x^0 + x^1 + x^2 + x^3 + x^4 + x^5 + x^6$$

- Sum over all points

$$\#P = \sum_{t \in \mathbb{Z}} [t \in P] = 7 = 1^0 + 1^1 + 1^2 + 1^3 + 1^4 + 1^5 + 1^6 = f(P; 1)$$

- Using generating functions $f(P; x) = \sum_{t \in \mathbb{Z}} [t \in P] x^t$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$

Faulhaber

splintering

$$\sum_{t \in P(s)} p(s, t)$$

Barvinok's decomposition

construct rational generating function

 $f(s; y)$

specialization

 $q(s)$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$

Faulhaber

splintering

$$\sum_{t \in P(s)} p(s, t)$$

Barvinok's decomposition

construct rational generating function

$f(s; y)$

specialization

$q(s)$

Rational Generating Functions

[6, 15, 36]

Generating function:

$$\sum_{\mathbf{s} \in \mathbb{Z}^d} [\mathbf{s} \in S] \mathbf{x}^{\mathbf{s}}$$

Theorem: If $S \subseteq Q$, with Q a pointed polyhedron, then $f(S; \mathbf{x})$ converges to rational function

Rational Generating Functions

[6, 15, 36]

Generating function:

$$\sum_{\mathbf{s} \in \mathbb{Z}^d} [\mathbf{s} \in S] \mathbf{x}^{\mathbf{s}}$$

Theorem: If $S \subseteq Q$, with Q a pointed polyhedron, then $f(S; \mathbf{x})$ converges to rational function

Example:

$$\frac{1}{1-x} = x^0 + x^1 + x^2 + x^3 + \dots \quad \text{if } x < 1$$

Rational Generating Functions

[6, 15, 36]

Generating function:

$$\sum_{\mathbf{s} \in \mathbb{Z}^d} [\mathbf{s} \in S] \mathbf{x}^{\mathbf{s}}$$

Theorem: If $S \subseteq Q$, with Q a pointed polyhedron, then $f(S; \mathbf{x})$ converges to rational function

Example:

$$\frac{1}{1-x} = x^0 + x^1 + x^2 + x^3 + \dots \quad \text{if } x < 1$$
$$\frac{-x^{-1}}{1-x^{-1}} = x^{-1} + x^{-2} + x^{-3} + x^{-4} + \dots \quad \text{if } x > 1$$

⇒ different generating functions may converge to same rational function
(on different regions)

Rational Generating Functions

[6, 15, 36]

Generating function:

$$\sum_{\mathbf{s} \in \mathbb{Z}^d} [\mathbf{s} \in S] \mathbf{x}^{\mathbf{s}}$$

Theorem: If $S \subseteq Q$, with Q a pointed polyhedron, then $f(S; \mathbf{x})$ converges to rational function

Example:

$$\begin{aligned} \frac{1}{1-x} &= x^0 + x^1 + x^2 + x^3 + \dots && \text{if } x < 1 \\ \frac{-x^{-1}}{1-x^{-1}} &= x^{-1} + x^{-2} + x^{-3} + x^{-4} + \dots && \text{if } x > 1 \end{aligned}$$

⇒ different generating functions may converge to same rational function
(on different regions)

Rational Generating Functions

[6, 15, 36]

Generating function:

$$\sum_{\mathbf{s} \in \mathbb{Z}^d} [\mathbf{s} \in S] \mathbf{x}^{\mathbf{s}}$$

Theorem: If $S \subseteq Q$, with Q a pointed polyhedron, then $f(S; \mathbf{x})$ converges to rational function

Example:

$$\begin{aligned}
 &= \frac{1}{1-x} = x^0 + x^1 + x^2 + x^3 + \dots && \text{if } x < 1 \\
 &= \frac{-x^{-1}}{1-x^{-1}} = x^{-1} + x^{-2} + x^{-3} + x^{-4} + \dots && \text{if } x > 1
 \end{aligned}$$

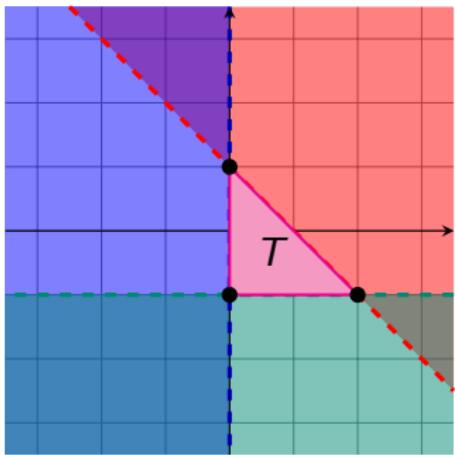
⇒ different generating functions may converge to same rational function
(on different regions)

Rational generating function of polyhedron with non-trivial lineality space is defined to be 0

Generating Function of a Polytope

[6, 15, 36]

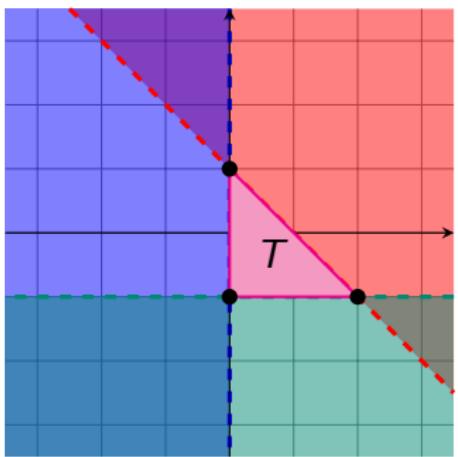
$$[\mathbb{Q}^2] - [T] = [H_1] + [H_2] + [H_3] - [H_1 \cap H_2] - [H_2 \cap H_3] - [H_3 \cap H_1]$$



Generating Function of a Polytope

[6, 15, 36]

$$[\mathbb{Q}^2] - [T] = [H_1] + [H_2] + [H_3] - [H_1 \cap H_2] - [H_2 \cap H_3] - [H_3 \cap H_1]$$

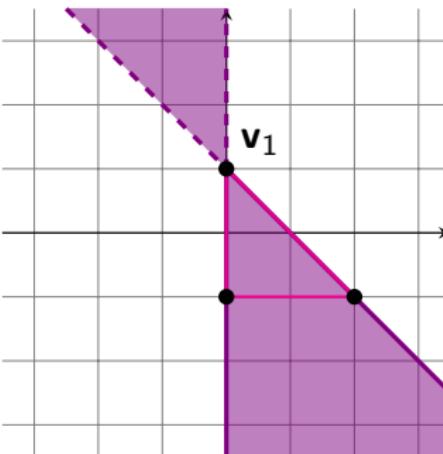
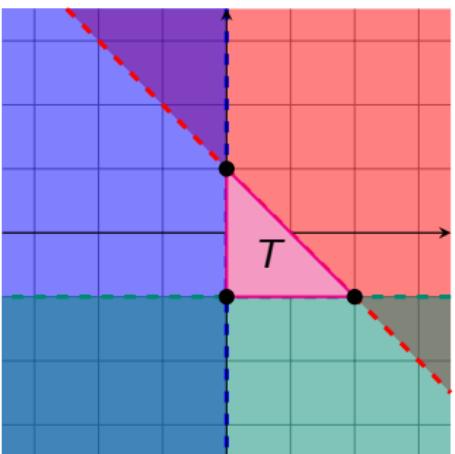


$$[T] = [H_1 \cap H_2] + [H_2 \cap H_3] + [H_3 \cap H_1] \quad \text{mod } \mathcal{L}$$

Generating Function of a Polytope

[6, 15, 36]

$$[\mathbb{Q}^2] - [T] = [H_1] + [H_2] + [H_3] - [H_1 \cap H_2] - [H_2 \cap H_3] - [H_3 \cap H_1]$$



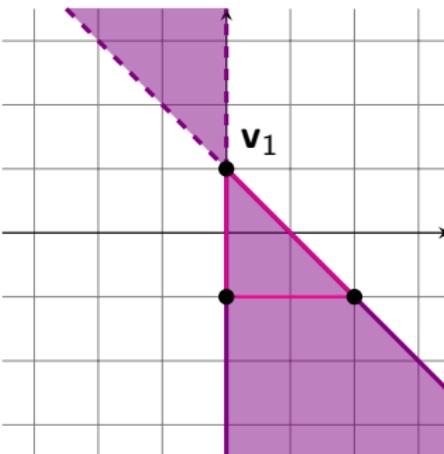
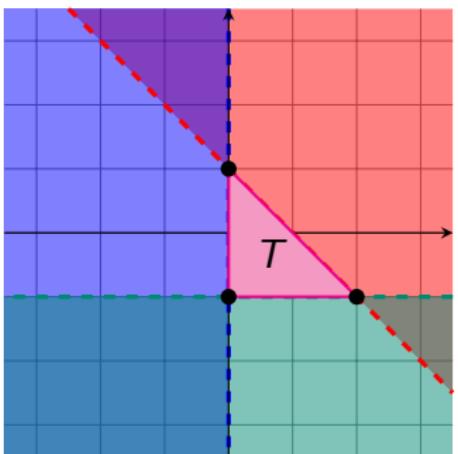
$$[T] = [H_1 \cap H_2] + [H_2 \cap H_3] + [H_3 \cap H_1] \quad \text{mod } \mathcal{L}$$

$$f(H_1 \cap H_2; \mathbf{x}) = f(\text{cone}(T, \mathbf{v}_1); \mathbf{x})$$

Generating Function of a Polytope

[6, 15, 36]

$$[\mathbb{Q}^2] - [T] = [H_1] + [H_2] + [H_3] - [H_1 \cap H_2] - [H_2 \cap H_3] - [H_3 \cap H_1]$$



$$[T] = [H_1 \cap H_2] + [H_2 \cap H_3] + [H_3 \cap H_1] \quad \text{mod } \mathcal{L}$$

$$f(H_1 \cap H_2; \mathbf{x}) = f(\text{cone}(T, \mathbf{v}_1); \mathbf{x})$$

$$f(P; \mathbf{x}) = \sum_{\mathbf{v} \in V(P)} f(\text{cone}(P, \mathbf{v}); \mathbf{x})$$

Generating Function of a Unimodular Cone

[6, 15, 36]

A d -dimensional cone K with d extremal rays \mathbf{u}_i is **unimodular** if each integer element can be written (uniquely) as a non-negative **integer** combination of the extremal rays

$$K \cap \mathbb{Z}^d = \left\{ \sum_i \lambda_i \mathbf{u}_i : \lambda_i \in \mathbb{Z}_{\geq 0} \right\}$$

Generating Function of a Unimodular Cone

[6, 15, 36]

A d -dimensional cone K with d extremal rays \mathbf{u}_i is **unimodular** if each integer element can be written (uniquely) as a non-negative **integer** combination of the extremal rays

$$K \cap \mathbb{Z}^d = \left\{ \sum_i \lambda_i \mathbf{u}_i : \lambda_i \in \mathbb{Z}_{\geq 0} \right\}$$

$$f(K; \mathbf{x}) = \sum_{\mathbf{m} \in K \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{m}} = \prod_{i=1}^d \frac{1}{1 - \mathbf{x}^{\mathbf{u}_i}}$$

Generating Function of a Unimodular Cone

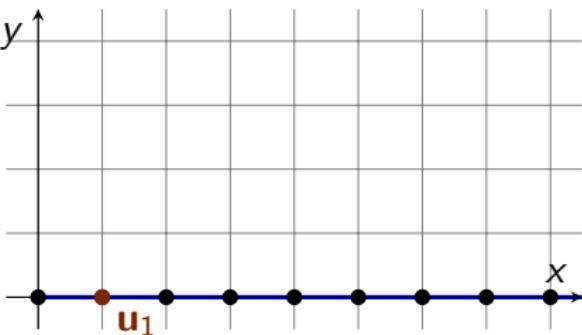
[6, 15, 36]

A d -dimensional cone K with d extremal rays \mathbf{u}_i is **unimodular** if each integer element can be written (uniquely) as a non-negative **integer** combination of the extremal rays

$$K \cap \mathbb{Z}^d = \left\{ \sum_i \lambda_i \mathbf{u}_i : \lambda_i \in \mathbb{Z}_{\geq 0} \right\}$$

$$f(K; \mathbf{x}) = \sum_{\mathbf{m} \in K \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{m}} = \prod_{i=1}^d \frac{1}{1 - \mathbf{x}^{\mathbf{u}_i}}$$

$$\frac{1}{1-x}$$



Generating Function of a Unimodular Cone

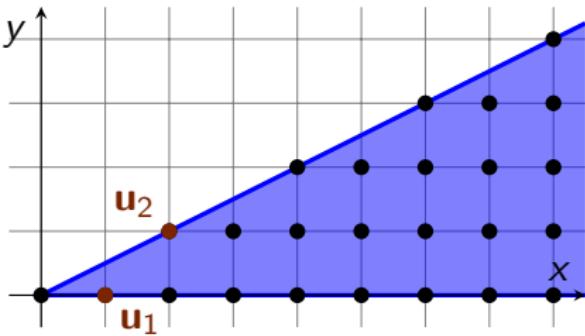
[6, 15, 36]

A d -dimensional cone K with d extremal rays \mathbf{u}_i is **unimodular** if each integer element can be written (uniquely) as a non-negative **integer** combination of the extremal rays

$$K \cap \mathbb{Z}^d = \left\{ \sum_i \lambda_i \mathbf{u}_i : \lambda_i \in \mathbb{Z}_{\geq 0} \right\}$$

$$f(K; \mathbf{x}) = \sum_{\mathbf{m} \in K \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{m}} = \prod_{i=1}^d \frac{1}{1 - \mathbf{x}^{\mathbf{u}_i}}$$

$$\frac{1}{(1-x)(1-x^2y)}$$



Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$



$$\sum_{t \in P(s)} p(s, t)$$

Barvinok's decomposition

construct rational generating function

$$f(s; y)$$

specialization

$$q(s)$$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$

Faulhaber

splintering

$$\sum_{t \in P(s)} p(s, t)$$

Barvinok's decomposition

construct rational generating function

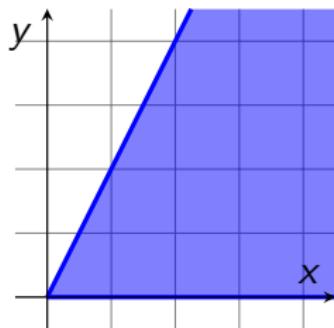
 $f(s; y)$

specialization

 $q(s)$

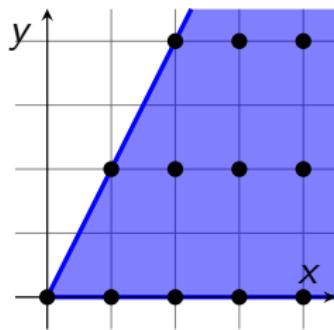
Barvinok's Signed Decomposition

[6, 15, 36, 23]



Barvinok's Signed Decomposition

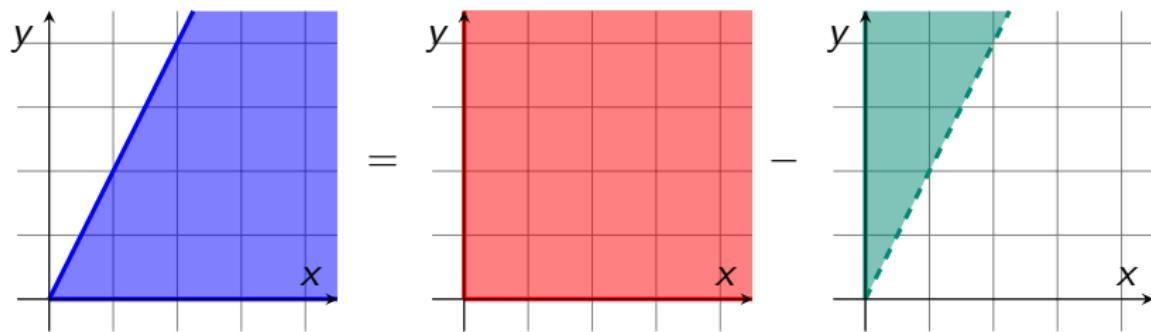
[6, 15, 36, 23]



$$\frac{1}{(1-x)(1-xy^2)}$$

Barvinok's Signed Decomposition

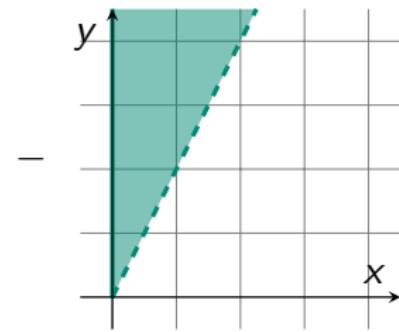
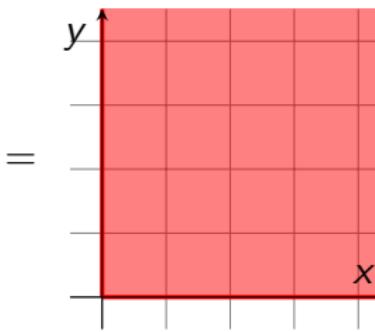
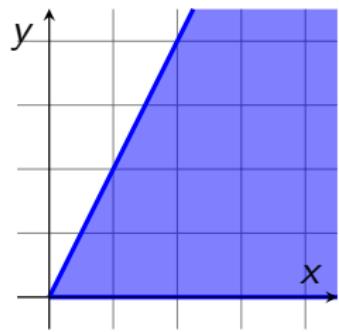
[6, 15, 36, 23]



Barvinok's Signed Decomposition

[6, 15, 36, 23]

$$[K] = \sum_{i \in I} \epsilon_i [K_i] \quad \text{with } \epsilon_i \in \{-1, 1\}$$



Barvinok's Algorithm

[6, 15, 36]

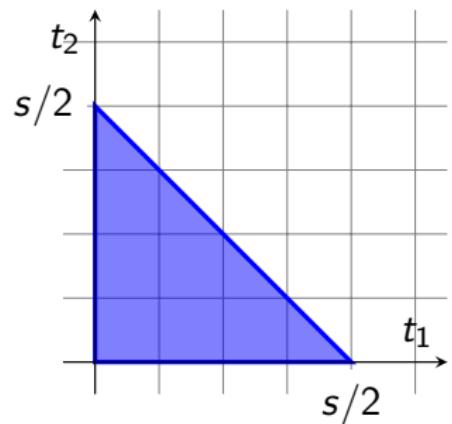
- ① For each vertex $\mathbf{v}_i(\mathbf{s})$ of P

- ① Determine supporting cone $\text{cone}(P, \mathbf{v}_i(\mathbf{s}))$
- ② Let $K = \text{cone}(P, \mathbf{v}_i(\mathbf{s})) - \mathbf{v}_i(\mathbf{s})$
- ③ Decompose K into unimodular cones $\{\epsilon_j, K_j\}$
- ④ For each K_j
 - ① Determine $f(K_j; \mathbf{x})$
 - ⑤ $f(\text{cone}(P, \mathbf{v}_i(\mathbf{s})); \mathbf{x}) = \sum_j \epsilon_j \mathbf{x}^{E(\mathbf{v}_i(\mathbf{p}), K_j)} f(K_j; \mathbf{x})$

- ② For each validity domain D of P

- ① $f(P; \mathbf{x}) = \sum_{\mathbf{v}_i \in D} f(\text{cone}(P, \mathbf{v}_i(\mathbf{s})); \mathbf{x})$
- ② evaluate $f(P; \mathbf{1})$

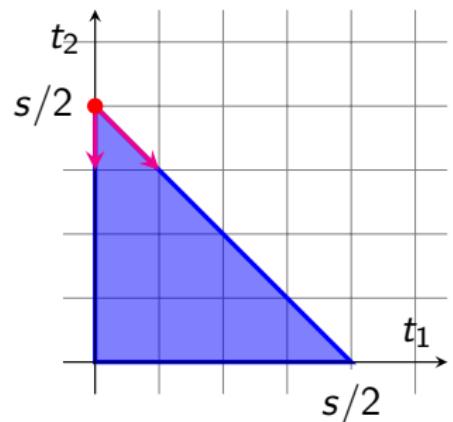
Barvinok's Algorithm — Example



$$\{ t_1, t_2 : t_1, t_2 \geq 0 \wedge 2t_1 + 2t_2 \leq s \}$$

$$\frac{x_2^{\lfloor \frac{s}{2} \rfloor}}{(1 - x_2^{-1})(1 - x_1 x_2^{-1})} + \frac{x_1^{\lfloor \frac{s}{2} \rfloor}}{(1 - x_1^{-1})(1 - x_1^{-1} x_2)} + \frac{1}{(1 - x_1)(1 - x_2)}$$

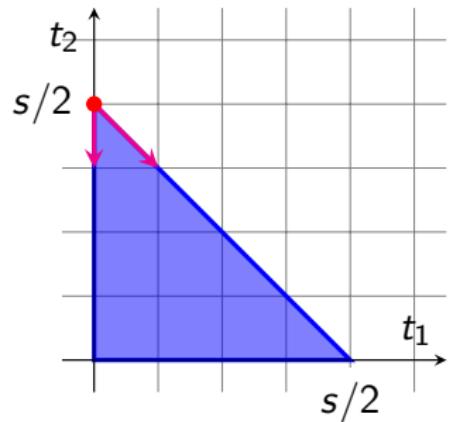
Barvinok's Algorithm — Example



$$\{ t_1, t_2 : t_1, t_2 \geq 0 \wedge 2t_1 + 2t_2 \leq s \}$$

$$\frac{x_1^0 x_2^{\lfloor \frac{s}{2} \rfloor}}{(1 - x_1^0 x_2^{-1})(1 - x_1^1 x_2^{-1})} + \frac{x_1^{\lfloor \frac{s}{2} \rfloor}}{(1 - x_1^{-1})(1 - x_1^{-1} x_2)} + \frac{1}{(1 - x_1)(1 - x_2)}$$

Barvinok's Algorithm — Example



$$\{ t_1, t_2 : t_1, t_2 \geq 0 \wedge 2t_1 + 2t_2 \leq s \}$$

$$\frac{x_1^0 x_2^{\lfloor \frac{s}{2} \rfloor}}{(1 - x_1^0 x_2^{-1})(1 - x_1^1 x_2^{-1})} + \frac{x_1^{\lfloor \frac{s}{2} \rfloor}}{(1 - x_1^{-1})(1 - x_1^{-1} x_2)} + \frac{1}{(1 - x_1)(1 - x_2)}$$

$$f(T; \mathbf{1}) = 1 + \frac{1}{8}s - \frac{1}{8}s^2 + \left(\frac{5}{4} + \frac{1}{2}s\right) \lfloor \frac{s}{2} \rfloor$$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$

Faulhaber

splintering

$$\sum_{t \in P(s)} p(s, t)$$

Barvinok's decomposition

construct rational generating function

 $f(s; y)$

specialization

 $q(s)$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$

Faulhaber

splintering

$$\sum_{t \in P(s)} p(s, t)$$

Barvinok's decomposition

construct rational generating function

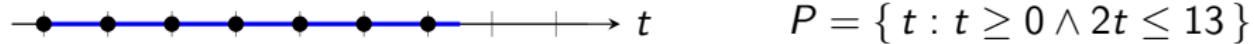
 $f(s; y)$

specialization

 $q(s)$

Specialization

[6, 15, 36]

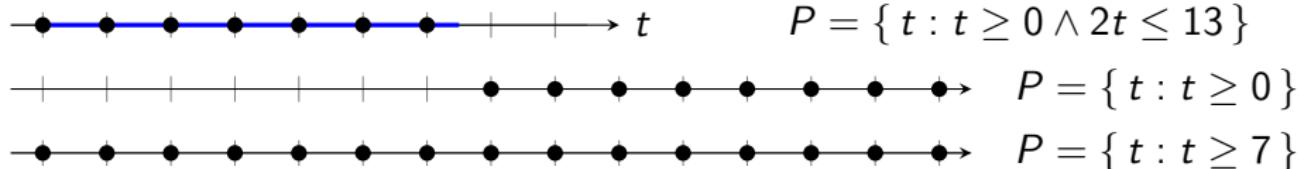


Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P

Specialization

[6, 15, 36]



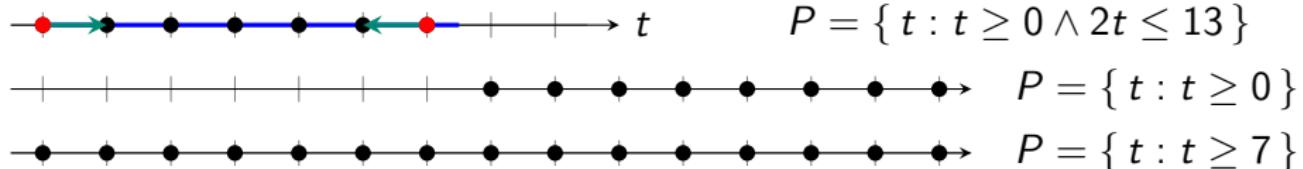
Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P

$$f(P; x) = \frac{x^0}{1-x} - \frac{x^7}{1-x}$$

Specialization

[6, 15, 36]



Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P

$$f(P; x) = \frac{x^0}{1-x} - \frac{x^7}{1-x} = \frac{x^0}{1-\textcolor{teal}{x}} + \frac{\textcolor{red}{x^6}}{1-\textcolor{teal}{x^{-1}}}$$

Specialization

[6, 15, 36]



$$P = \{ t : t \geq 0 \wedge 2t \leq 13 \}$$

Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P

$$f(P; x) = \frac{x^0}{1-x} - \frac{x^7}{1-x} = \frac{x^0}{1-x} + \frac{x^6}{1-x^{-1}}$$

- Sum over all points

Specialization

[6, 15, 36]



$$P = \{ t : t \geq 0 \wedge 2t \leq 13 \}$$

Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P

$$f(P; x) = \frac{x^0}{1-x} - \frac{x^7}{1-x} = \frac{x^0}{1-x} + \frac{x^6}{1-x^{-1}} = \frac{x^0}{1-x} - \frac{(1-(1-x))^7}{1-x}$$

- Sum over all points \Rightarrow Laurent expansion

$$\begin{aligned} f(P; 1) = & -1(x-1)^{-1} + 0(x-1)^0 + 0(x-1)^1 + \dots \\ & + 1(x-1)^{-1} + 7(x-1)^0 + 21(x-1)^1 + \dots \end{aligned}$$

Specialization

[6, 15, 36]



$$P = \{ t : t \geq 0 \wedge 2t \leq 13 \}$$

Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P

$$f(P; x) = \frac{x^0}{1-x} - \frac{x^7}{1-x} = \frac{x^0}{1-x} + \frac{x^6}{1-x^{-1}} = \frac{x^0}{1-x} - \frac{(1-(1-x))^7}{1-x}$$

- Sum over all points \Rightarrow Laurent expansion

$$\begin{aligned} f(P; 1) &= -1(x-1)^{-1} + 0(x-1)^0 + 0(x-1)^1 + \dots \\ &\quad + 1(x-1)^{-1} + 7(x-1)^0 + 21(x-1)^1 + \dots \\ &= 7 \end{aligned}$$

Specialization

[6, 15, 36]



$$P = \{ t : t \geq 0 \wedge 2t \leq 13 \}$$

Counting the integer points in polytope P

- For each point in \mathbb{Z} count the number of occurrences in P

$$f(P; x) = \frac{x^0}{1-x} - \frac{x^7}{1-x} = \frac{x^0}{1-x} + \frac{x^6}{1-x^{-1}} = \frac{x^0}{1-x} - \frac{(1-(1-x))^7}{1-x}$$

- Sum over all points \Rightarrow Laurent expansion

$$\begin{aligned} f(P; 1) &= -1(x-1)^{-1} + 0(x-1)^0 + 0(x-1)^1 + \dots \\ &\quad + 1(x-1)^{-1} + 7(x-1)^0 + 21(x-1)^1 + \dots \\ &= 7 \end{aligned}$$

In higher dimensions: compute residue

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$

Faulhaber

$$\sum_{t \in P(s)} p(s, t)$$

splintering

Barvinok's decomposition

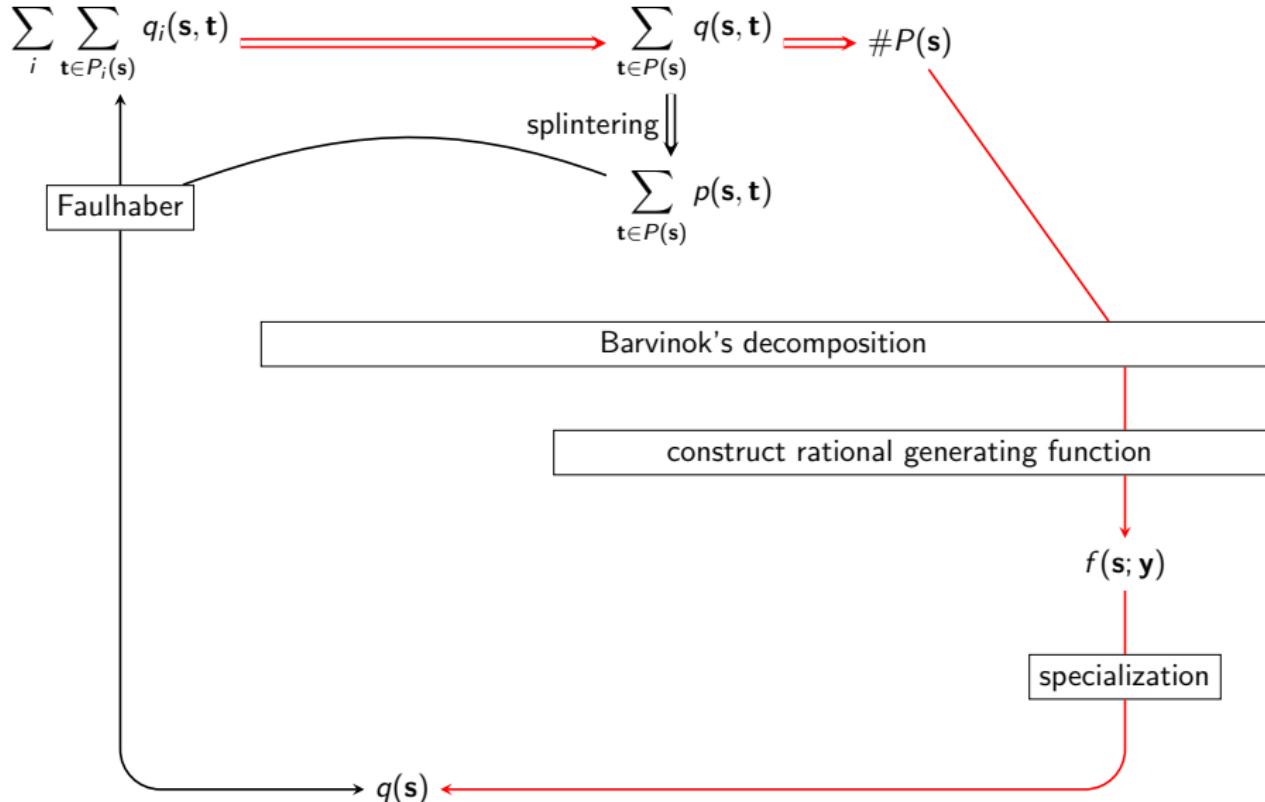
construct rational generating function

 $f(s; y)$

specialization

 $q(s)$

Weighted Counting Overview: Reduction to Unweighted^[32, 29]



Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$

Faulhaber

splintering \Downarrow

\downarrow extra dim

$$\sum_{t \in P(s)} p(s, t)$$

Barvinok's decomposition

construct rational generating function

$f(s; y)$

specialization

$q(s)$

From Quasi Polynomial to Polynomial

$$\text{Quasi monomial } \prod_i \left[\frac{a_{i0}t + \sum_j a_{ij}s_j}{D_i} \right]^{m_i} \quad j' \left[\frac{j' - i'}{4} \right]$$

- Splintering (on single dimension t)

⇒ Split domain into D parts (t replaced by t') $j' \left(-t' + \left[\frac{j' - k}{4} \right] \right)$

$$t = Dt' + k \quad \text{for } 0 \leq k < D, D = \text{lcm}_i D_i$$

$$i' = 4t' + k \quad \text{for } 0 \leq k < 4$$

From Quasi Polynomial to Polynomial

$$\text{Quasi monomial } \prod_i \left[\frac{a_{i0}t + \sum_j a_{ij}s_j}{D_i} \right]^{m_i} \quad j' \left[\frac{j' - i'}{4} \right]$$

- Splintering (on single dimension t)

$$\Rightarrow \text{Split domain into } D \text{ parts } (t \text{ replaced by } t') \quad j' \left(-t' + \left[\frac{j' - k}{4} \right] \right)$$

$$t = Dt' + k \quad \text{for } 0 \leq k < D, D = \text{lcm}_i D_i$$

$$i' = 4t' + k \quad \text{for } 0 \leq k < 4$$

- Extra dimensions

$$\Rightarrow \text{Add extra variable } t'_i \text{ for each distinct floor expression} \quad j't'$$

$$D_i t'_i \leq a_{i0}t + \sum_j a_{ij}s_j \leq D_i t'_i + D_i - 1$$

$$4t' \leq j' - i' \leq 4t' + 3$$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$

Faulhaber

splintering \Downarrow

\downarrow extra dim

$$\sum_{t \in P(s)} p(s, t)$$

Barvinok's decomposition

construct rational generating function

$f(s; y)$

specialization

$q(s)$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$

Faulhaber

$$\sum_{t \in P(s)} p(s, t)$$

splintering \Downarrow

extra dim

Barvinok's decomposition

construct rational generating function

$$f(s; y), p(s, t) \xrightarrow{D} f(s; y)$$

specialization

$$q(s) \leftarrow$$

Weighted Counting through Derivation

Basic idea:

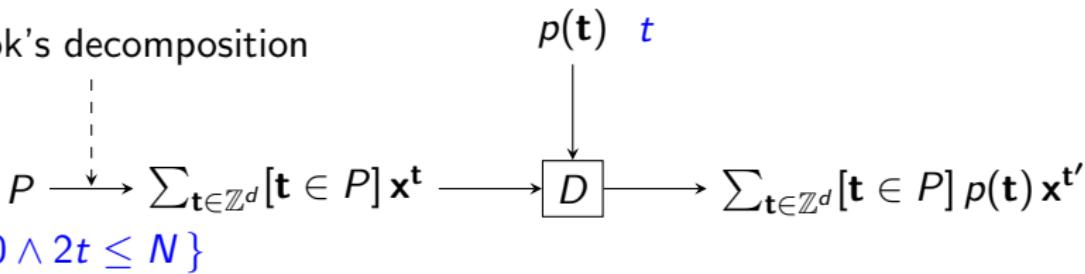
$$D x^m = m x^{m-1}$$

Weighted Counting through Derivation

Basic idea:

$$D x^m = m x^{m-1}$$

Barvinok's decomposition

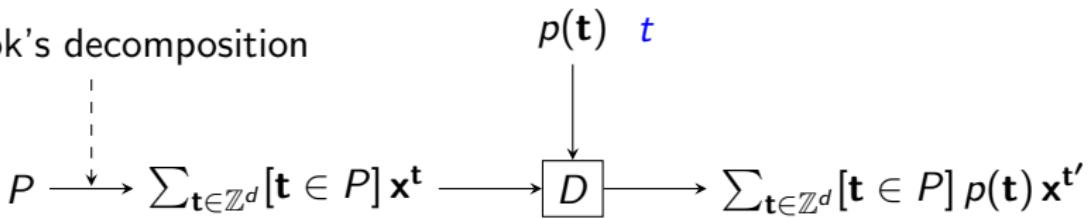


Weighted Counting through Derivation

Basic idea:

$$D x^m = m x^{m-1}$$

Barvinok's decomposition



$$\{ t : t \geq 0 \wedge 2t \leq N \}$$

$$f(x) = \sum_t [t \in P] x^t \quad \frac{x^0}{1-x} - \frac{x^{\lfloor \frac{N}{2} \rfloor + 1}}{1-x}$$

$$f'(x) = \sum_t [t \in P] t x^{t-1} \quad \frac{1}{(1-x)^2} - \frac{(\lfloor \frac{N}{2} \rfloor + 1) x^{\lfloor \frac{N}{2} \rfloor}}{1-x} - \frac{x^{\lfloor \frac{N}{2} \rfloor + 1}}{(1-x)^2}$$

$$f'(1) = \sum_t [t \in P] t \quad \frac{1}{2} \left\lfloor \frac{N}{2} \right\rfloor^2 + \frac{1}{2} \left\lfloor \frac{N}{2} \right\rfloor$$

Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{t \in P_i(s)} q_i(s, t) \longrightarrow \sum_{t \in P(s)} q(s, t) \longrightarrow \#P(s)$$

Faulhaber

$$\sum_{t \in P(s)} p(s, t)$$

splintering \Downarrow

extra dim

Barvinok's decomposition

construct rational generating function

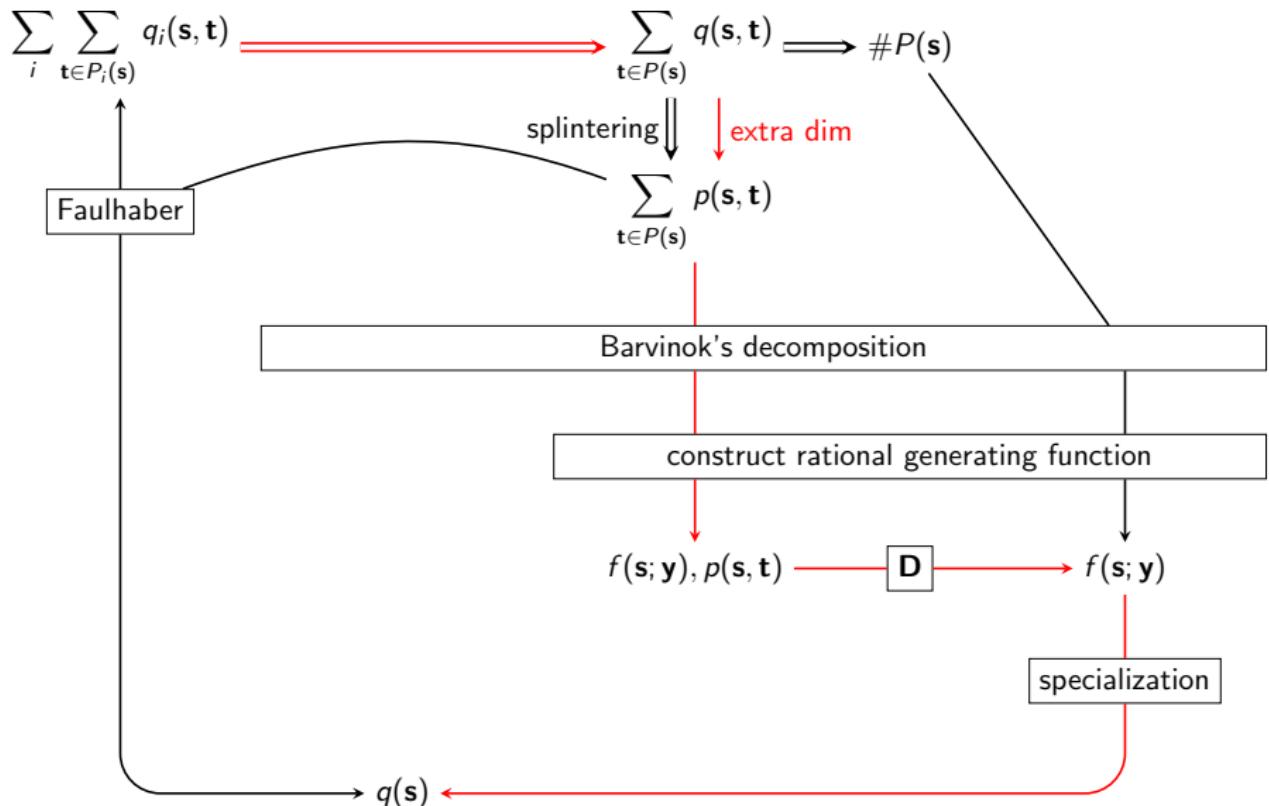
$$f(s; y), p(s, t) \xrightarrow{D} f(s; y)$$

specialization

$$q(s) \leftarrow$$

Weighted Counting Overview: Derivation

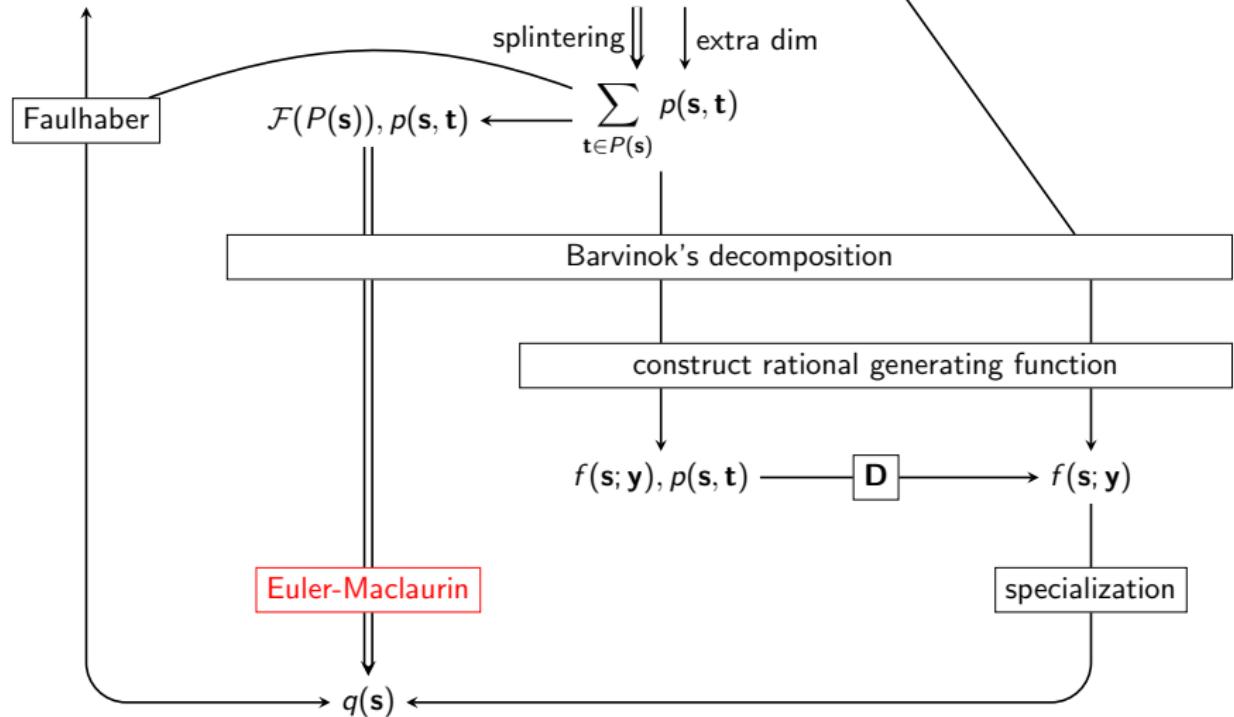
[32, 29]



Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{\mathbf{t} \in P_i(\mathbf{s})} q_i(\mathbf{s}, \mathbf{t}) \longrightarrow \sum_{\mathbf{t} \in P(\mathbf{s})} q(\mathbf{s}, \mathbf{t}) \longrightarrow \#P(\mathbf{s})$$



Weighted Counting using Local Euler-Maclaurin Formulas [9]

Basic idea:

$$\sum_{\mathbf{t} \in P(\mathbf{s}) \cap \Lambda} p(\mathbf{s})(\mathbf{t}) = \sum_{F(\mathbf{s}) \in \mathcal{F}(P(\mathbf{s}))} \int_{F(\mathbf{s})} D_{P(\mathbf{s}), F(\mathbf{s})} \cdot p(\mathbf{s})$$

Weighted Counting using Local Euler-Maclaurin Formulas [9]

Basic idea:

$$\sum_{\mathbf{t} \in P(\mathbf{s}) \cap \Lambda} p(\mathbf{s})(\mathbf{t}) = \sum_{F(\mathbf{s}) \in \mathcal{F}(P(\mathbf{s}))} \int_{F(\mathbf{s})} D_{P(\mathbf{s}), F(\mathbf{s})} \cdot p(\mathbf{s})$$

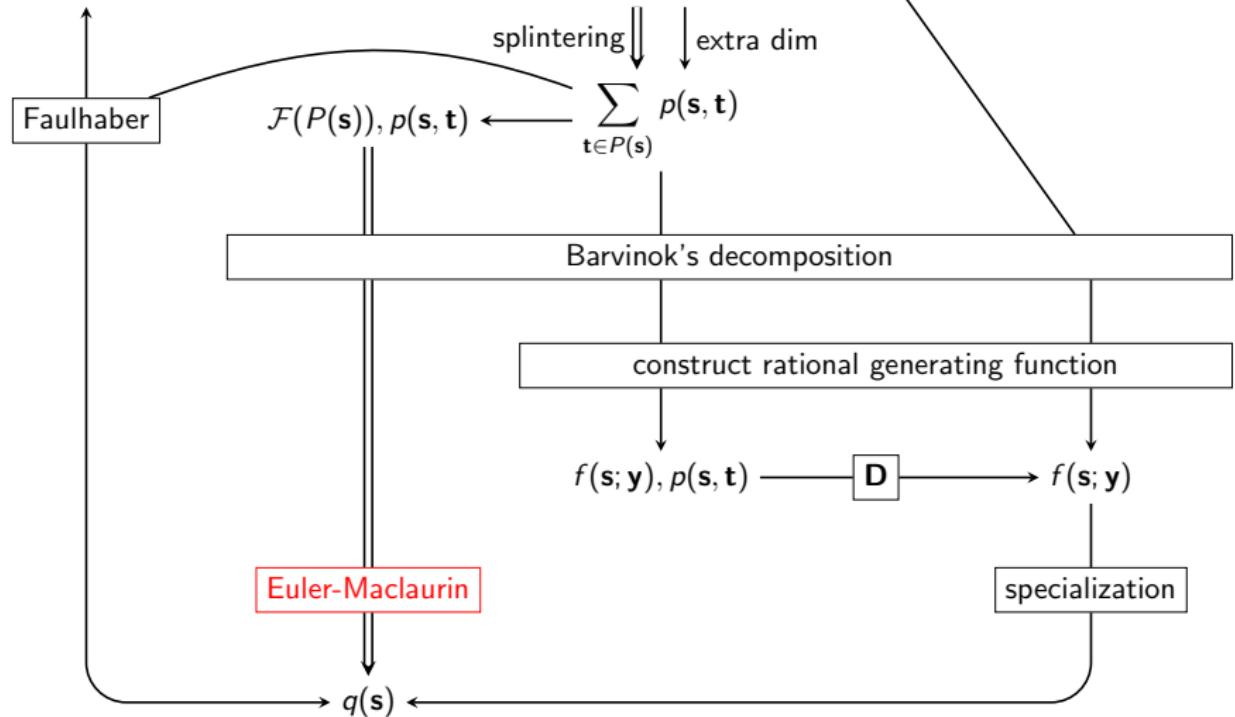
One-dimensional polytope:

$$\begin{aligned} \sum_{t \in P(\mathbf{s}) \cap \mathbb{Z}} p(\mathbf{s})(t) &= \int_{v_1(\mathbf{s})}^{v_2(\mathbf{s})} p(\mathbf{s})(t) dt \\ &\quad - \sum_{n=0}^{\infty} \frac{b(n+1, \{-v_1(\mathbf{s})\})}{(n+1)!} (D^n p(\mathbf{s}))(v_1(\mathbf{s})) \\ &\quad - \sum_{n=0}^{\infty} (-1)^n \frac{b(n+1, \{v_2(\mathbf{s})\})}{(n+1)!} (D^n p(\mathbf{s}))(v_2(\mathbf{s})) \end{aligned}$$

Weighted Counting Overview

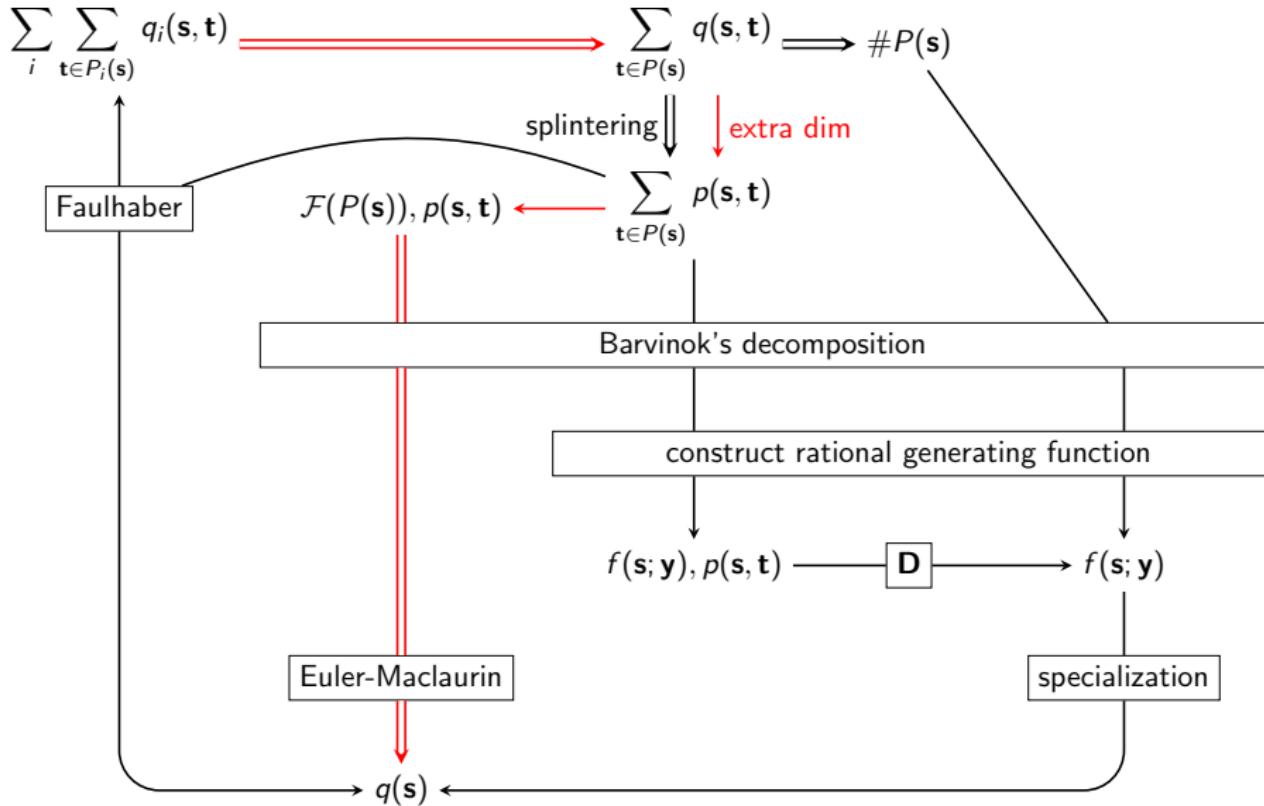
[32, 29]

$$\sum_i \sum_{\mathbf{t} \in P_i(\mathbf{s})} q_i(\mathbf{s}, \mathbf{t}) \longrightarrow \sum_{\mathbf{t} \in P(\mathbf{s})} q(\mathbf{s}, \mathbf{t}) \longrightarrow \#P(\mathbf{s})$$



Weighted Counting Overview: Local Euler-Maclaurin

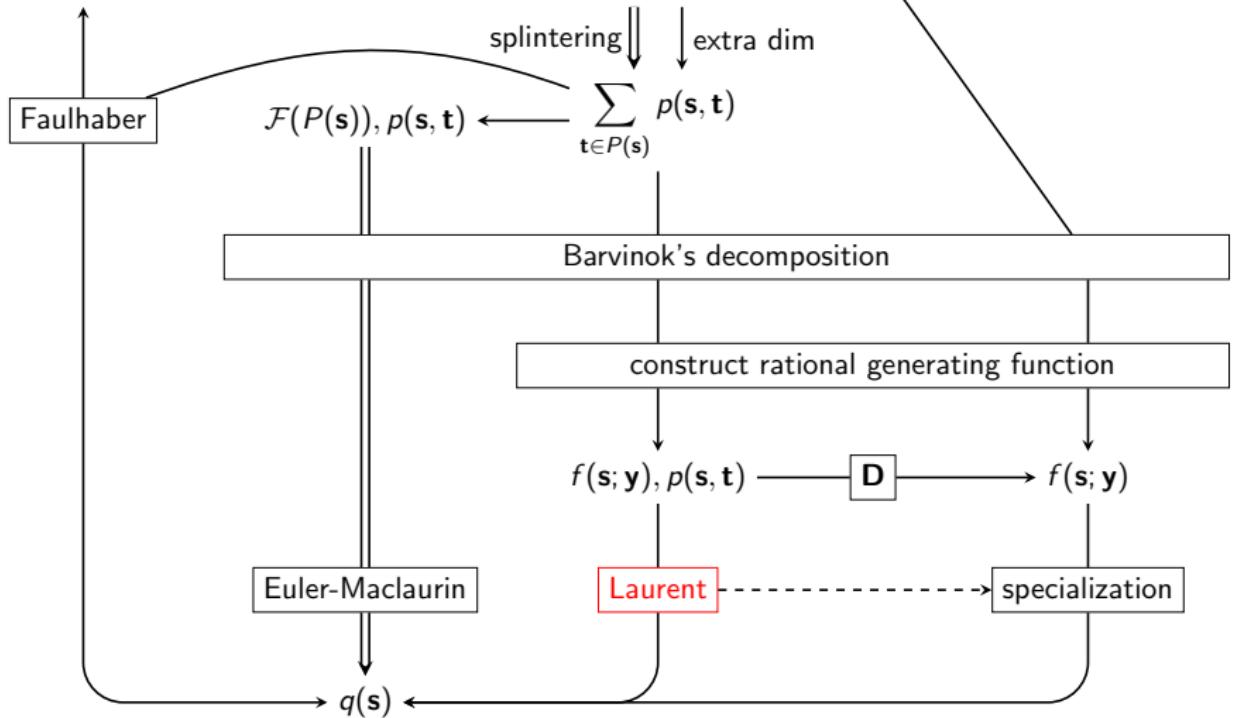
[32, 29]



Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{\mathbf{t} \in P_i(\mathbf{s})} q_i(\mathbf{s}, \mathbf{t}) \longrightarrow \sum_{\mathbf{t} \in P(\mathbf{s})} q(\mathbf{s}, \mathbf{t}) \longrightarrow \#P(\mathbf{s})$$



Weighted Counting using Laurent Series Expansion

Basic idea:

$$f(\mathbf{e}^{\mathbf{y}}) = \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} e^{\langle \mathbf{t}, \mathbf{y} \rangle} = \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \sum_{\mathbf{n} \geq \mathbf{0}} \frac{\mathbf{t}^{\mathbf{n}} \mathbf{y}^{\mathbf{n}}}{\mathbf{n}!} = \sum_{\mathbf{n} \geq \mathbf{0}} \left(\sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \mathbf{t}^{\mathbf{n}} \right) \frac{\mathbf{y}^{\mathbf{n}}}{\mathbf{n}!}$$

$\Rightarrow \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \mathbf{t}^{\mathbf{n}}$ is $\mathbf{n}!$ times coefficient of $\mathbf{y}^{\mathbf{n}}$ in $f(\mathbf{e}^{\mathbf{y}})$

Weighted Counting using Laurent Series Expansion

Basic idea:

$$f(\mathbf{e}^{\mathbf{y}}) = \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} e^{\langle \mathbf{t}, \mathbf{y} \rangle} = \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \sum_{\mathbf{n} \geq \mathbf{0}} \frac{\mathbf{t}^{\mathbf{n}} \mathbf{y}^{\mathbf{n}}}{\mathbf{n}!} = \sum_{\mathbf{n} \geq \mathbf{0}} \left(\sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \mathbf{t}^{\mathbf{n}} \right) \frac{\mathbf{y}^{\mathbf{n}}}{\mathbf{n}!}$$

$\Rightarrow \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \mathbf{t}^{\mathbf{n}}$ is $\mathbf{n}!$ times coefficient of $\mathbf{y}^{\mathbf{n}}$ in $f(\mathbf{e}^{\mathbf{y}})$

$$f(\mathbf{x}) = \sum_{i \in I} \alpha_i \frac{\sum_{k=1}^r \mathbf{x}^{\mathbf{w}_{ik}(\mathbf{p})}}{\prod_{j=1}^{k_i} (1 - \mathbf{x}^{\mathbf{b}_{ij}})}$$

Weighted Counting using Laurent Series Expansion

[5]

Basic idea:

$$f(\mathbf{e}^{\mathbf{y}}) = \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} e^{\langle \mathbf{t}, \mathbf{y} \rangle} = \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \sum_{\mathbf{n} \geq \mathbf{0}} \frac{\mathbf{t}^{\mathbf{n}} \mathbf{y}^{\mathbf{n}}}{\mathbf{n}!} = \sum_{\mathbf{n} \geq \mathbf{0}} \left(\sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \mathbf{t}^{\mathbf{n}} \right) \frac{\mathbf{y}^{\mathbf{n}}}{\mathbf{n}!}$$

$\Rightarrow \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \mathbf{t}^{\mathbf{n}}$ is $\mathbf{n}!$ times coefficient of $\mathbf{y}^{\mathbf{n}}$ in $f(\mathbf{e}^{\mathbf{y}})$

$$\begin{aligned} f(\mathbf{x}) &= \sum_{i \in I} \alpha_i \frac{\sum_{k=1}^r \mathbf{x}^{\mathbf{w}_{ik}(\mathbf{p})}}{\prod_{j=1}^{k_i} (1 - \mathbf{x}^{\mathbf{b}_{ij}})} \\ f_i(\mathbf{e}^{\mathbf{y}}) &= \frac{e^{\sum_{j=1}^d s_j(\mathbf{p}) \langle \mathbf{b}_j, \mathbf{y} \rangle}}{\prod_{j=1}^d (1 - e^{\langle \mathbf{b}_j, \mathbf{y} \rangle})} \\ &= \prod_{j=1}^d \left(-\frac{1}{\langle \mathbf{b}_j, \mathbf{y} \rangle} + \left(\frac{1}{\langle \mathbf{b}_j, \mathbf{y} \rangle} + \frac{e^{\sum_{j=1}^d s_j(\mathbf{p}) \langle \mathbf{b}_j, \mathbf{y} \rangle}}{1 - e^{\langle \mathbf{b}_j, \mathbf{y} \rangle}} \right) \right) \\ &= \prod_{j=1}^d \left(-\frac{1}{\langle \mathbf{b}_j, \mathbf{y} \rangle} - \sum_{k=0}^{\infty} \frac{b(k+1, s_j(\mathbf{p}))}{(k+1)!} \langle \mathbf{b}_j, \mathbf{y} \rangle^k \right) \end{aligned}$$

Weighted Counting using Laurent Series Expansion

Basic idea:

$$f(\mathbf{e}^{\mathbf{y}}) = \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} e^{\langle \mathbf{t}, \mathbf{y} \rangle} = \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \sum_{\mathbf{n} \geq \mathbf{0}} \frac{\mathbf{t}^{\mathbf{n}} \mathbf{y}^{\mathbf{n}}}{\mathbf{n}!} = \sum_{\mathbf{n} \geq \mathbf{0}} \left(\sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \mathbf{t}^{\mathbf{n}} \right) \frac{\mathbf{y}^{\mathbf{n}}}{\mathbf{n}!}$$

$\Rightarrow \sum_{\mathbf{t} \in P \cap \mathbb{Z}^d} \mathbf{t}^{\mathbf{n}}$ is $\mathbf{n}!$ times coefficient of $\mathbf{y}^{\mathbf{n}}$ in $f(\mathbf{e}^{\mathbf{y}})$

$$f(\mathbf{x}) = \sum_{i \in I} \alpha_i \frac{\sum_{k=1}^r \mathbf{x}^{\mathbf{w}_{ik}(\mathbf{p})}}{\prod_{j=1}^{k_i} (1 - \mathbf{x}^{\mathbf{b}_{ij}})}$$

$$f_i(\mathbf{e}^{\mathbf{y}}) = \frac{e^{\sum_{j=1}^d s_j(\mathbf{p}) \langle \mathbf{b}_j, \mathbf{y} \rangle}}{\prod_{j=1}^d (1 - e^{\langle \mathbf{b}_j, \mathbf{y} \rangle})}$$

Bernoulli polynomials

$$= \prod_{j=1}^d \left(-\frac{1}{\langle \mathbf{b}_j, \mathbf{y} \rangle} + \left(\frac{1}{\langle \mathbf{b}_j, \mathbf{y} \rangle} + \frac{e^{s_j(\mathbf{p}) \langle \mathbf{b}_j, \mathbf{y} \rangle}}{1 - e^{\langle \mathbf{b}_j, \mathbf{y} \rangle}} \right) \right)$$

$$= \prod_{j=1}^d \left(-\frac{1}{\langle \mathbf{b}_j, \mathbf{y} \rangle} - \sum_{k=0}^{\infty} \frac{b(k+1, s_j(\mathbf{p}))}{(k+1)!} \langle \mathbf{b}_j, \mathbf{y} \rangle^k \right)$$

Weighted Counting using Laurent Series Expansion

[5]

Example

$$P = \{ t : t \geq 0 \wedge 2t \leq N \}$$

$$p(t) = t$$

$$f(x) = \sum_t [t \in P] x^t \quad \frac{x^0}{1-x} - \frac{x^{\lfloor \frac{N}{2} \rfloor + 1}}{1-x}$$

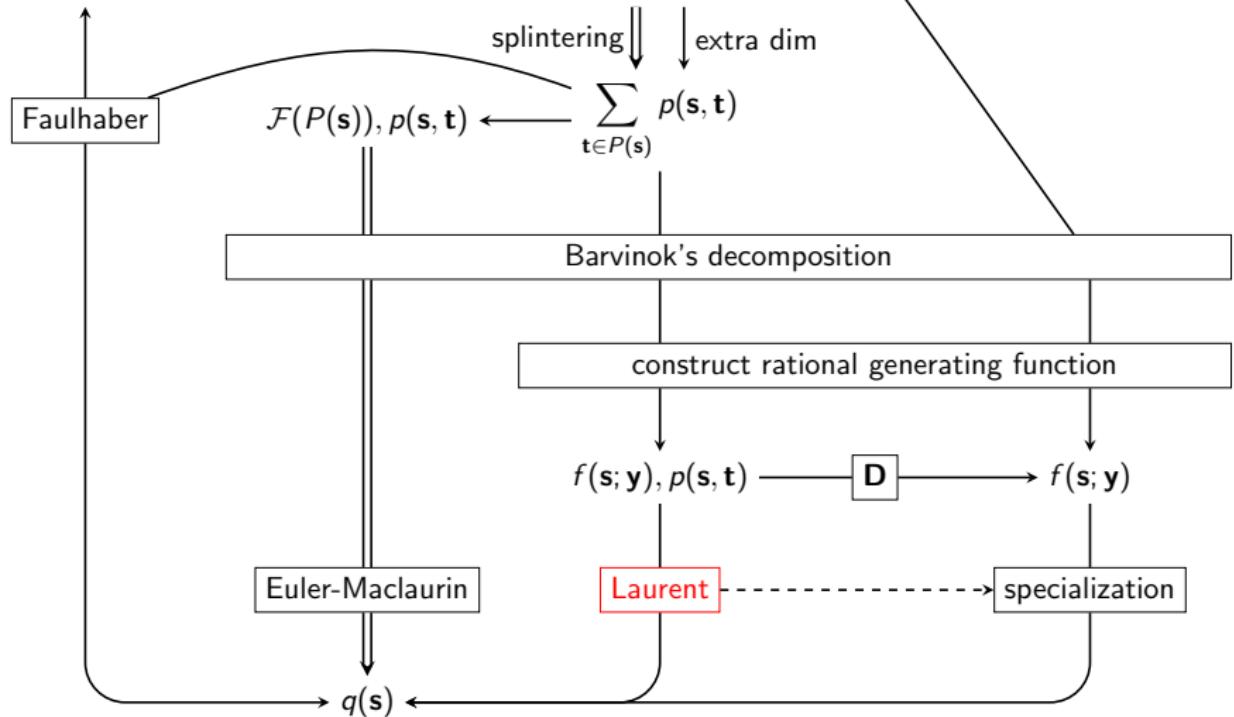
$$f(e^y) = \sum_t [t \in P] e^{ty} \quad \frac{1}{1-e^y} - \frac{e^{(\lfloor \frac{N}{2} \rfloor + 1)y}}{1-e^y}$$

$$\begin{aligned} y^1\text{-term} &= \sum_t [t \in P] t y^1 \\ &= \frac{1}{y} \left(-\frac{b(2, 0)}{2} + \frac{b(2, \lfloor \frac{N}{2} \rfloor + 1)}{2} \right) y^2 \\ &\quad \left(\frac{1}{2} \left[\frac{N}{2} \right]^2 + \frac{1}{2} \left[\frac{N}{2} \right] \right) y \end{aligned}$$

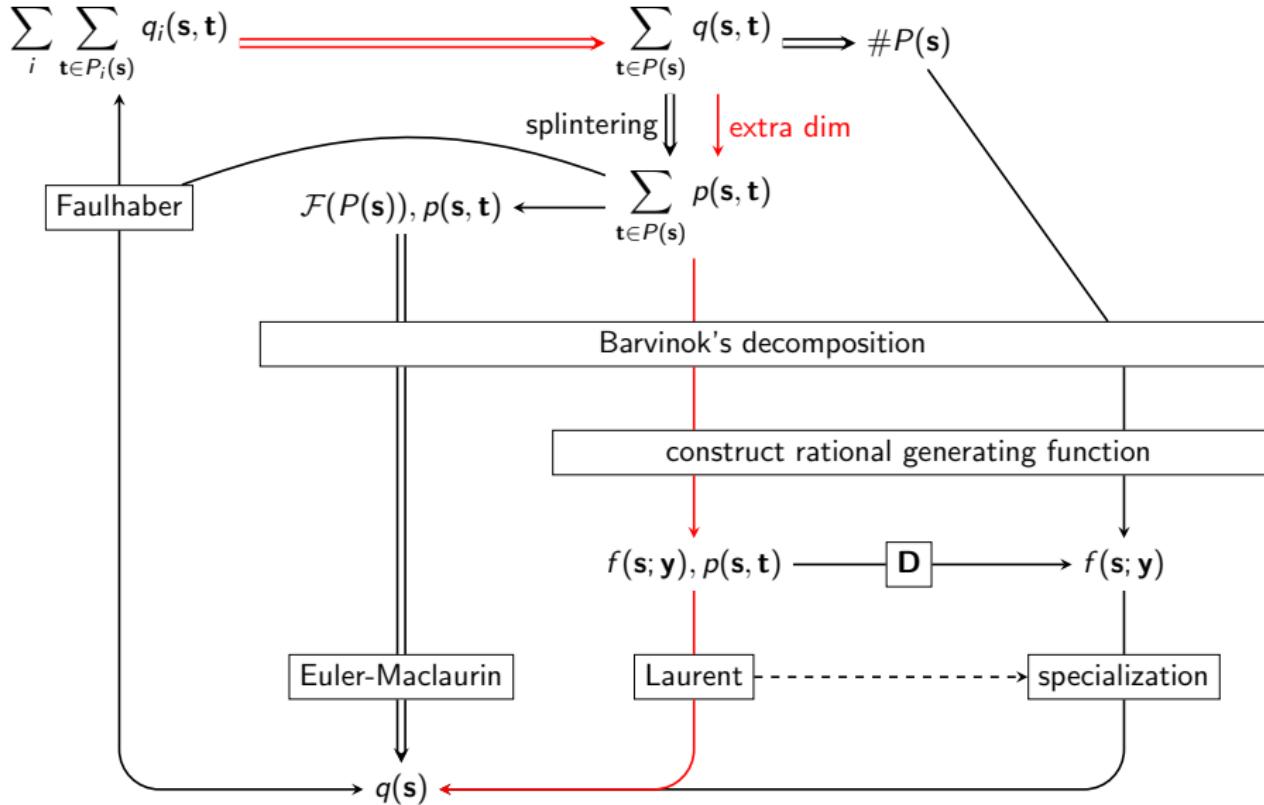
Weighted Counting Overview

[32, 29]

$$\sum_i \sum_{\mathbf{t} \in P_i(\mathbf{s})} q_i(\mathbf{s}, \mathbf{t}) \longrightarrow \sum_{\mathbf{t} \in P(\mathbf{s})} q(\mathbf{s}, \mathbf{t}) \longrightarrow \#P(\mathbf{s})$$

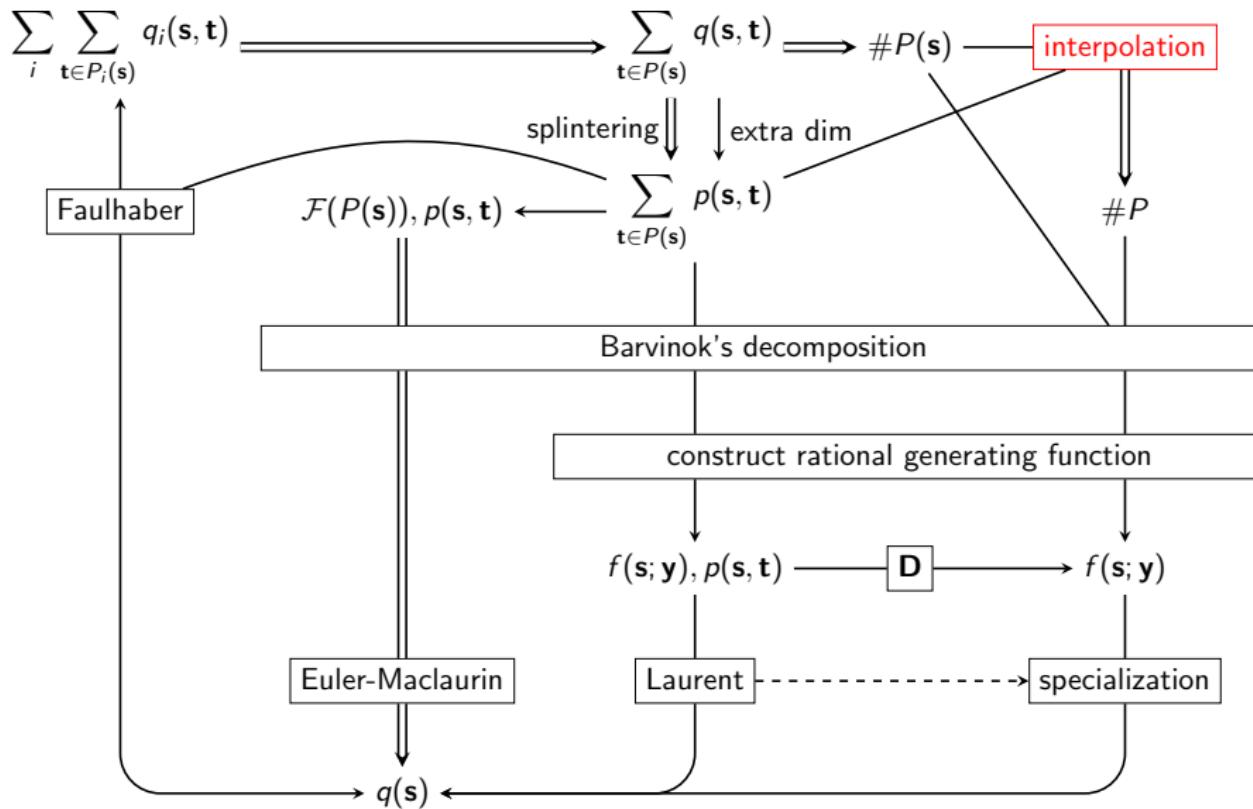


Weighted Counting Overview: Laurent Series Expansion [32, 29]



Weighted Counting Overview

[32, 29]



Interpolation Example

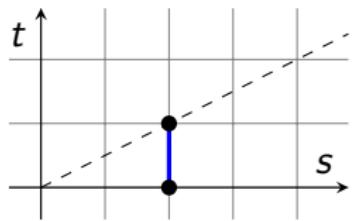
[12]

$$P = \{ t : t \geq 0 \wedge 2t \leq s \} \quad V = \left\{ 0, \frac{s}{2} \right\}$$

Interpolation Example

[12]

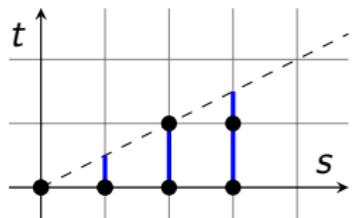
$$P = \{ t : t \geq 0 \wedge 2t \leq s \} \quad V = \left\{ 0, \frac{s}{2} \right\}$$



Interpolation Example

[12]

$$P = \{ t : t \geq 0 \wedge 2t \leq s \} \quad V = \left\{ 0, \frac{s}{2} \right\}$$



$$\begin{aligned}\text{card } P &= [a, b]_s s + [c, d]_s \\ &= \begin{cases} as + c & s \text{ even} \\ bs + d & s \text{ odd} \end{cases}\end{aligned}$$

$$c = 1$$

$$b + d = 1$$

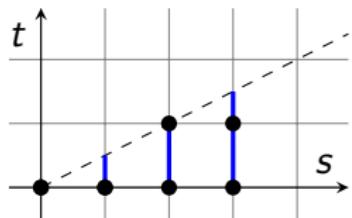
$$2a + c = 2$$

$$3b + d = 2$$

Interpolation Example

[12]

$$P = \{ t : t \geq 0 \wedge 2t \leq s \} \quad V = \left\{ 0, \frac{s}{2} \right\}$$

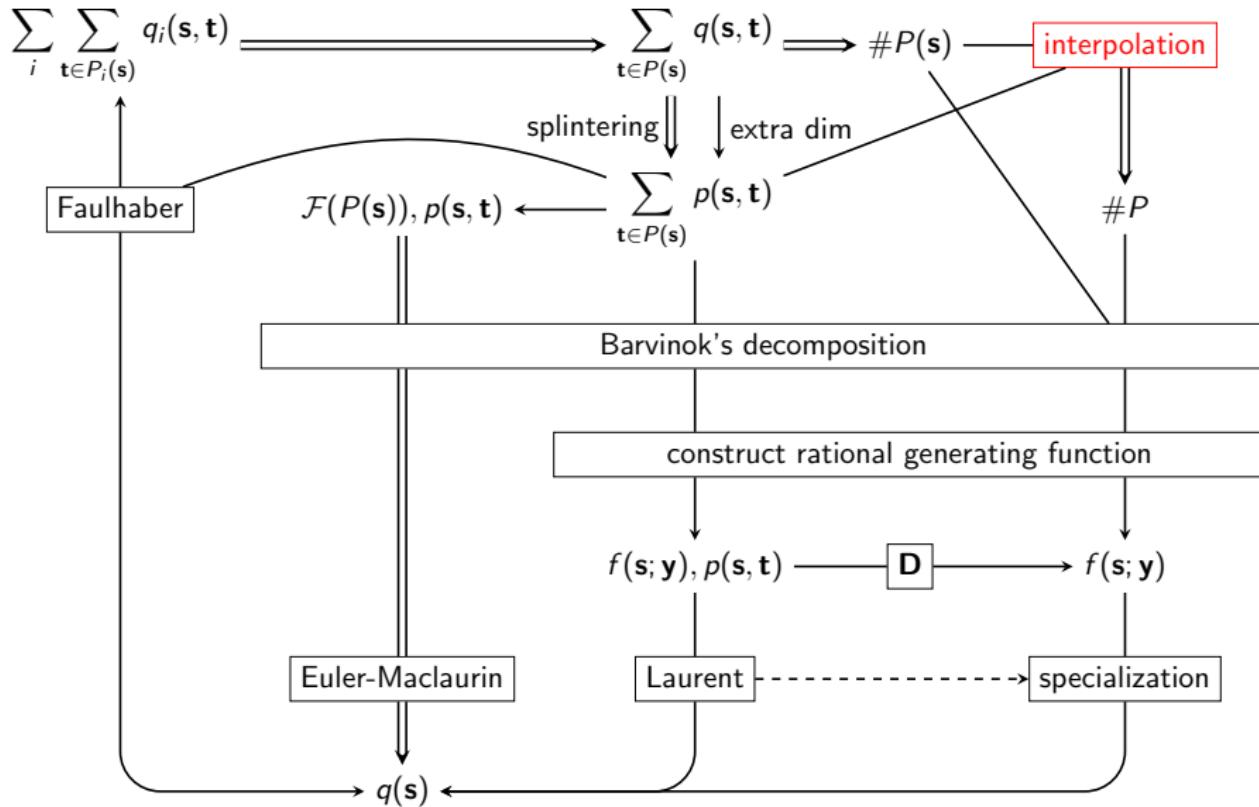


$$\begin{aligned}\text{card } P &= [a, b]_s s + [c, d]_s \\ &= \begin{cases} as + c & s \text{ even} \\ bs + d & s \text{ odd} \end{cases}\end{aligned}$$

$$\begin{array}{rcl} c & = & 1 \\ b+d & = & 1 \\ 2a+c & = & 2 \\ 3b+d & = & 2 \end{array} \Rightarrow \begin{array}{rcl} a & = & 1/2 \\ b & = & 1/2 \\ c & = & 1 \\ d & = & 1/2 \end{array}$$

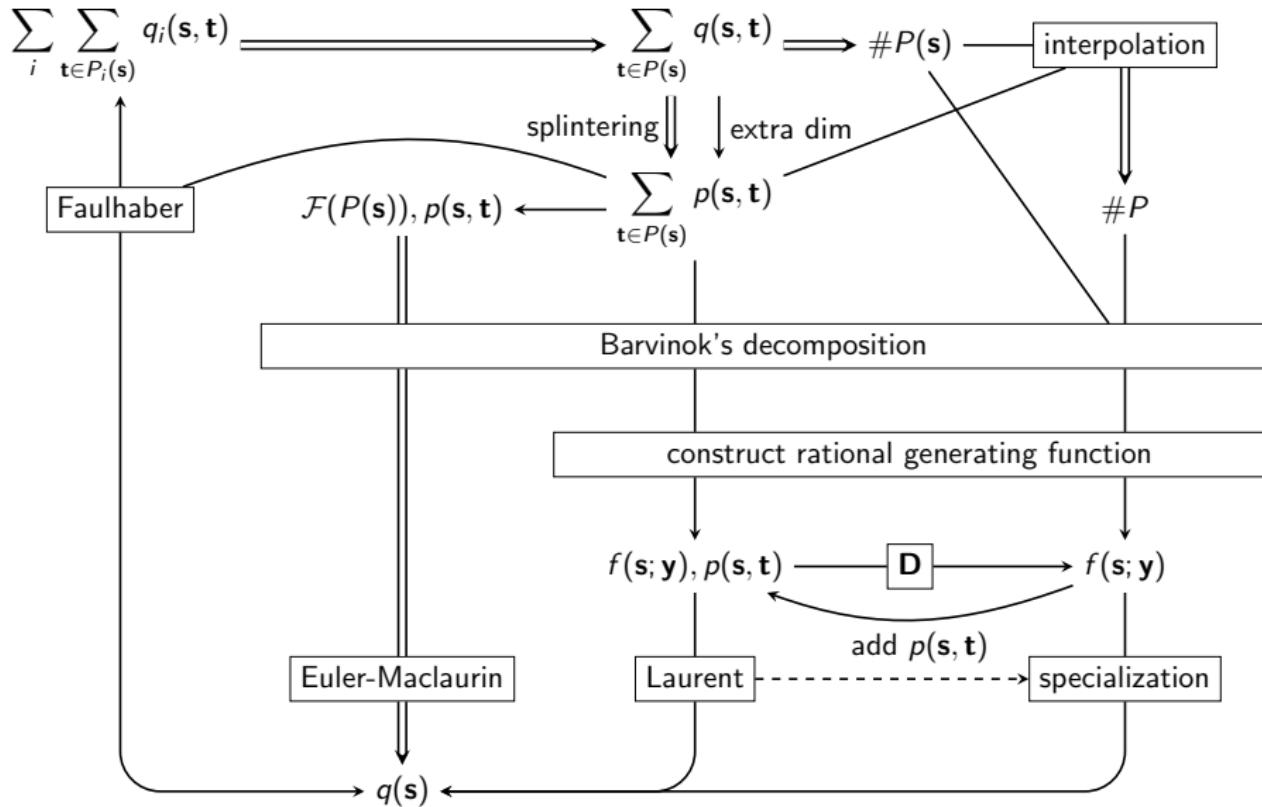
Weighted Counting Overview

[32, 29]



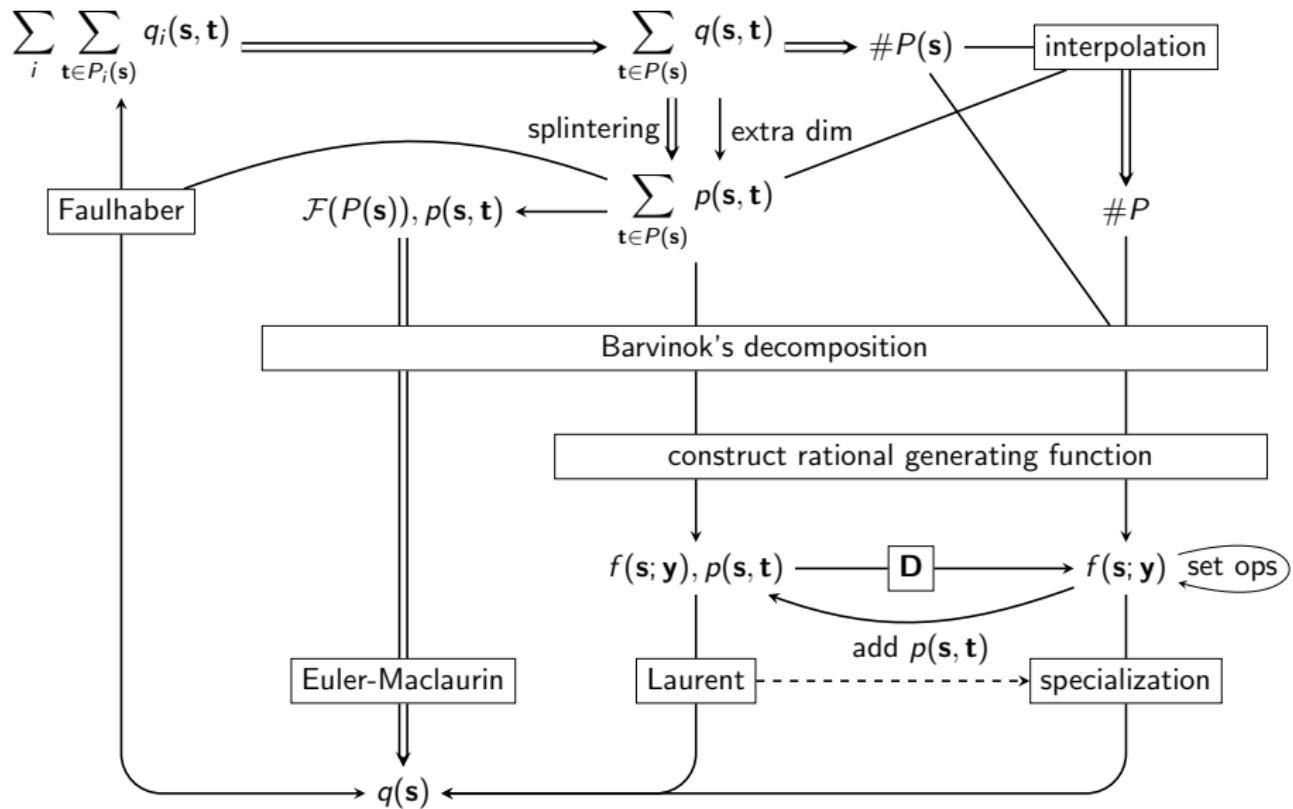
Weighted Counting Overview

[32, 29]



Weighted Counting Overview

[32, 29]



Example Set Operation on Generating Functions: Hadamard Product

[7]

$$f(\mathbf{x}) = \sum_{\mathbf{s} \in \mathbb{Z}^d} a(\mathbf{s}) \mathbf{x}^{\mathbf{s}} \quad g(\mathbf{x}) = \sum_{\mathbf{s} \in \mathbb{Z}^d} b(\mathbf{s}) \mathbf{x}^{\mathbf{s}}$$

Hadamard product:

$$f(\mathbf{x}) \star g(\mathbf{x}) = \sum_{\mathbf{s} \in \mathbb{Z}^d} a(\mathbf{s})b(\mathbf{s}) \mathbf{x}^{\mathbf{s}}$$

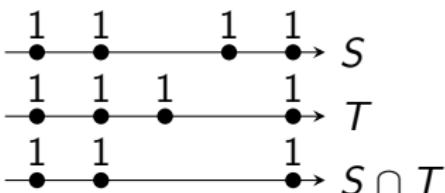
Example Set Operation on Generating Functions: Hadamard Product

[7]

$$f(\mathbf{x}) = \sum_{\mathbf{s} \in \mathbb{Z}^d} a(\mathbf{s}) \mathbf{x}^{\mathbf{s}} \quad g(\mathbf{x}) = \sum_{\mathbf{s} \in \mathbb{Z}^d} b(\mathbf{s}) \mathbf{x}^{\mathbf{s}}$$

Hadamard product:

$$f(\mathbf{x}) * g(\mathbf{x}) = \sum_{\mathbf{s} \in \mathbb{Z}^d} a(\mathbf{s})b(\mathbf{s}) \mathbf{x}^{\mathbf{s}}$$



Corresponds to intersection on sets:

$$\begin{aligned} f(S; \mathbf{x}) * f(T; \mathbf{x}) &= \sum_{\mathbf{s} \in \mathbb{Z}^d} [\mathbf{s} \in S][\mathbf{s} \in T] \mathbf{x}^{\mathbf{s}} \\ &= \sum_{\mathbf{s} \in \mathbb{Z}^d} [\mathbf{s} \in S \cap T] \mathbf{x}^{\mathbf{s}} \end{aligned}$$

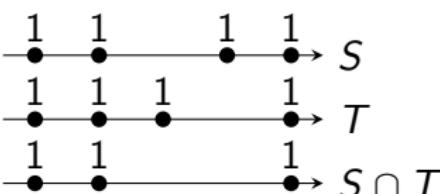
Example Set Operation on Generating Functions: Hadamard Product

$$f(\mathbf{x}) = \sum_{\mathbf{s} \in \mathbb{Z}^d} a(\mathbf{s}) \mathbf{x}^{\mathbf{s}} \quad g(\mathbf{x}) = \sum_{\mathbf{s} \in \mathbb{Z}^d} b(\mathbf{s}) \mathbf{x}^{\mathbf{s}}$$

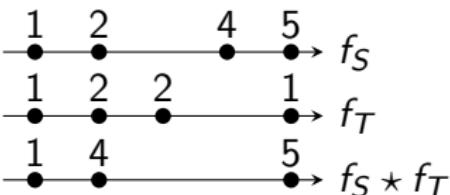
Hadamard product:

$$f(\mathbf{x}) * g(\mathbf{x}) = \sum_{\mathbf{s} \in \mathbb{Z}^d} a(\mathbf{s}) b(\mathbf{s}) \mathbf{x}^{\mathbf{s}}$$

Corresponds to intersection on sets:



$$\begin{aligned} f(S; \mathbf{x}) * f(T; \mathbf{x}) &= \sum_{\mathbf{s} \in \mathbb{Z}^d} [\mathbf{s} \in S][\mathbf{s} \in T] \mathbf{x}^{\mathbf{s}} \\ &= \sum_{\mathbf{s} \in \mathbb{Z}^d} [\mathbf{s} \in S \cap T] \mathbf{x}^{\mathbf{s}} \end{aligned}$$



Multi-sets: counts pairs of occurrences

Outline

8 Polyhedra

9 Operations

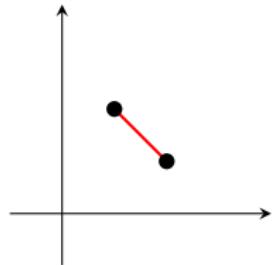
10 Vertex Enumeration

- Parametric Vertex Enumeration

11 Bounds on Quasi-Polynomials

12 Parametric Integer Programming

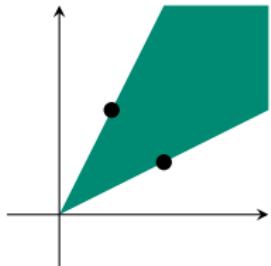
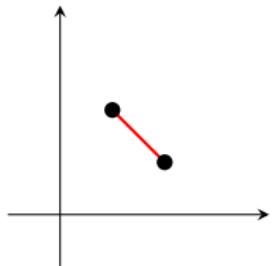
Hulls



- Convex hull

$$\text{conv.hull } S = \left\{ \sum_i \lambda_i \mathbf{v}_i : \mathbf{v}_i \in S \wedge \lambda_i \in \mathbb{Q} \wedge 0 \leq \lambda_i \leq 1 \wedge \sum_i \lambda_i = 1 \right\}$$

Hulls



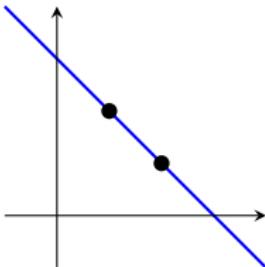
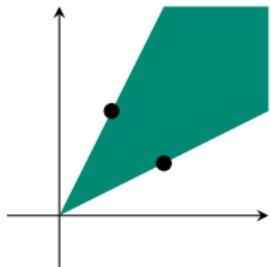
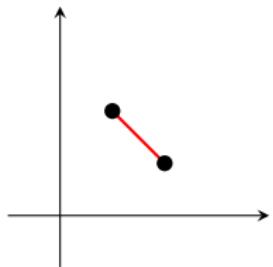
- Convex hull

$$\text{conv.hull } S = \left\{ \sum_i \lambda_i \mathbf{v}_i : \mathbf{v}_i \in S \wedge \lambda_i \in \mathbb{Q} \wedge 0 \leq \lambda_i \leq 1 \wedge \sum_i \lambda_i = 1 \right\}$$

- Positive/conic hull

$$\text{pos.hull } S = \left\{ \sum_i \lambda_i \mathbf{v}_i : \mathbf{v}_i \in S \wedge \lambda_i \in \mathbb{Q} \wedge \lambda_i \geq 0 \right\}$$

Hulls



- Convex hull

$$\text{conv.hull } S = \left\{ \sum_i \lambda_i \mathbf{v}_i : \mathbf{v}_i \in S \wedge \lambda_i \in \mathbb{Q} \wedge 0 \leq \lambda_i \leq 1 \wedge \sum_i \lambda_i = 1 \right\}$$

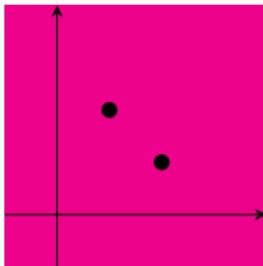
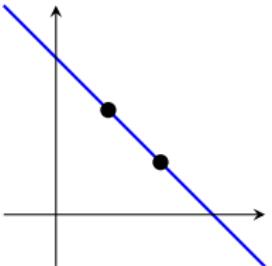
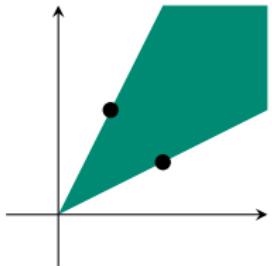
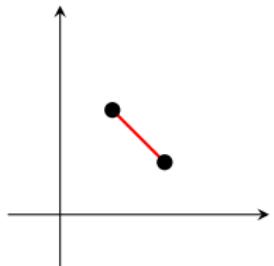
- Positive/conic hull

$$\text{pos.hull } S = \left\{ \sum_i \lambda_i \mathbf{v}_i : \mathbf{v}_i \in S \wedge \lambda_i \in \mathbb{Q} \wedge \lambda_i \geq 0 \right\}$$

- Affine hull (equalities satisfied by all points in S)

$$\text{aff.hull } S = \left\{ \sum_i \lambda_i \mathbf{v}_i : \mathbf{v}_i \in S \wedge \lambda_i \in \mathbb{Q} \wedge \sum_i \lambda_i = 1 \right\}$$

Hulls



- Convex hull

$$\text{conv.hull } S = \left\{ \sum_i \lambda_i \mathbf{v}_i : \mathbf{v}_i \in S \wedge \lambda_i \in \mathbb{Q} \wedge 0 \leq \lambda_i \leq 1 \wedge \sum_i \lambda_i = 1 \right\}$$

- Positive/conic hull

$$\text{pos.hull } S = \left\{ \sum_i \lambda_i \mathbf{v}_i : \mathbf{v}_i \in S \wedge \lambda_i \in \mathbb{Q} \wedge \lambda_i \geq 0 \right\}$$

- Affine hull (equalities satisfied by all points in S)

$$\text{aff.hull } S = \left\{ \sum_i \lambda_i \mathbf{v}_i : \mathbf{v}_i \in S \wedge \lambda_i \in \mathbb{Q} \wedge \sum_i \lambda_i = 1 \right\}$$

- Linear hull

$$\text{lin.hull } S = \left\{ \sum_i \lambda_i \mathbf{v}_i : \mathbf{v}_i \in S \wedge \lambda_i \in \mathbb{Q} \right\}$$

Polyhedra and Cones

- Polytope

$$P = \text{conv.hull } V$$

- Polyhedron (pos.hull R is called the *recession cone* of P)

$$P = \text{conv.hull } V + \text{pos.hull } R$$

- Cone

$$C = \text{pos.hull } R$$

- Polyhedron: intersection of a finite number of affine halfspaces

$$P = \left\{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{c} \right\}$$

- Cone: intersection of a finite number of linear halfspaces

$$C = \left\{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{0} \right\}$$

Polyhedra and Cones

- Polyhedron: intersection of affine halfspaces $P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{c} \}$
- Cone: intersection of linear halfspaces $C = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{0} \}$

Polyhedra and Cones

- Polyhedron: intersection of affine halfspaces $P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{c} \}$
- Cone: intersection of linear halfspaces $C = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{0} \}$

Take cone

$$C = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} + \mathbf{b}z \geq \mathbf{0} \}$$

and substitute $z = 1$:

$$P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} + \mathbf{b} \geq \mathbf{0} \}$$

\Rightarrow polyhedron P that is the intersection of C with hyperplane $z = 1$

Polyhedra and Cones

- Polyhedron: intersection of affine halfspaces $P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{c} \}$
- Cone: intersection of linear halfspaces $C = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{0} \}$

Take cone

$$C = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} + \mathbf{b}z \geq \mathbf{0} \}$$

and substitute $z = 1$:

$$P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} + \mathbf{b} \geq \mathbf{0} \}$$

\Rightarrow polyhedron P that is the intersection of C with hyperplane $z = 1$

Similarly, any polyhedron

$$P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{c} \}$$

can be represented by a cone in “homogeneous space”

$$C = \text{pos.hull}(P \times \{1\}) = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} - \mathbf{c}z \geq \mathbf{0} \wedge z \geq 0 \}$$

Polyhedra and Cones

- Polyhedron: intersection of affine halfspaces $P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{c} \}$
- Cone: intersection of linear halfspaces $C = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{0} \}$

Take cone

$$C = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} + \mathbf{b}z \geq \mathbf{0} \}$$

and substitute $z = 1$:

$$P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} + \mathbf{b} \geq \mathbf{0} \}$$

\Rightarrow polyhedron P that is the intersection of C with hyperplane $z = 1$

Similarly, any polyhedron

$$P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{c} \}$$

can be represented by a cone in “homogeneous space”

$$C = \text{pos.hull}(P \times \{1\}) = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} - \mathbf{c}z \geq \mathbf{0} \wedge z \geq 0 \}$$

Polyhedra and Cones

- Polyhedron: intersection of affine halfspaces $P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{c} \}$
- Cone: intersection of linear halfspaces $C = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{0} \}$

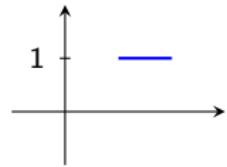
Take cone

$$C = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} + \mathbf{b}z \geq \mathbf{0} \}$$

and substitute $z = 1$:

$$P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} + \mathbf{b} \geq \mathbf{0} \}$$

\Rightarrow polyhedron P that is the intersection of C with hyperplane $z = 1$



Similarly, any polyhedron

$$P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{c} \}$$

can be represented by a cone in “homogeneous space”

$$C = \text{pos.hull}(P \times \{1\}) = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} - \mathbf{c}z \geq \mathbf{0} \wedge z \geq 0 \}$$

Polyhedra and Cones

- Polyhedron: intersection of affine halfspaces $P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{c} \}$
- Cone: intersection of linear halfspaces $C = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{0} \}$

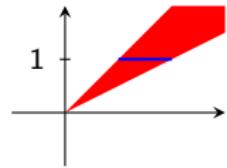
Take cone

$$C = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} + \mathbf{b}z \geq \mathbf{0} \}$$

and substitute $z = 1$:

$$P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} + \mathbf{b} \geq \mathbf{0} \}$$

\Rightarrow polyhedron P that is the intersection of C with hyperplane $z = 1$



Similarly, any polyhedron

$$P = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} \geq \mathbf{c} \}$$

can be represented by a cone in “homogeneous space”

$$C = \text{pos.hull}(P \times \{1\}) = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} - \mathbf{c}z \geq \mathbf{0} \wedge z \geq 0 \}$$

Linear Programming and Faces

LP = Linear Programming

$$\max_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle \quad \text{with } P = \{ \mathbf{x} : A\mathbf{x} \geq \mathbf{c} \}$$

⇒ maximum of linear objective function over polyhedron

Linear Programming and Faces

LP = Linear Programming

$$\max_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle \quad \text{with } P = \{ \mathbf{x} : A\mathbf{x} \geq \mathbf{c} \}$$

⇒ maximum of linear objective function over polyhedron

Set of elements of P that optimizes the objective function is called a *face*

- face of dimension $d - 1$: facet
- face of dimension $d - 2$: ridge
- face of dimension 2: edge
- face of dimension 1: vertex

(Polyhedron P of dimension d)

Linear Programming and Faces

LP = Linear Programming

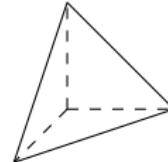
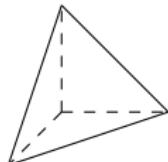
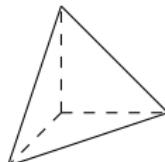
$$\max_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle \quad \text{with } P = \{ \mathbf{x} : A\mathbf{x} \geq \mathbf{c} \}$$

⇒ maximum of linear objective function over polyhedron

Set of elements of P that optimizes the objective function is called a *face*

- face of dimension $d - 1$: facet
- face of dimension $d - 2$: ridge
- face of dimension 2: edge
- face of dimension 1: vertex

(Polyhedron P of dimension d)



Linear Programming and Faces

LP = Linear Programming

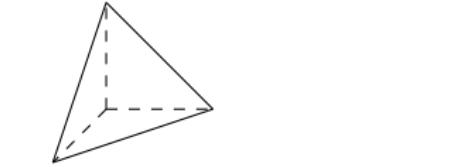
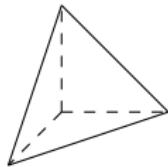
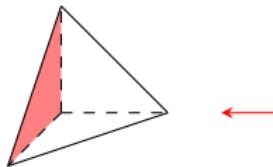
$$\max_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle \quad \text{with } P = \{ \mathbf{x} : A\mathbf{x} \geq \mathbf{c} \}$$

⇒ maximum of linear objective function over polyhedron

Set of elements of P that optimizes the objective function is called a *face*

- face of dimension $d - 1$: **facet**
- face of dimension $d - 2$: ridge
- face of dimension 2: edge
- face of dimension 1: vertex

(Polyhedron P of dimension d)



Linear Programming and Faces

LP = Linear Programming

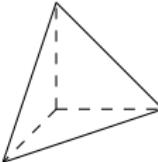
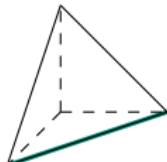
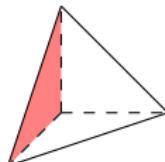
$$\max_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle \quad \text{with } P = \{ \mathbf{x} : A\mathbf{x} \geq \mathbf{c} \}$$

⇒ maximum of linear objective function over polyhedron

Set of elements of P that optimizes the objective function is called a *face*

- face of dimension $d - 1$: **facet**
- face of dimension $d - 2$: **ridge**
- face of dimension 2: **edge**
- face of dimension 1: **vertex**

(Polyhedron P of dimension d)



Linear Programming and Faces

LP = Linear Programming

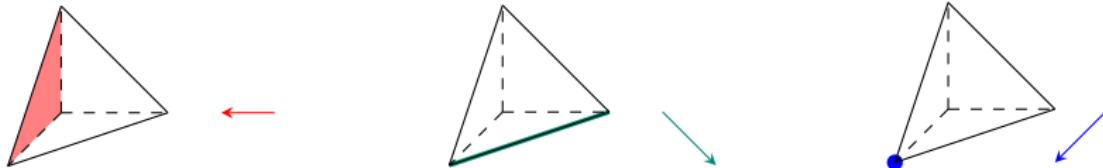
$$\max_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle \quad \text{with } P = \{ \mathbf{x} : A\mathbf{x} \geq \mathbf{c} \}$$

⇒ maximum of linear objective function over polyhedron

Set of elements of P that optimizes the objective function is called a *face*

- face of dimension $d - 1$: **facet**
- face of dimension $d - 2$: **ridge**
- face of dimension 2: **edge**
- face of dimension 1: **vertex**

(Polyhedron P of dimension d)



Outline

8 Polyhedra

9 Operations

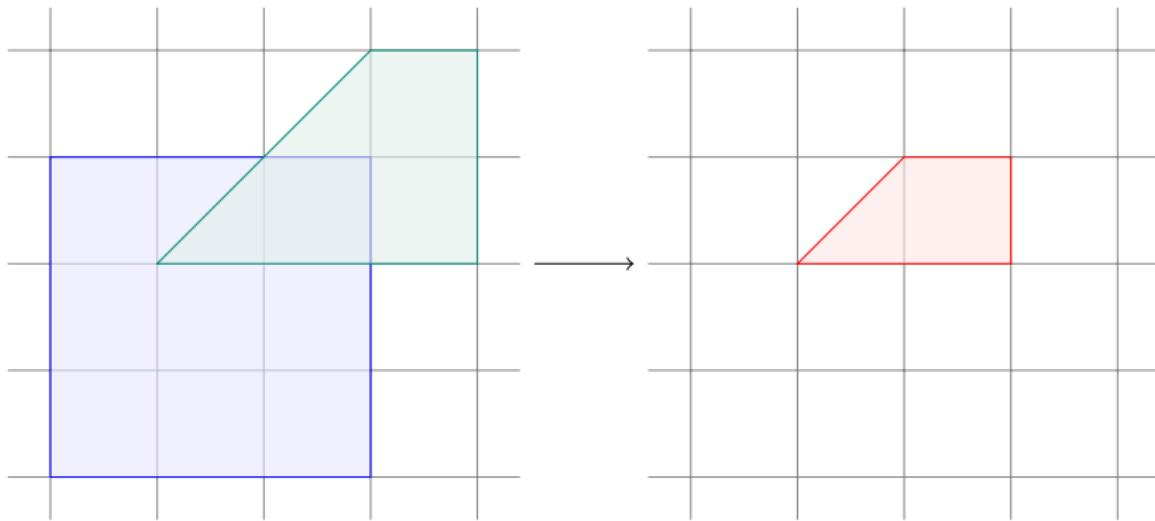
10 Vertex Enumeration

- Parametric Vertex Enumeration

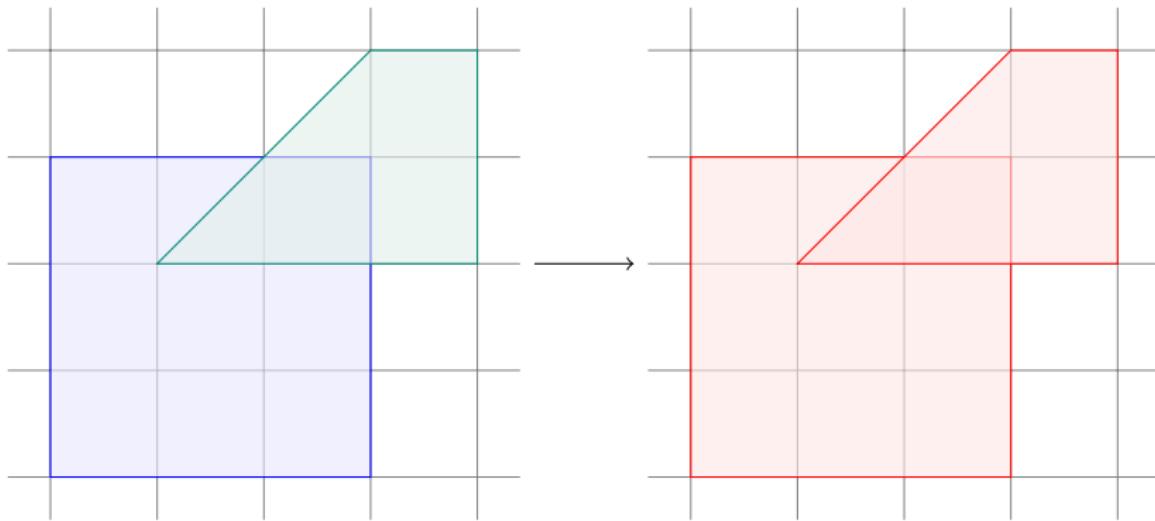
11 Bounds on Quasi-Polynomials

12 Parametric Integer Programming

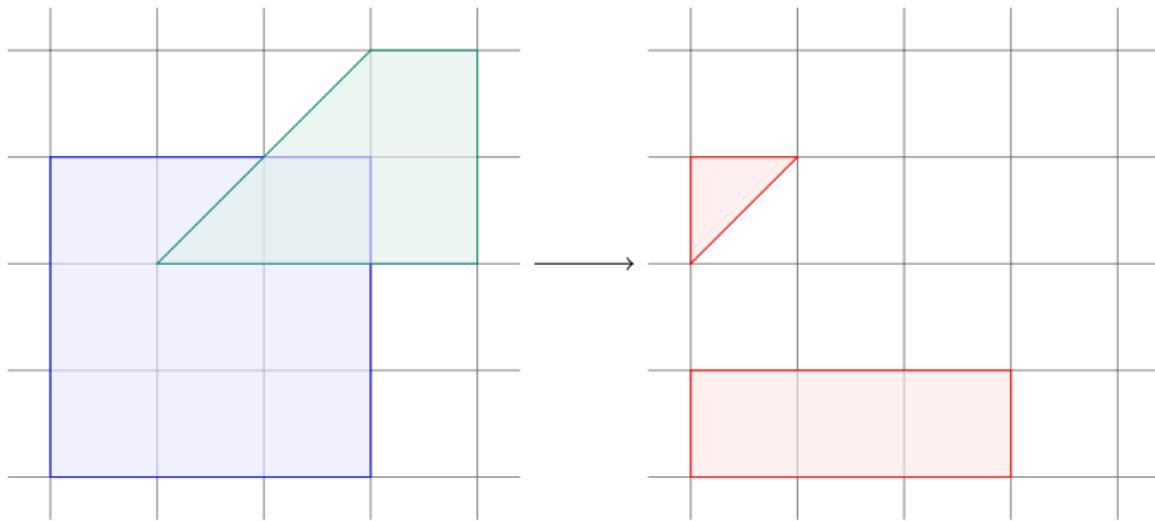
Intersection



Union



isl Operation: Set Difference



isl Operation: Set Difference

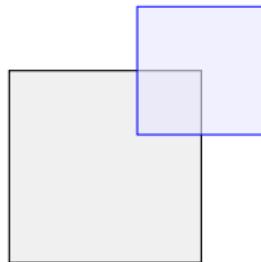
$$S(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : A\mathbf{x} + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \}$$

Set difference $S_1 \setminus S_2$

- no existentially quantified variables

$$S_2(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Z}^d \mid \bigwedge_i \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i \}$$

$$S_1 \setminus S_2 = \bigcup_i (S_1 \cap \{ \mathbf{x} \mid \neg (\langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i) \})$$



isl Operation: Set Difference

$$S(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : A\mathbf{x} + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \}$$

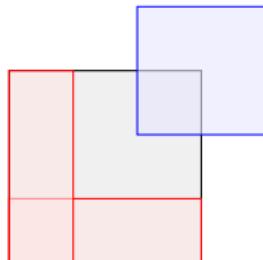
Set difference $S_1 \setminus S_2$

- no existentially quantified variables

$$S_2(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Z}^d \mid \bigwedge_i \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i \}$$

$$S_1 \setminus S_2 = \bigcup_i (S_1 \cap \{ \mathbf{x} \mid \neg (\langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i) \})$$

$$= \bigcup_i (S_1 \cap \{ \mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \leq c_i - 1 \})$$



isl Operation: Set Difference

$$S(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : A\mathbf{x} + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \}$$

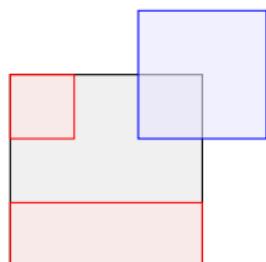
Set difference $S_1 \setminus S_2$

- no existentially quantified variables

$$S_2(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Z}^d \mid \bigwedge_i \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i \}$$

$$S_1 \setminus S_2 = \bigcup_i (S_1 \cap \{ \mathbf{x} \mid \neg (\langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i) \})$$

$$= \bigcup_i (S_1 \cap \{ \mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \leq c_i - 1 \})$$



$$= \bigcup_i (S_1 \cap \bigcap_{j < i} \{ \mathbf{x} \mid \langle \mathbf{a}_j, \mathbf{x} \rangle + \langle \mathbf{b}_j, \mathbf{s} \rangle \geq c_j \}$$

$$\cap \{ \mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \leq c_i - 1 \})$$

isl Operation: Set Difference

$$S(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : A\mathbf{x} + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \}$$

Set difference $S_1 \setminus S_2$

- no existentially quantified variables

$$S_2(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Z}^d \mid \bigwedge_i \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i \}$$

$$\begin{aligned} S_1 \setminus S_2 &= \bigcup_i (S_1 \cap \bigcap_{j < i} \{ \mathbf{x} \mid \langle \mathbf{a}_j, \mathbf{x} \rangle + \langle \mathbf{b}_j, \mathbf{s} \rangle \geq c_j \} \\ &\quad \cap \{ \mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \leq c_i - 1 \}) \end{aligned}$$

isl Operation: Set Difference

$$S(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : A\mathbf{x} + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \}$$

Set difference $S_1 \setminus S_2$

- no existentially quantified variables

$$S_2(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Z}^d \mid \bigwedge_i \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i \}$$

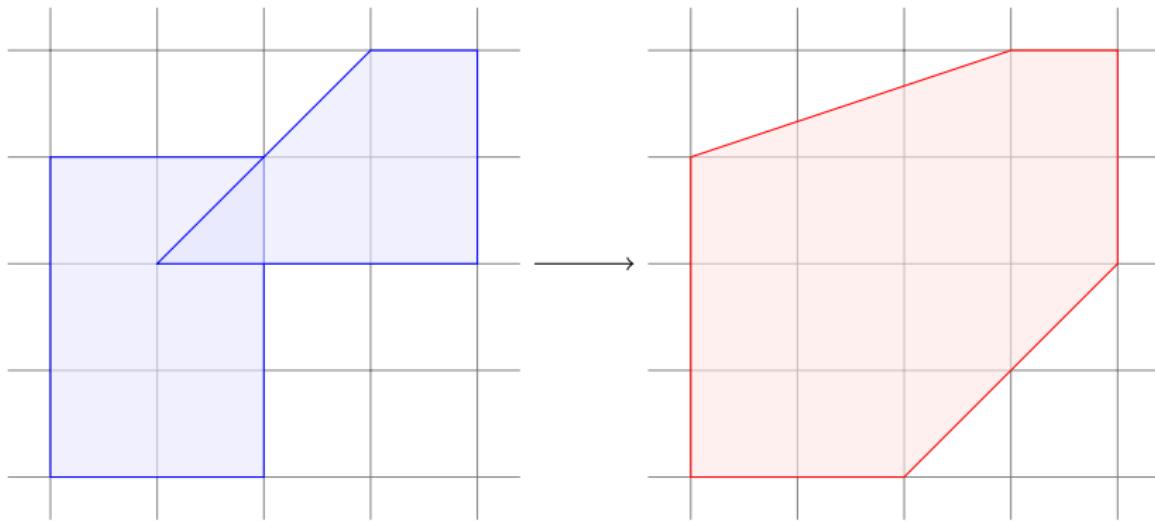
$$\begin{aligned} S_1 \setminus S_2 &= \bigcup_i (S_1 \cap \bigcap_{j < i} \{ \mathbf{x} \mid \langle \mathbf{a}_j, \mathbf{x} \rangle + \langle \mathbf{b}_j, \mathbf{s} \rangle \geq c_j \} \\ &\quad \cap \{ \mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \leq c_i - 1 \}) \end{aligned}$$

- with existentially quantified variables
 ⇒ compute explicit representation

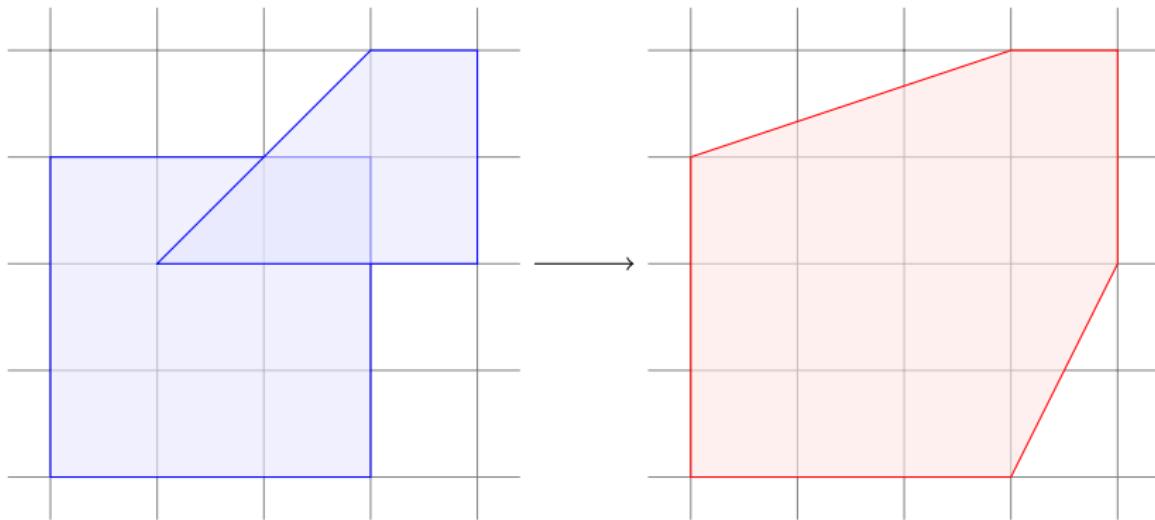
$$S_2(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Z}^d \mid \bigwedge_i \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle + \left\langle \mathbf{d}_i, \left\lfloor \frac{\langle \mathbf{p}, \mathbf{x} \rangle + \langle \mathbf{q}_i, \mathbf{s} \rangle + r}{m} \right\rfloor \right\rangle \geq c_i \}$$

isl Operation: Closed Convex Hull

[20]



isl Operation: Set Coalescing



isl Operation: Set Coalescing

After many applications of projection, set difference, union,
a set may be represented as a union of many basic sets
⇒ try to combine several basic sets into a single basic set

isl Operation: Set Coalescing

After many applications of projection, set difference, union,
a set may be represented as a union of many basic sets
⇒ try to combine several basic sets into a single basic set

$$S_1 = \{ \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \} \quad S_2 = \{ \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \}$$

PolyLib way:

- ① Compute $H = \text{conv.hull}(S_1 \cup S_2)$
- ② Replace $S_1 \cup S_2$ by $H \setminus (H \setminus (S_1 \cup S_2))$

isl Operation: Set Coalescing

After many applications of projection, set difference, union,
a set may be represented as a union of many basic sets
⇒ try to combine several basic sets into a single basic set

$$S_1 = \{ \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \} \quad S_2 = \{ \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \}$$

PolyLib way:

- ① Compute $H = \text{conv.hull}(S_1 \cup S_2)$
- ② Replace $S_1 \cup S_2$ by $H \setminus (H \setminus (S_1 \cup S_2))$

isl way:

- ① Classify constraints
 - ▶ redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
 - ▶ valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
 - ▶ separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
 - special cases:
 - ★ adjacent to equality: $\langle \mathbf{a}_i, \mathbf{x} \rangle = c_i - 1$ over S_2
 - ★ adjacent to inequality: $\langle (\mathbf{a}_i + \mathbf{b}_j), \mathbf{x} \rangle = (c_i + d_j) - 1$ over S_2
 - ▶ cut: otherwise

isl Operation: Set Coalescing

① Classify constraints

- ▶ redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
- ▶ valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
- ▶ separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
- special cases:
 - ★ adjacent to equality: $\langle \mathbf{a}_i, \mathbf{x} \rangle = c_i - 1$ over S_2
 - ★ adjacent to inequality: $\langle (\mathbf{a}_i + \mathbf{b}_j), \mathbf{x} \rangle = (c_i + d_j) - 1$ over S_2
- ▶ cut: otherwise

isl Operation: Set Coalescing

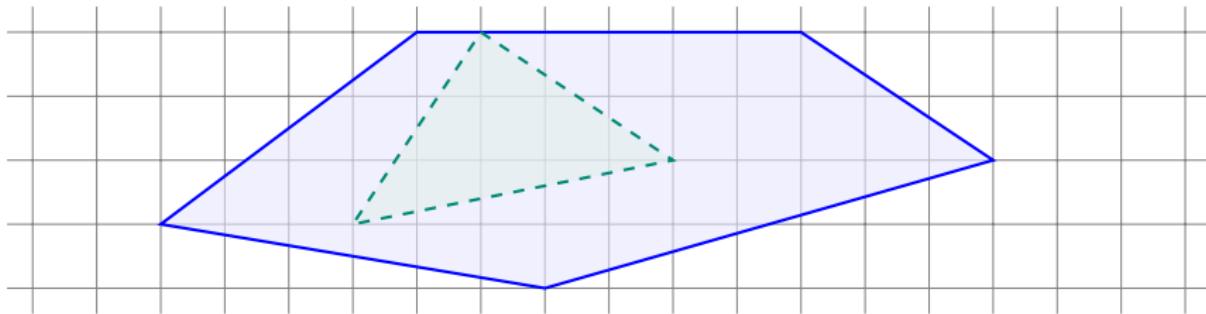
① Classify constraints

- ▶ redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
- ▶ valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
- ▶ separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
special cases:
 - ★ adjacent to equality: $\langle \mathbf{a}_i, \mathbf{x} \rangle = c_i - 1$ over S_2
 - ★ adjacent to inequality: $\langle (\mathbf{a}_i + \mathbf{b}_j), \mathbf{x} \rangle = (c_i + d_j) - 1$ over S_2
- ▶ cut: otherwise

② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
 $\Rightarrow S_2$ can be dropped

isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
 $\Rightarrow S_2$ can be dropped

isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
 $\Rightarrow S_2$ can be dropped

isl Operation: Set Coalescing

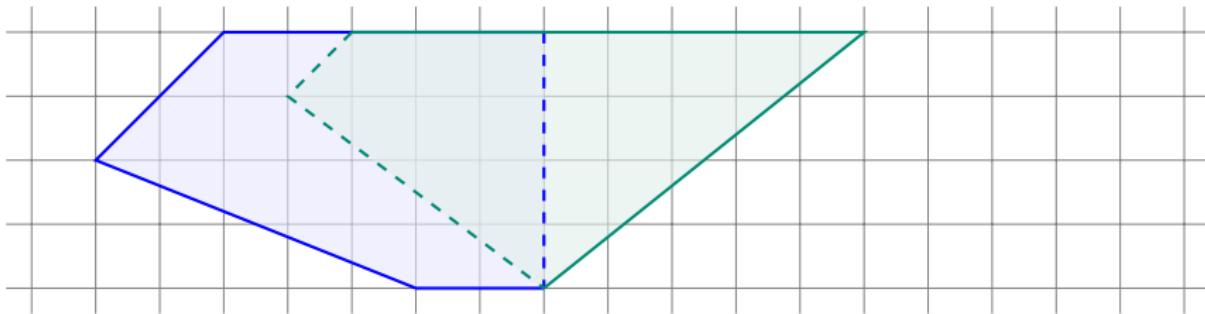
① Classify constraints

- ▶ redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
- ▶ valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
- ▶ separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
special cases:
 - ★ adjacent to equality: $\langle \mathbf{a}_i, \mathbf{x} \rangle = c_i - 1$ over S_2
 - ★ adjacent to inequality: $\langle (\mathbf{a}_i + \mathbf{b}_j), \mathbf{x} \rangle = (c_i + d_j) - 1$ over S_2
- ▶ cut: otherwise

② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
 \Rightarrow replace S_1 and S_2 by basic set with all valid constraints

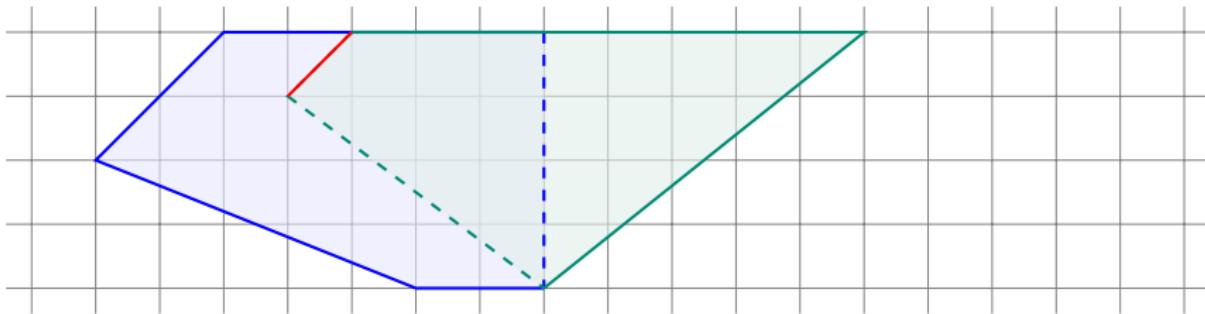
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
⇒ replace S_1 and S_2 by basic set with all valid constraints

isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
⇒ replace S_1 and S_2 by basic set with all valid constraints

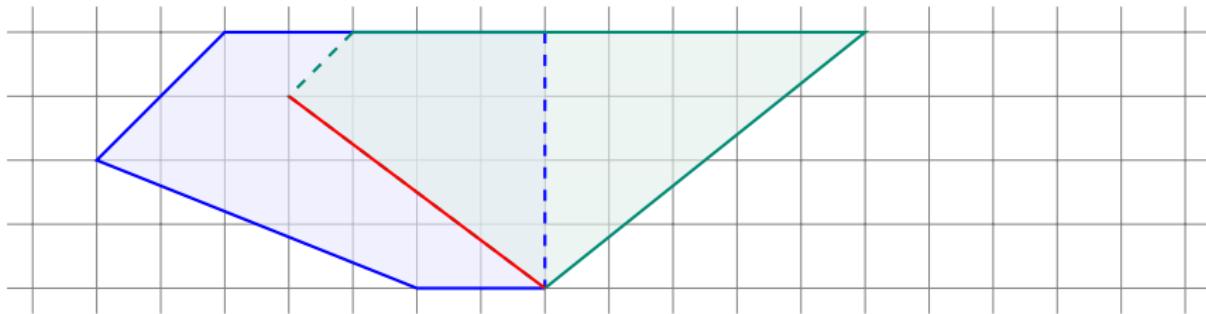
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
⇒ replace S_1 and S_2 by basic set with all valid constraints

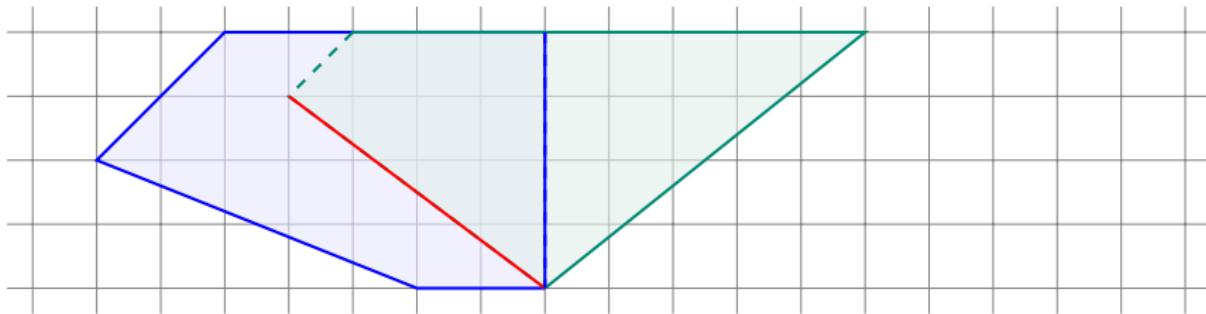
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
⇒ replace S_1 and S_2 by basic set with all valid constraints

isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
⇒ replace S_1 and S_2 by basic set with all valid constraints

isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
⇒ replace S_1 and S_2 by basic set with all valid constraints

isl Operation: Set Coalescing

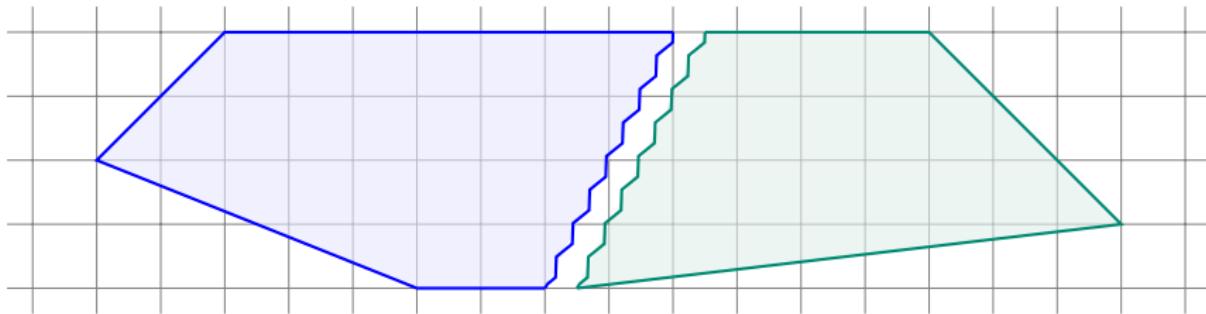
① Classify constraints

- ▶ redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
- ▶ valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
- ▶ separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
special cases:
 - ★ adjacent to equality: $\langle \mathbf{a}_i, \mathbf{x} \rangle = c_i - 1$ over S_2
 - ★ adjacent to inequality: $\langle (\mathbf{a}_i + \mathbf{b}_j), \mathbf{x} \rangle = (c_i + d_j) - 1$ over S_2
- ▶ cut: otherwise

② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
 \Rightarrow replace S_1 and S_2 by basic set with all valid constraints

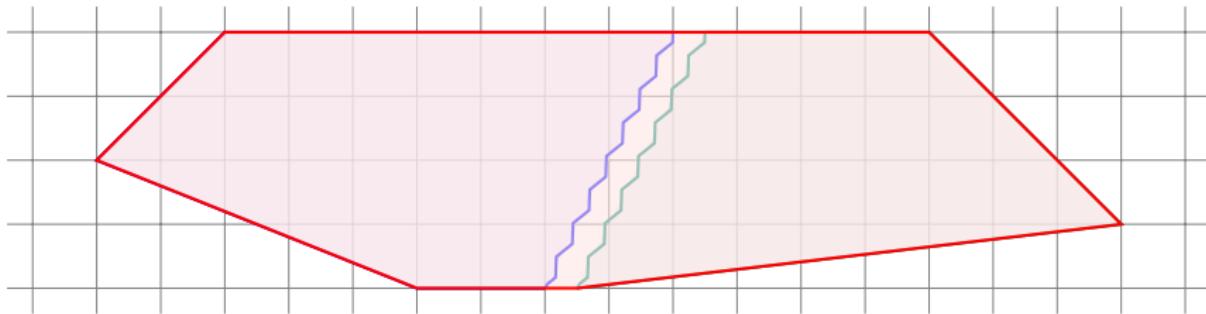
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
⇒ replace S_1 and S_2 by basic set with all valid constraints

isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
⇒ replace S_1 and S_2 by basic set with all valid constraints

isl Operation: Set Coalescing

① Classify constraints

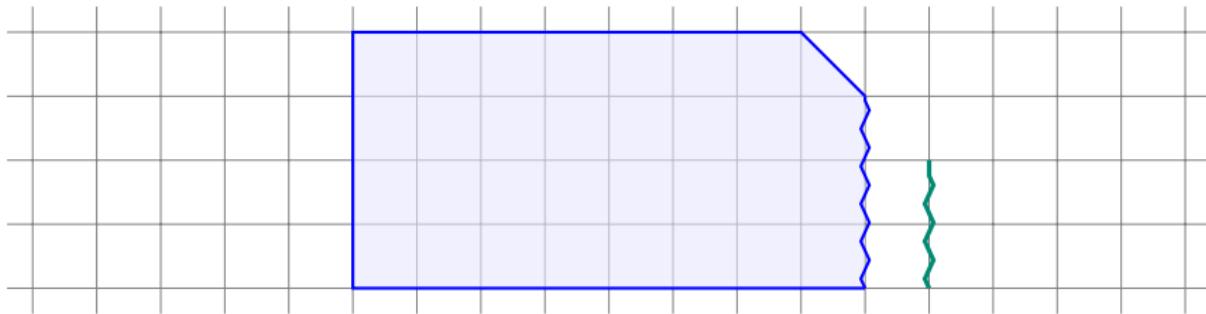
- ▶ redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
- ▶ valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
- ▶ separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
special cases:
 - ★ adjacent to equality: $\langle \mathbf{a}_i, \mathbf{x} \rangle = c_i - 1$ over S_2
 - ★ adjacent to inequality: $\langle (\mathbf{a}_i + \mathbf{b}_j), \mathbf{x} \rangle = (c_i + d_j) - 1$ over S_2
- ▶ cut: otherwise

② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
 - + other constraints of S_1 are valid
 - + constraints of S_2 valid for facet of relaxed inequality

\Rightarrow drop S_2 and relax adjacent inequality of S_1

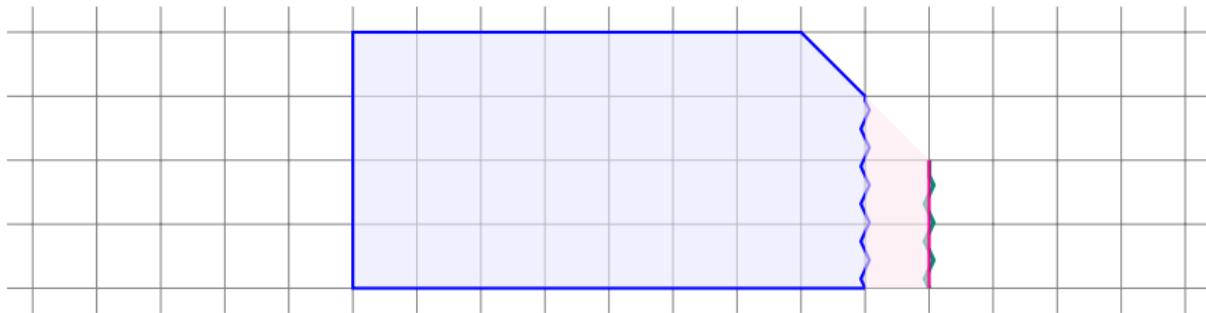
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
 - + other constraints of S_1 are valid
 - + constraints of S_2 valid for facet of relaxed inequality
 - \Rightarrow drop S_2 and relax adjacent inequality of S_1

isl Operation: Set Coalescing

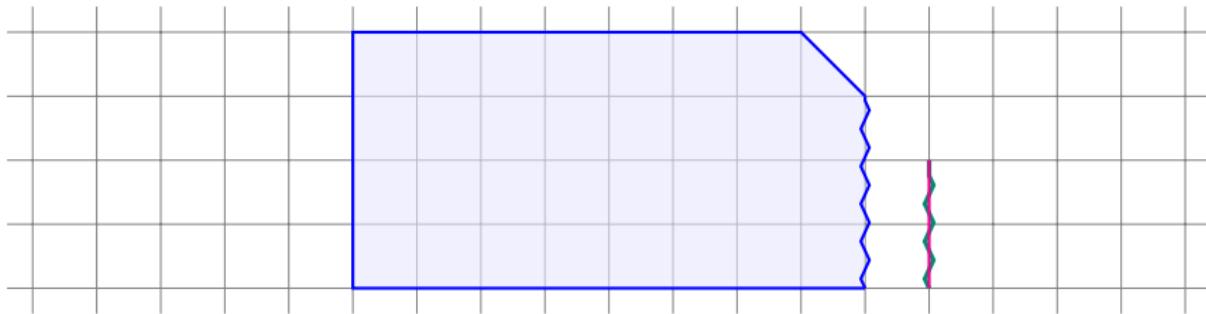


② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
 - + other constraints of S_1 are valid
 - + constraints of S_2 valid for facet of relaxed inequality

⇒ drop S_2 and relax adjacent inequality of S_1

isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
 - + other constraints of S_1 are valid
 - + constraints of S_2 valid for facet of relaxed inequality

\Rightarrow drop S_2 and relax adjacent inequality of S_1

isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
 - + other constraints of S_1 are valid
 - + constraints of S_2 valid for facet of relaxed inequality
 - \Rightarrow drop S_2 and relax adjacent inequality of S_1

isl Operation: Set Coalescing

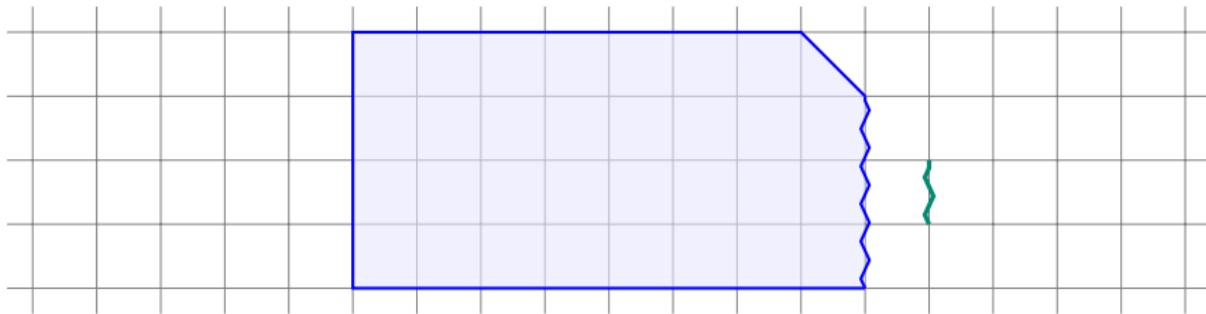
① Classify constraints

- ▶ redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
- ▶ valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
- ▶ separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
special cases:
 - ★ adjacent to equality: $\langle \mathbf{a}_i, \mathbf{x} \rangle = c_i - 1$ over S_2
 - ★ adjacent to inequality: $\langle (\mathbf{a}_i + \mathbf{b}_j), \mathbf{x} \rangle = (c_i + d_j) - 1$ over S_2
- ▶ cut: otherwise

② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
 - + constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
 - + other constraints of S_1 are valid
 - + inequality and equality can be wrapped to include union
 - \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

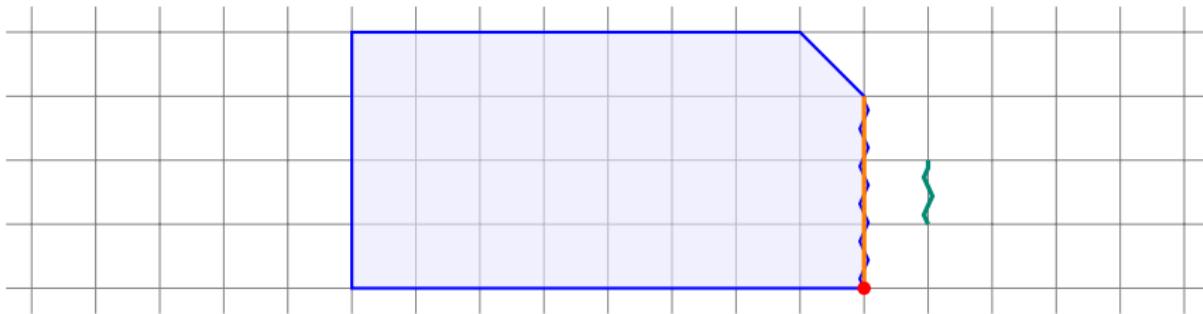
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

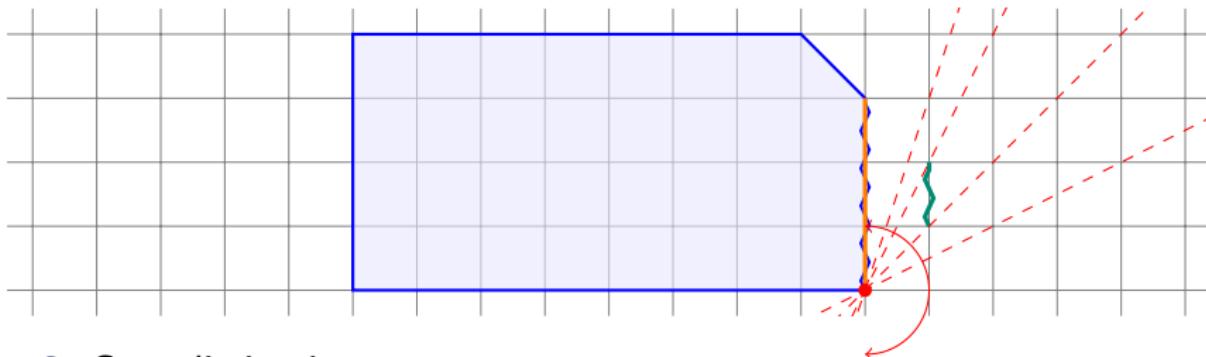
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

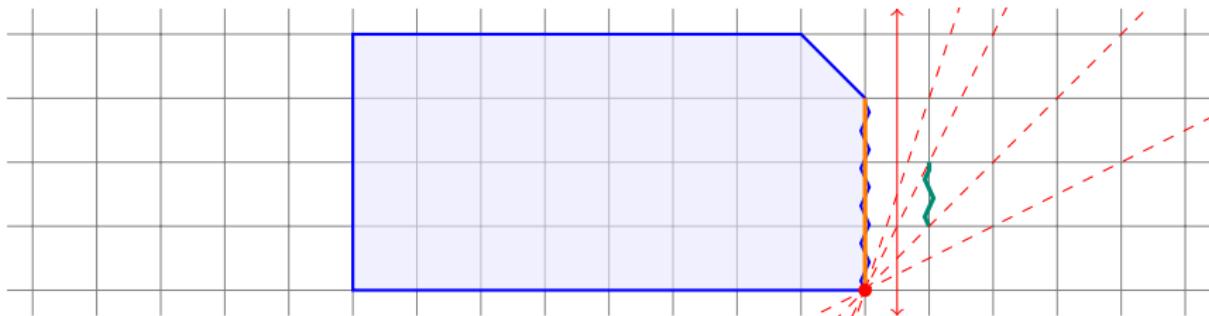
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

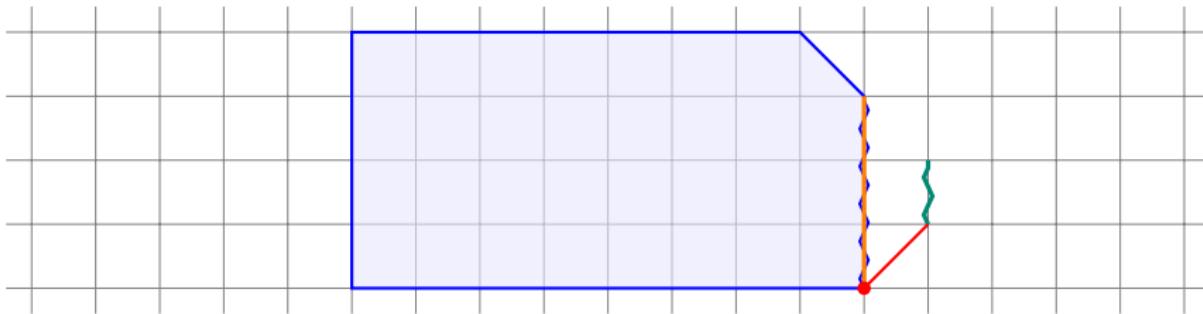
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

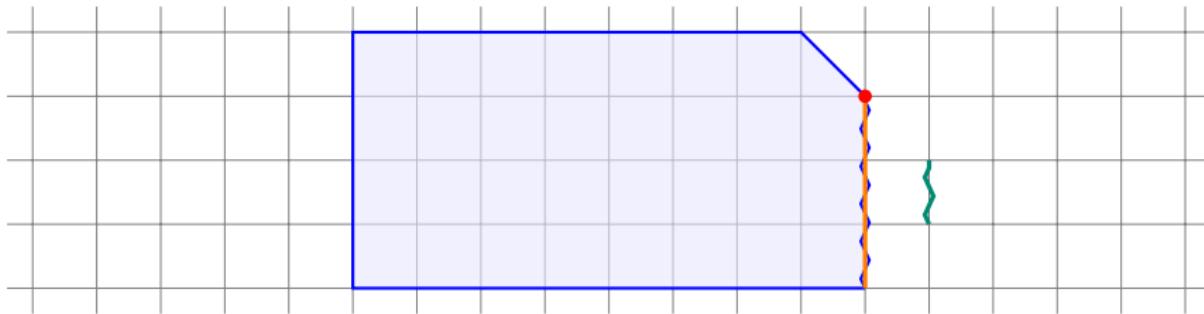
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

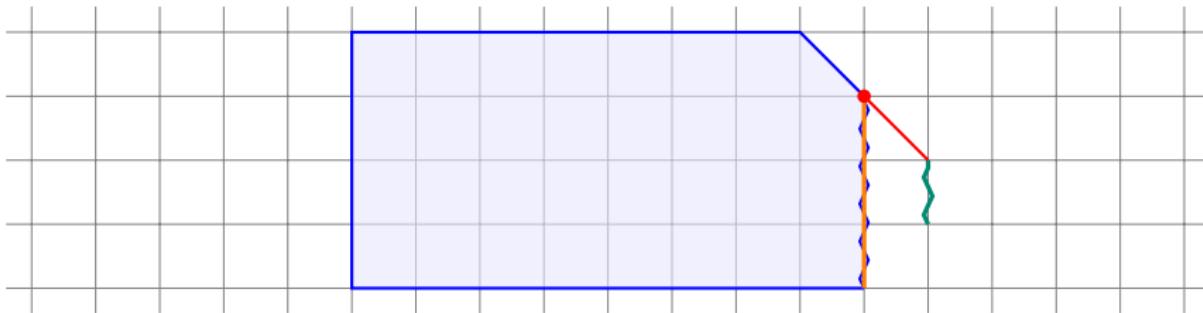
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

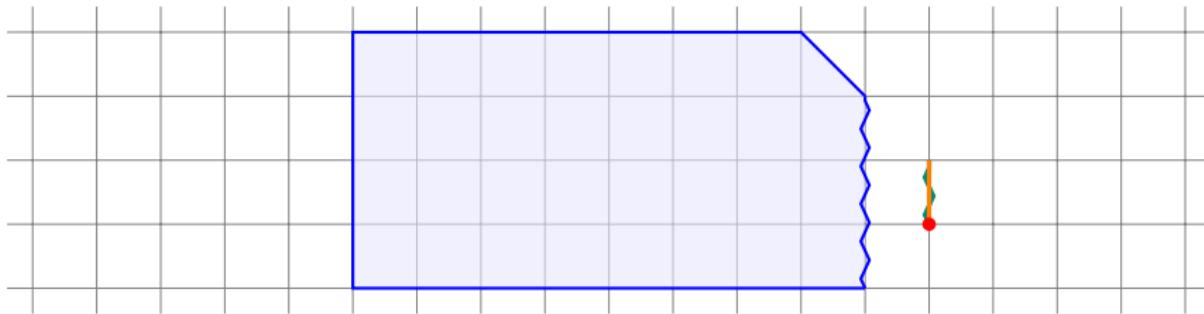
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

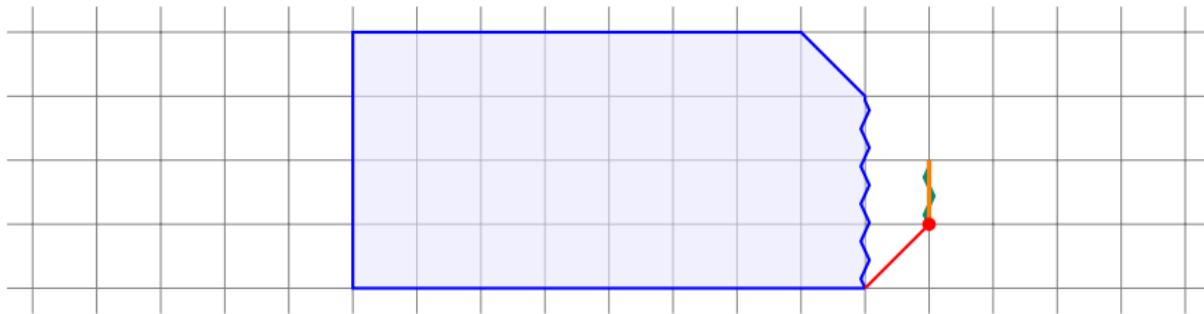
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

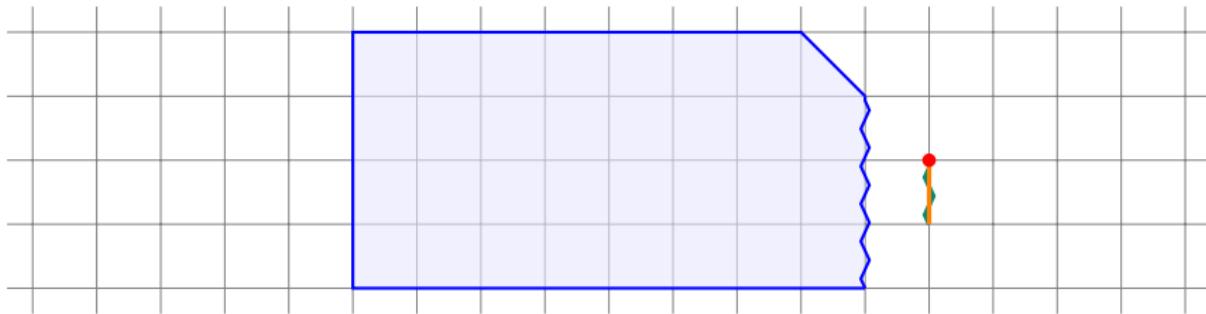
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

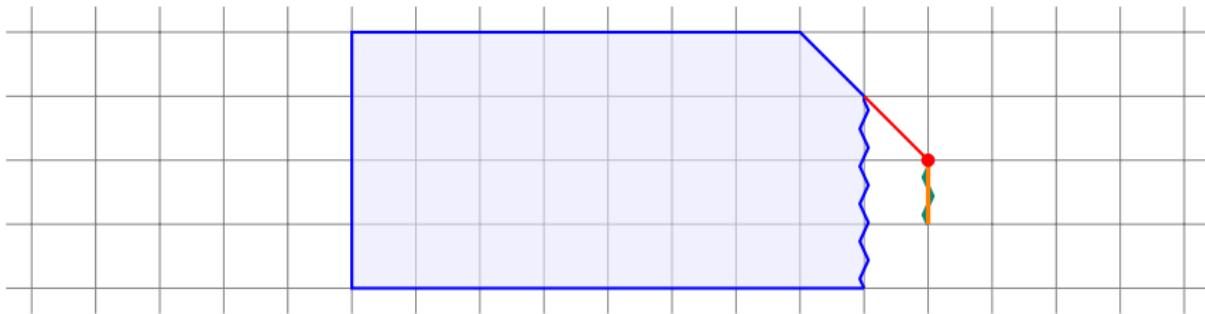
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ other constraints of S_1 are valid
+ inequality and equality can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

isl Operation: Set Coalescing

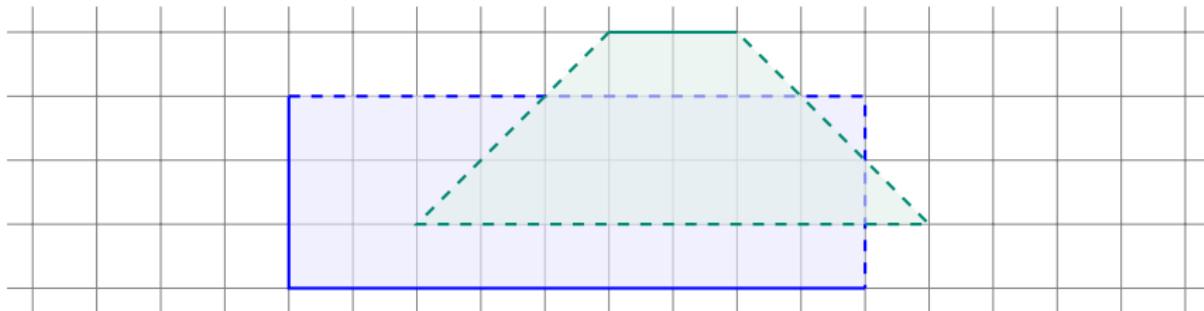
① Classify constraints

- ▶ redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
- ▶ valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
- ▶ separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
special cases:
 - ★ adjacent to equality: $\langle \mathbf{a}_i, \mathbf{x} \rangle = c_i - 1$ over S_2
 - ★ adjacent to inequality: $\langle (\mathbf{a}_i + \mathbf{b}_j), \mathbf{x} \rangle = (c_i + d_j) - 1$ over S_2
- ▶ cut: otherwise

② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
⇒ replace S_1 and S_2 by valid and wrapping constraints

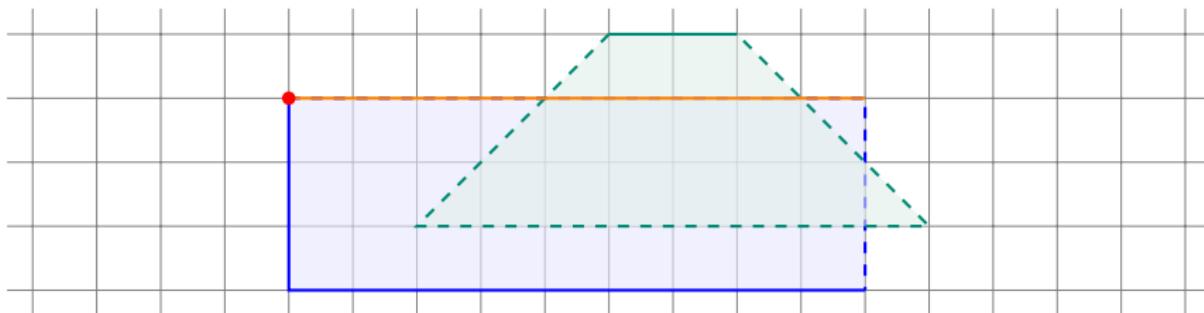
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

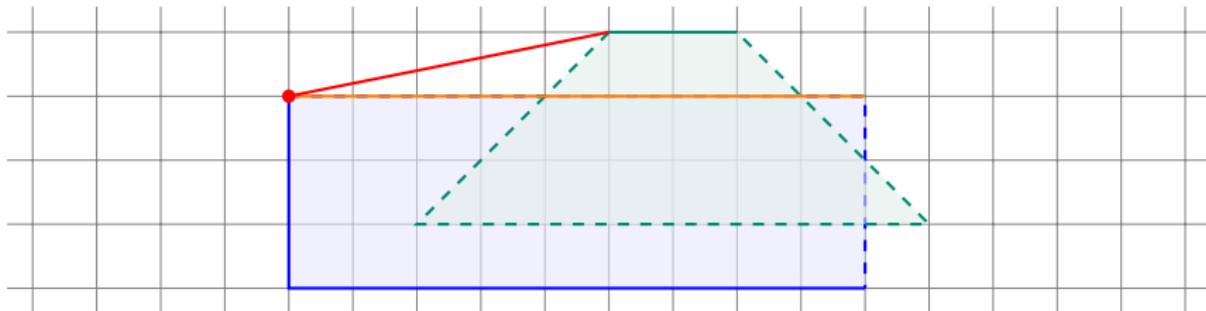
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

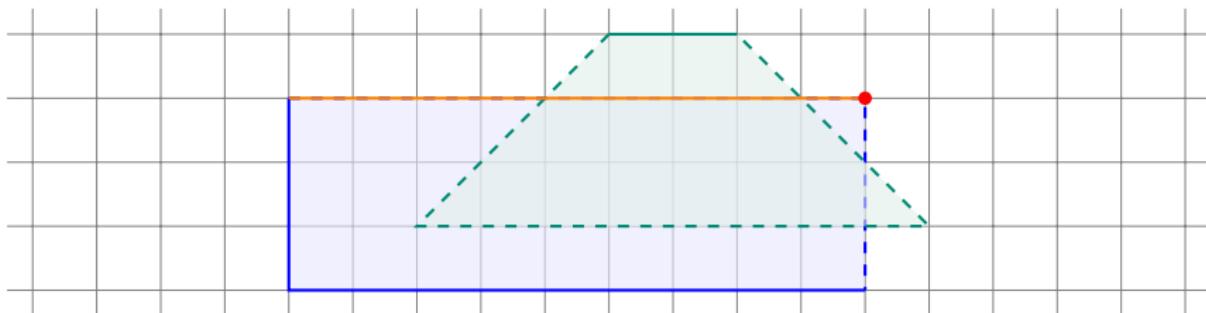
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
⇒ replace S_1 and S_2 by valid and wrapping constraints

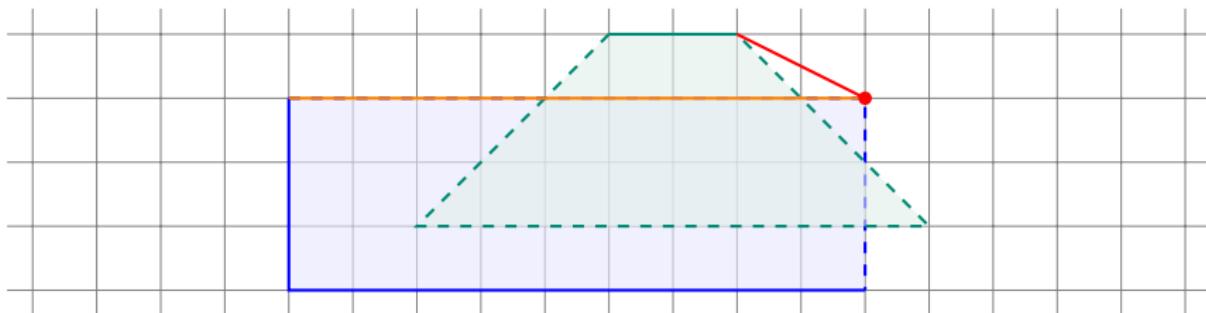
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

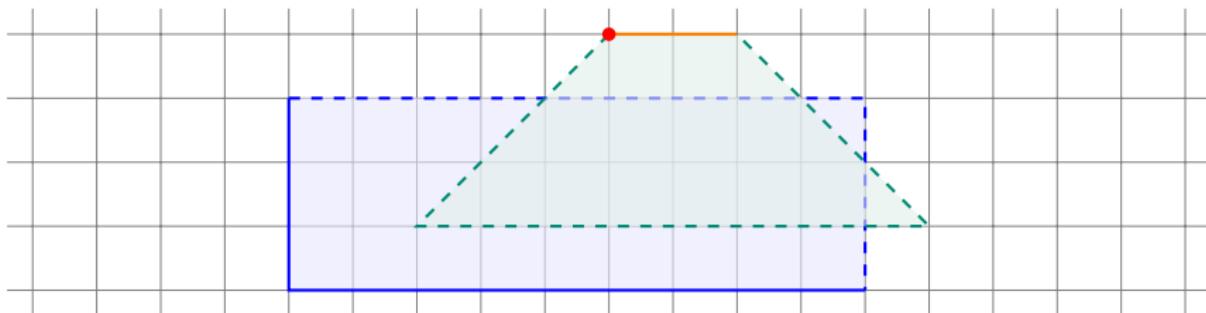
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

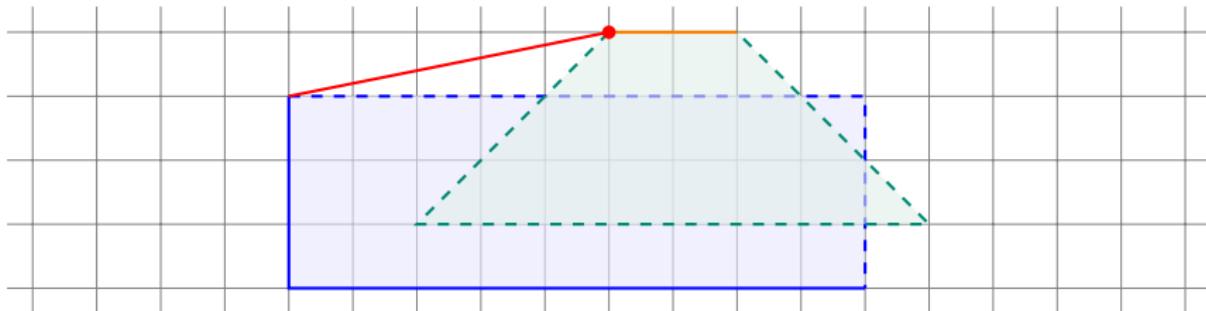
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

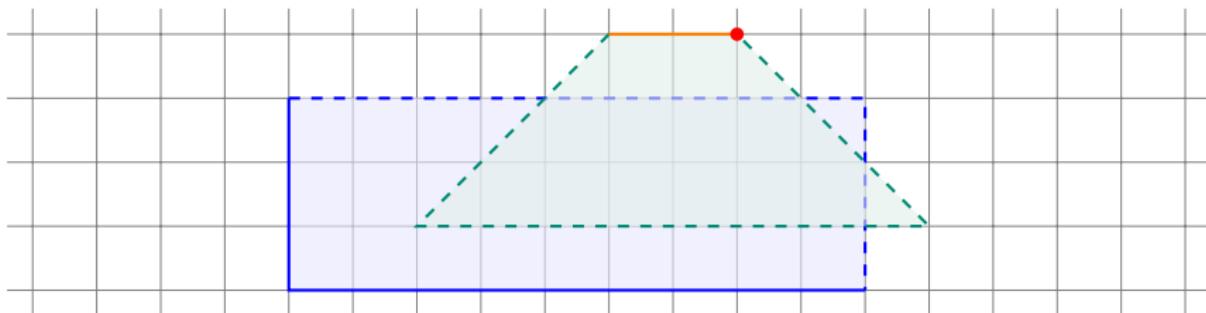
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

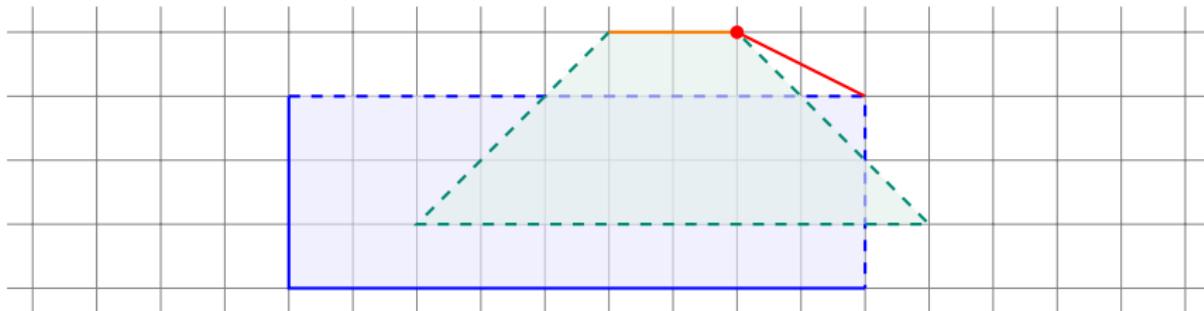
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

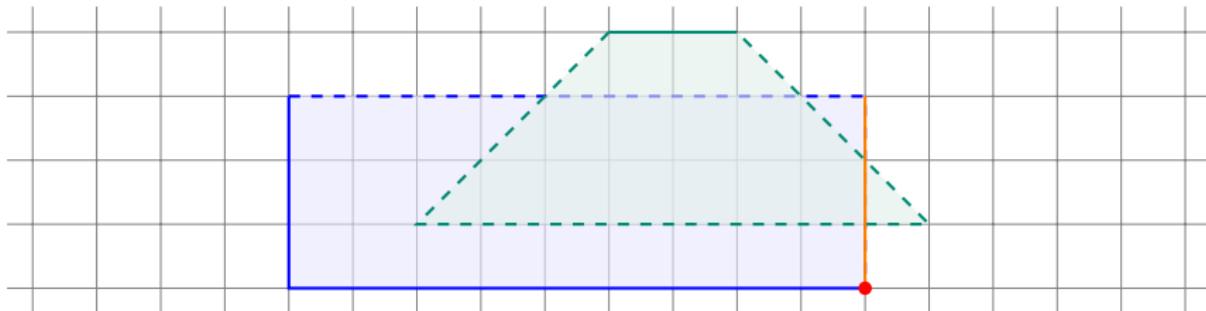
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
⇒ replace S_1 and S_2 by valid and wrapping constraints

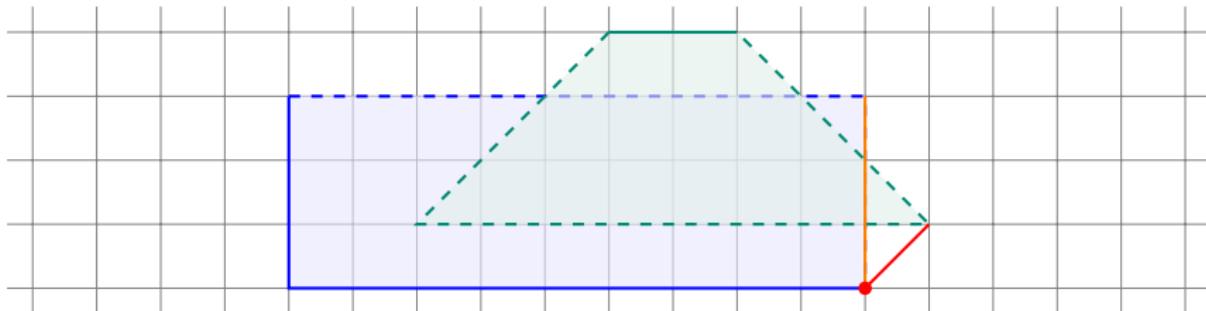
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

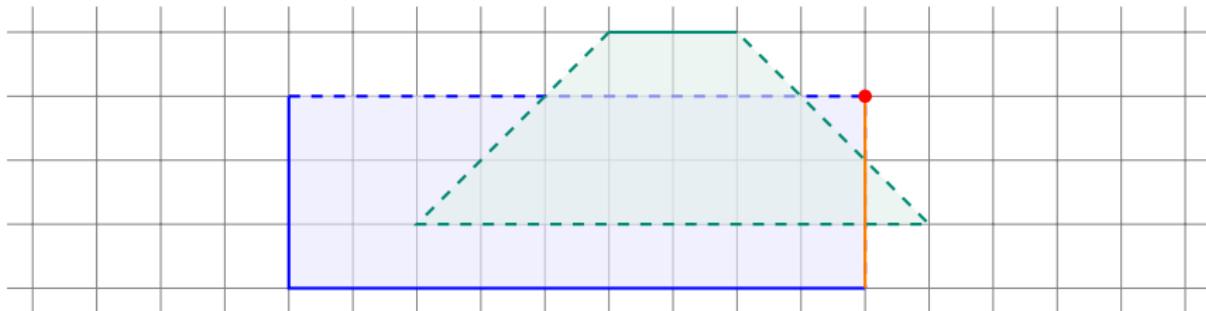
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

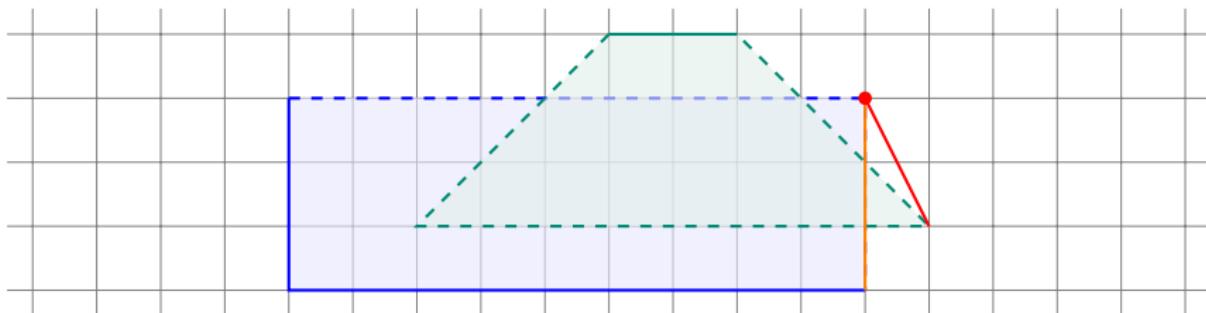
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

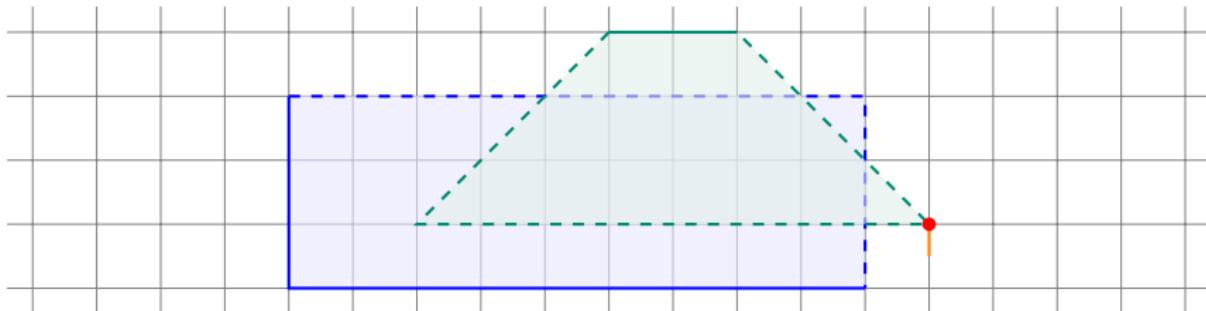
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

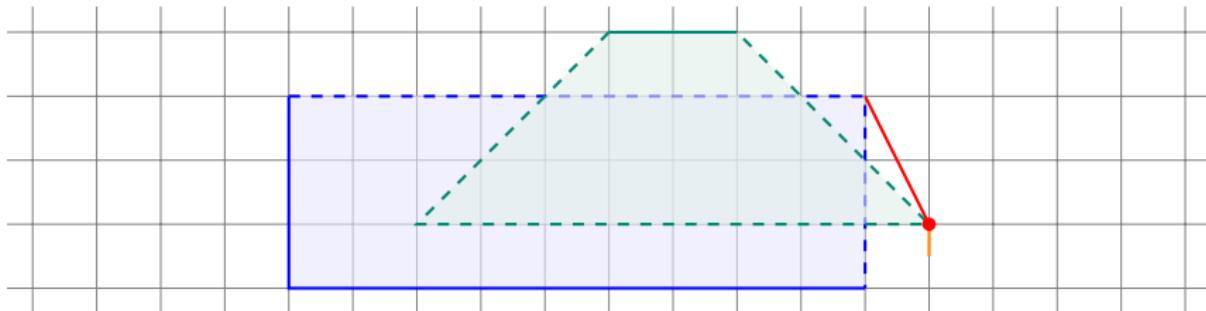
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

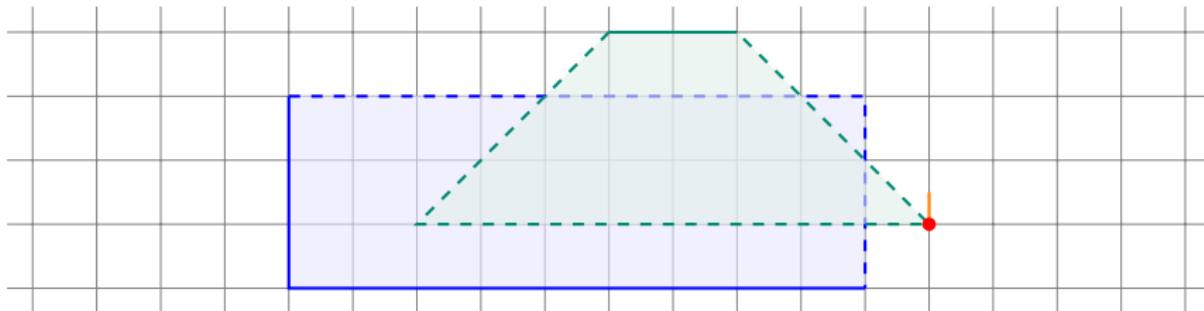
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

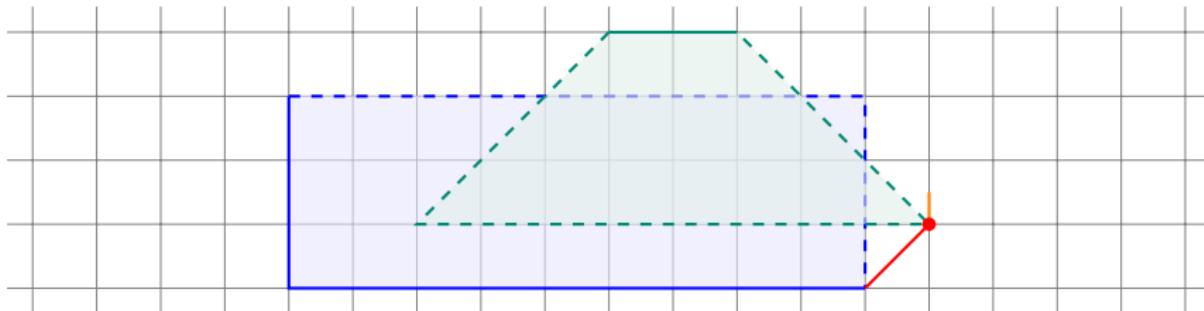
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

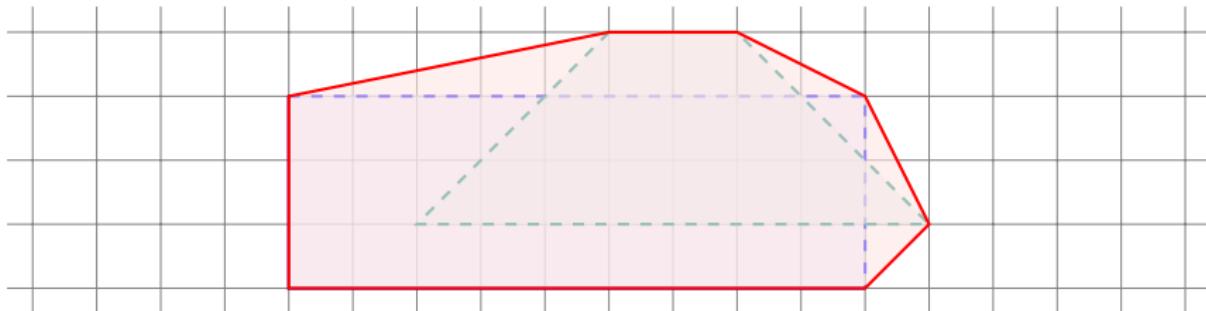
isl Operation: Set Coalescing



② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

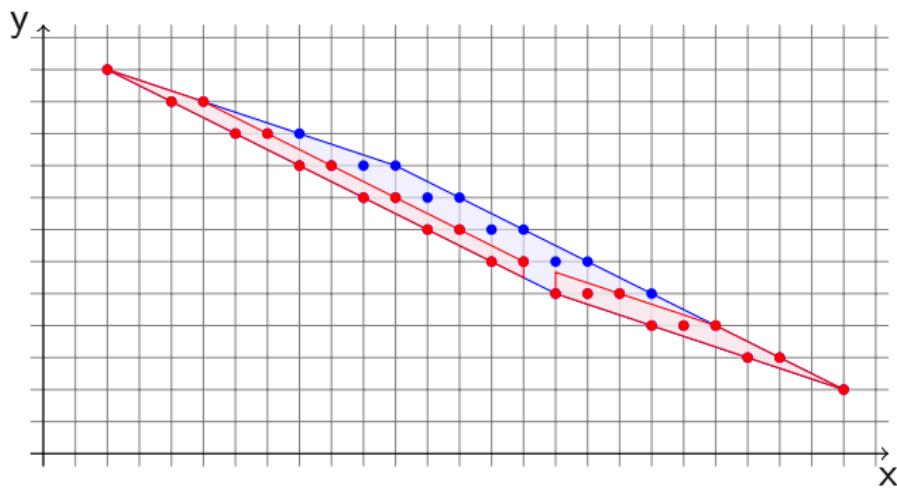
isl Operation: Set Coalescing



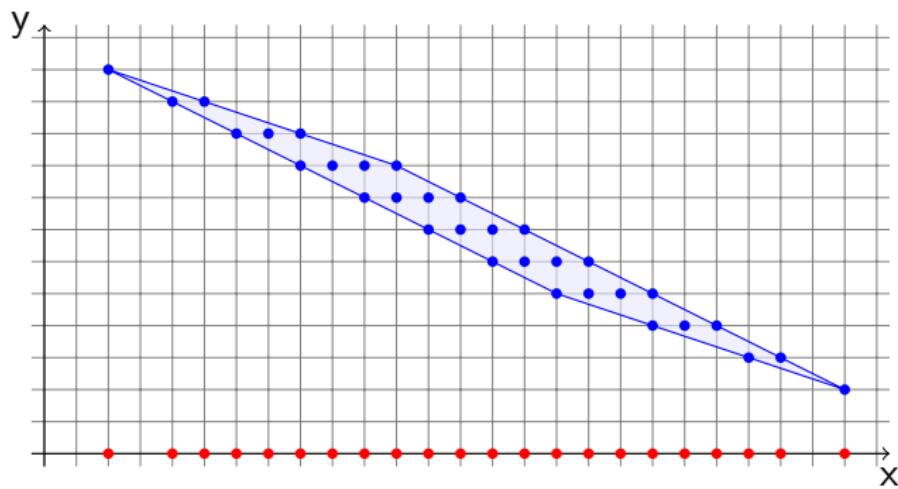
② Case distinction

- ① non-redundant constraints of S_1 are valid for S_2 , i.e., $S_2 \subseteq S_1$
- ② no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
- ③ single pair of adjacent inequalities (other constraints valid)
- ④ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ constraints of S_2 valid for facet of relaxed inequality
- ⑤ single adjacent pair of an inequality (S_1) and an equality (S_2)
+ inequality and equality can be wrapped to include union
- ⑥ S_2 extends beyond S_1 by at most one and all cut constraints of S_1 and parallel slices of S_2 can be wrapped to include union
 \Rightarrow replace S_1 and S_2 by valid and wrapping constraints

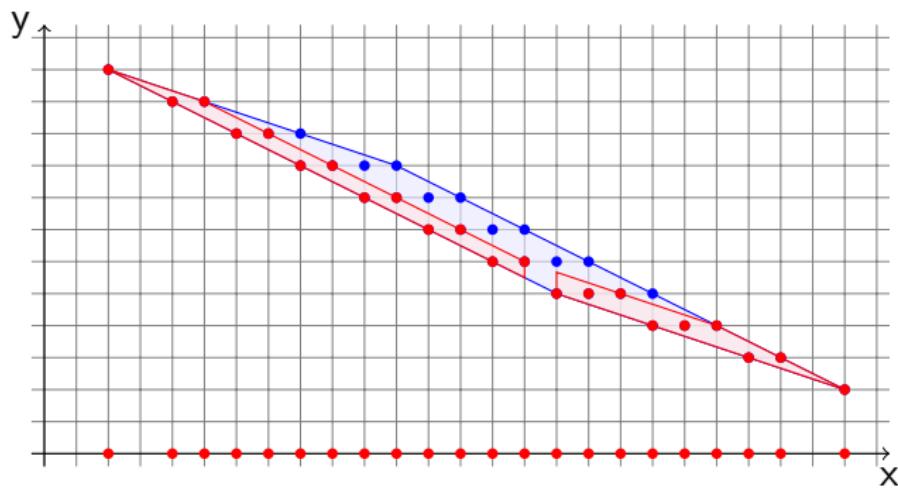
isl Operation: Lexicographic Minimization



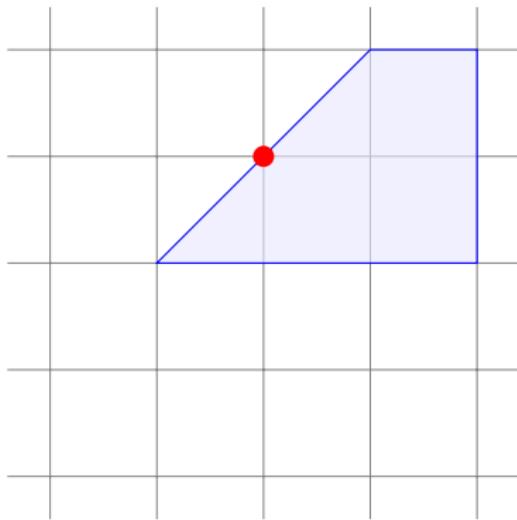
isl Operation: Integer Projection



isl Operation: Integer Projection

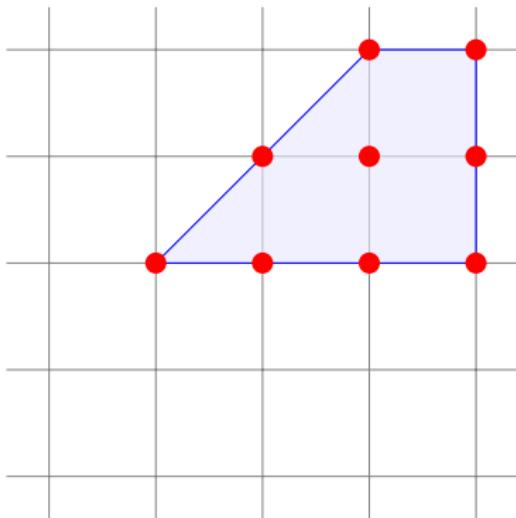


isl Operation: Sampling (Generalized Basis Reduction) [14]

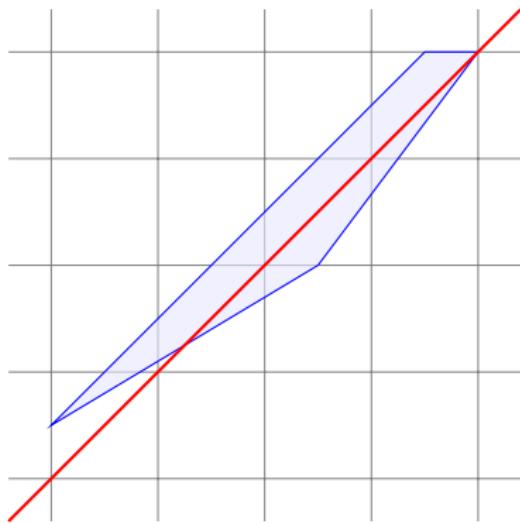


isl Operation: Scanning (Generalized Basis Reduction)

[14]



isl Operation: Integer Affine Hull



Outline

8 Polyhedra

9 Operations

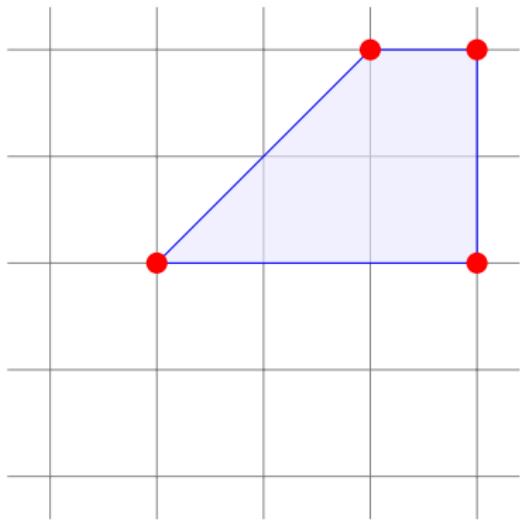
10 Vertex Enumeration

- Parametric Vertex Enumeration

11 Bounds on Quasi-Polynomials

12 Parametric Integer Programming

isl Operation: Vertex Enumeration



Non-parametric Vertex Enumeration

[2]

Popular vertex enumeration algorithms

- reverse search
 - ⇒ “graph traversal” method
 - ⇒ reverse pivoting of simplex method
 - ⇒ good for near simple polyhedra
(simple polyhedron: each vertex is adjacent to d edges)
 - ⇒ implemented in `lrs` (maybe also in `isl` one day)
- Motzkin’s double description method (aka Chernikova)
 - ⇒ “incremental” method
 - ⇒ maintains incidence information between facets and vertices
 - ⇒ implemented in PolyLib, PPL, ...
- Farkas + elimination of Farkas multipliers
 - ⇒ implemented in Pluto, omega(+) (convex hull), `isl`, ...

\mathcal{H} -Parametric Polytopes and their Vertices

Polytopes described by hyperplanes that depend linearly on parameters

$$P(\mathbf{s}) = \{ \mathbf{x} \in \mathbb{Q}^d : A\mathbf{x} + B\mathbf{s} \geq \mathbf{c} \}$$

Example:

$$P(N) = \{ (i, j) : i \geq 1 \wedge i \leq N \wedge j \geq 1 \wedge j \leq i \}$$

Parametric vertices:

$$P = \text{conv.hull} \left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} N \\ 1 \end{bmatrix}, \begin{bmatrix} N \\ N \end{bmatrix} \right\}$$

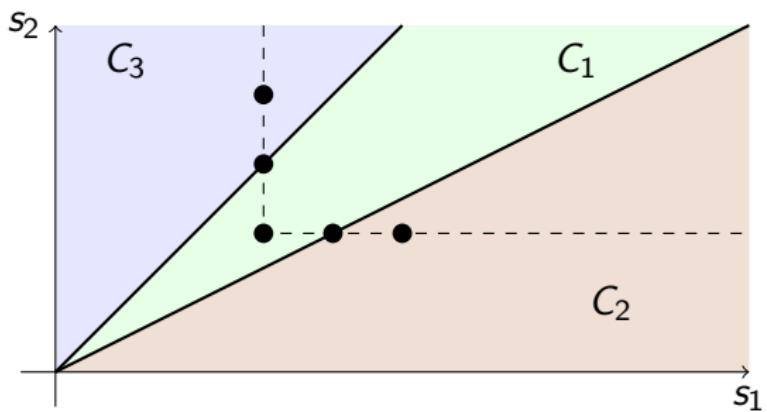
In general: different (active) vertices on different parts of the parameter space (chamber decomposition)

Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$

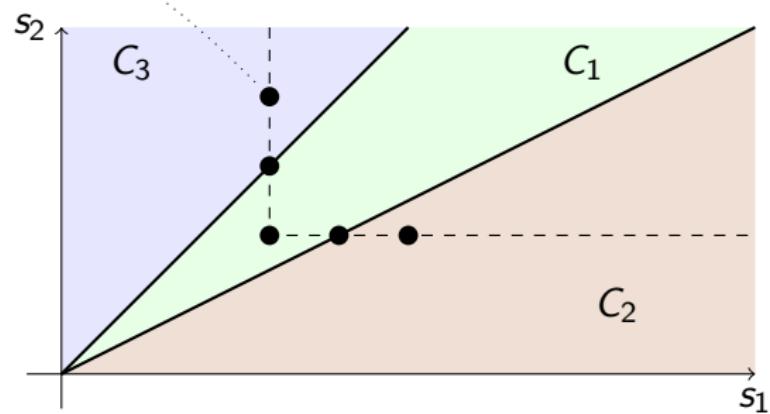
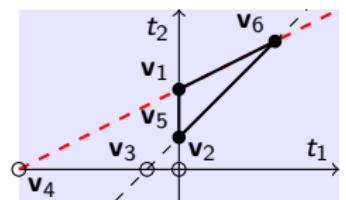
Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



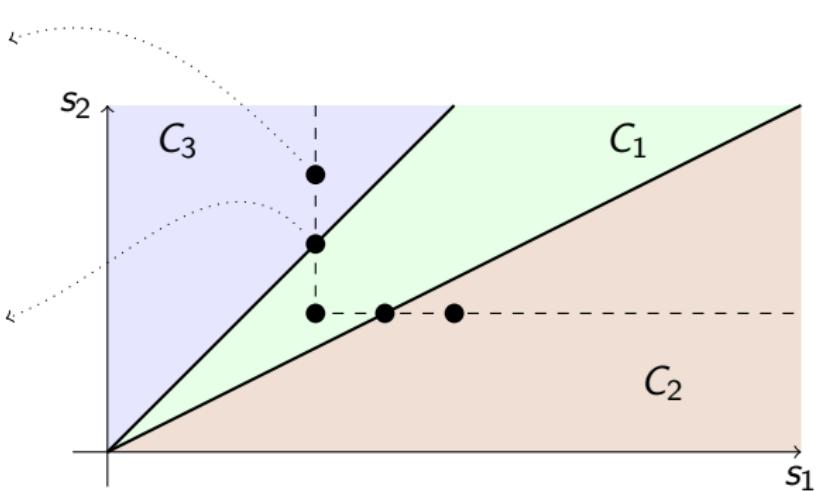
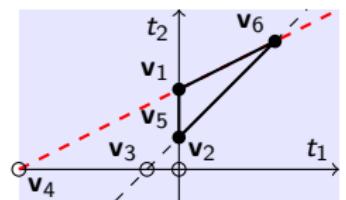
Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



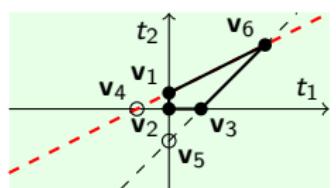
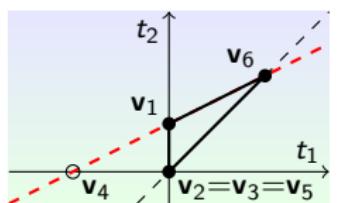
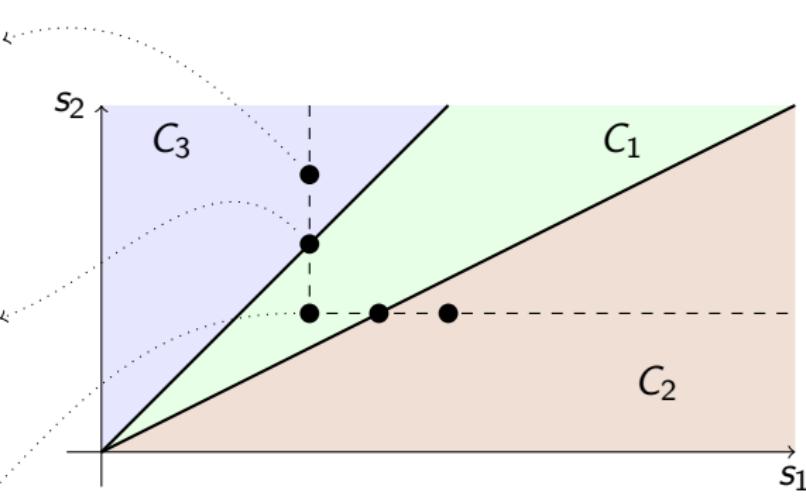
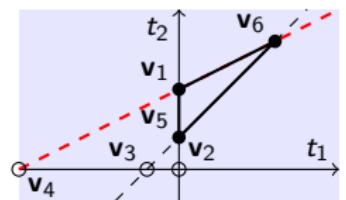
Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



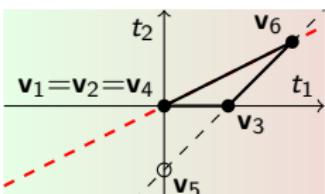
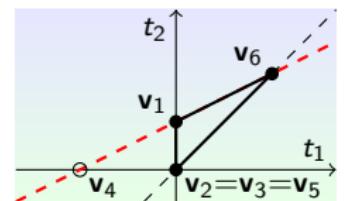
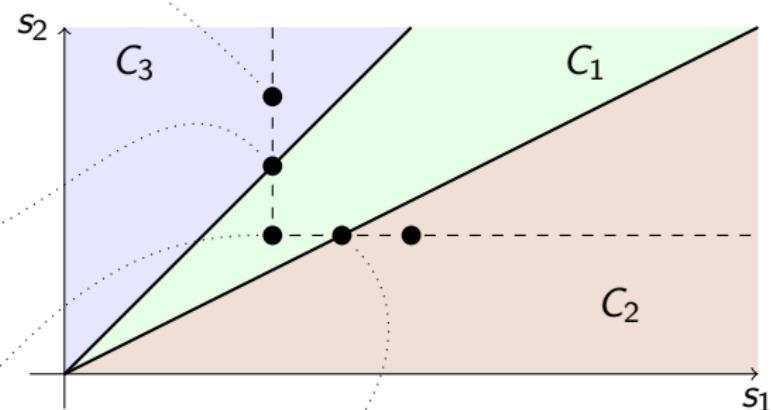
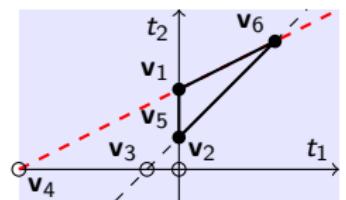
Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



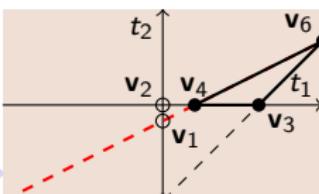
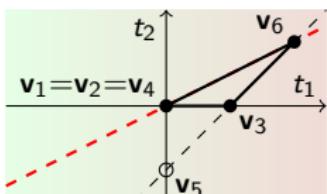
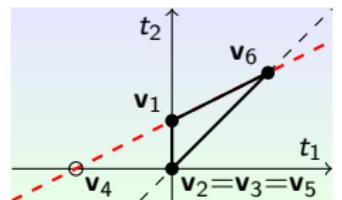
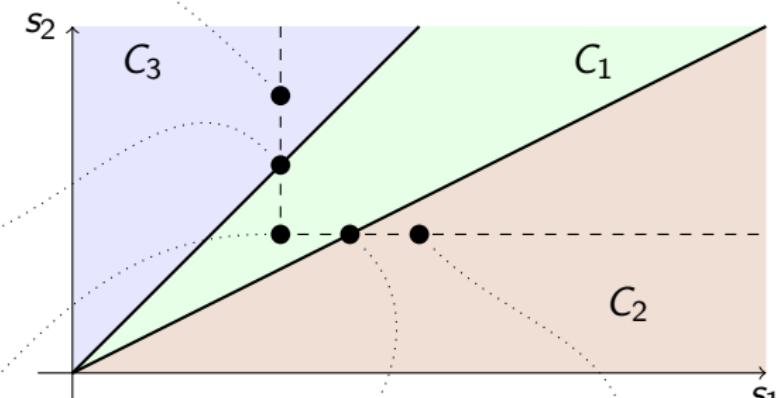
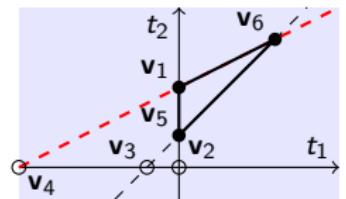
Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



Parametric Vertex Enumeration

- Vertex computation (simple “incremental” method)
 - ▶ Consider all combinations of d inequalities
 - ▶ Turn them into equalities
 - ▶ Record vertex and activity domain if non-empty

Parametric Vertex Enumeration

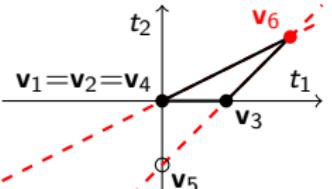
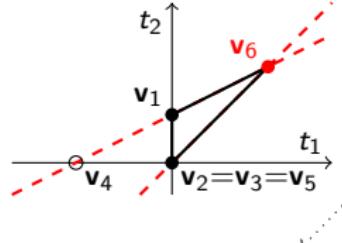
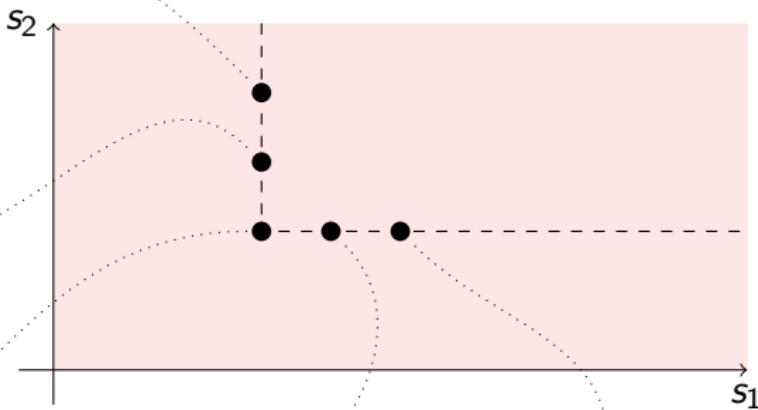
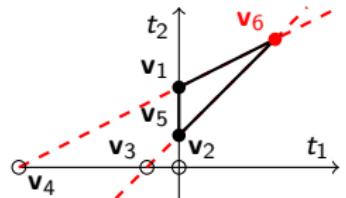
- Vertex computation (simple “incremental” method)
 - ▶ Consider all combinations of d inequalities
⇒ using backtracking and incremental LP solver
 - ▶ Turn them into equalities
 - ▶ Record vertex and activity domain if non-empty

Parametric Vertex Enumeration

- Vertex computation (simple “incremental” method)
 - ▶ Consider all combinations of d inequalities
 - ⇒ using backtracking and incremental LP solver
 - ▶ Turn them into equalities
 - ▶ Record vertex and activity domain if non-empty
 - ⇒ only record for lexmin inequalities

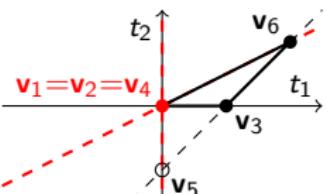
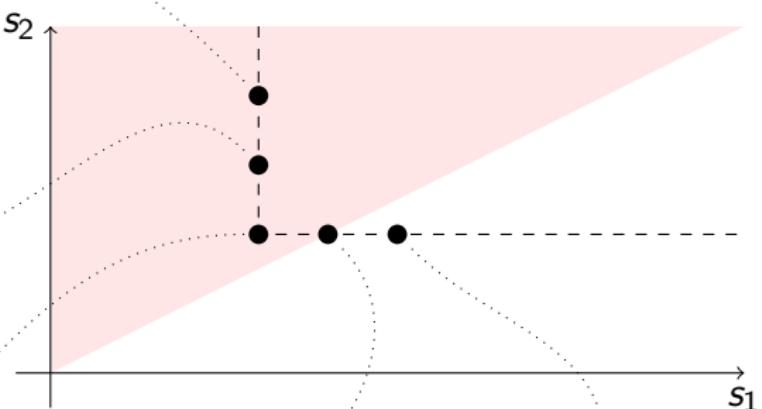
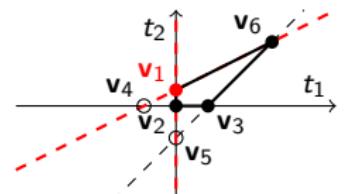
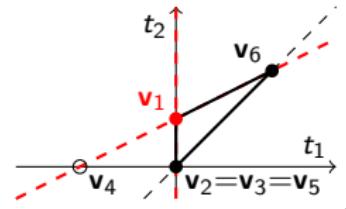
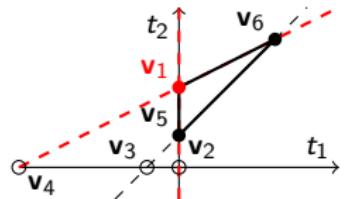
Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



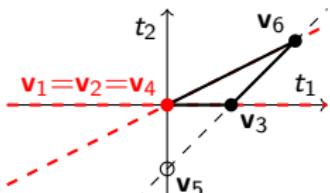
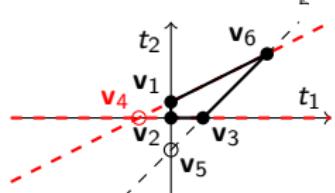
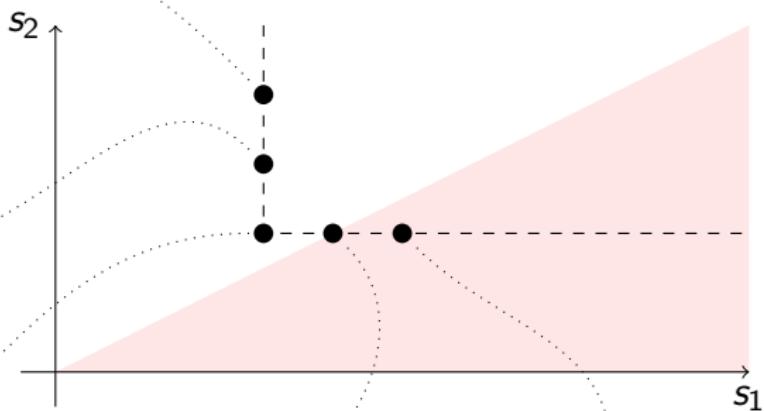
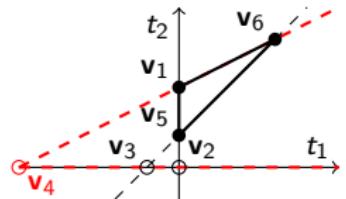
Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



Parametric Vertex Enumeration

- Vertex computation (simple “incremental” method)
 - ▶ Consider all combinations of d inequalities
 - ⇒ using backtracking and incremental LP solver
 - ▶ Turn them into equalities
 - ▶ Record vertex and activity domain if non-empty
 - ⇒ only record for lexmin inequalities

Parametric Vertex Enumeration

- Vertex computation (simple “incremental” method)
 - ▶ Consider all combinations of d inequalities
 - ⇒ using backtracking and incremental LP solver
 - ▶ Turn them into equalities
 - ▶ Record vertex and activity domain if non-empty
 - ⇒ only record for lexmin inequalities
- Chamber decomposition (note: only full-dimensional chambers)
PolyLib:
 - ▶ iterate over all activity domains
 - ▶ compute differences and intersections with previous activity domains

Parametric Vertex Enumeration

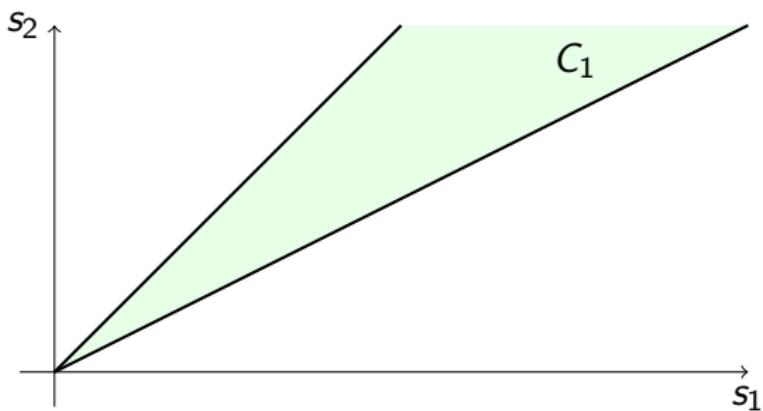
- Vertex computation (simple “incremental” method)
 - ▶ Consider all combinations of d inequalities
 - ⇒ using backtracking and incremental LP solver
 - ▶ Turn them into equalities
 - ▶ Record vertex and activity domain if non-empty
 - ⇒ only record for lexmin inequalities
- Chamber decomposition (note: only full-dimensional chambers)
PolyLib:
 - ▶ iterate over all activity domains
 - ▶ compute differences and intersections with previous activity domains

isl:

- ▶ compute initial chamber (intersection of activity domains)
- ▶ pick unhandled internal facet
- ▶ intersect activity domains that contain facet and other side
 - ⇒ new chamber
- ▶ repeat while there are unhandled internal facets

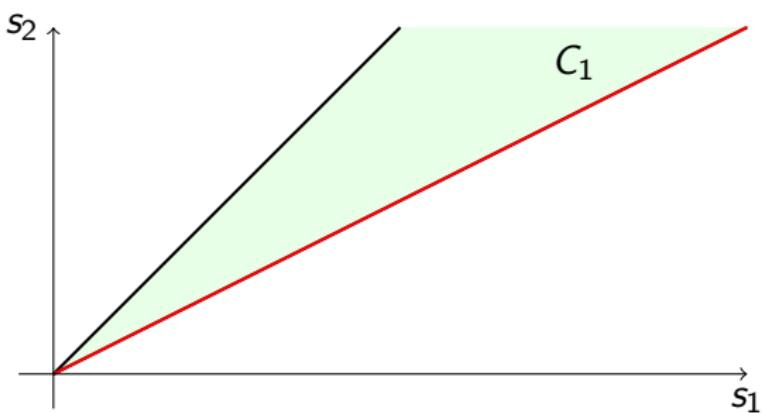
Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



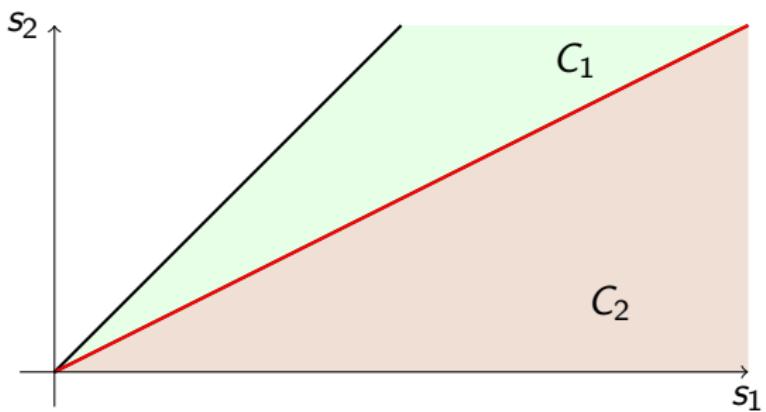
Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



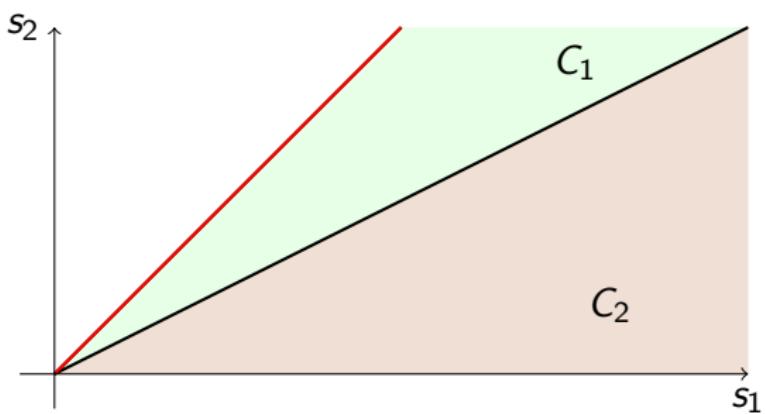
Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



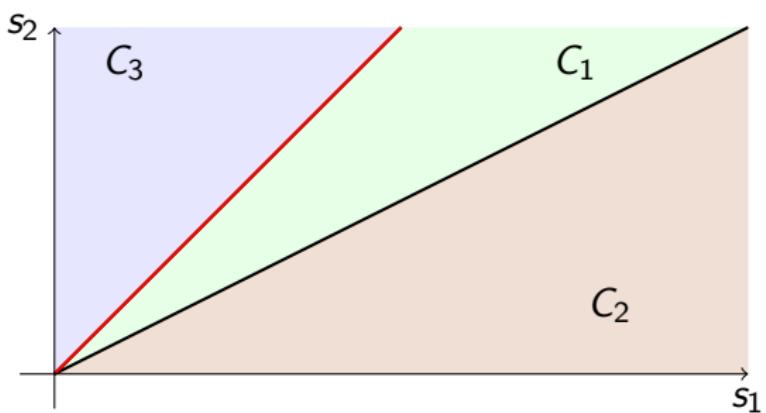
Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



Chamber Decomposition

$$\{ \mathbf{t} \in \mathbb{Q}^2 : -s_1 + 2s_2 + t_1 - 2t_2 \geq 0 \wedge s_1 - s_2 - t_1 + t_2 \geq 0 \wedge t_1 \geq 0 \wedge t_2 \geq 0 \}$$



Parametric Vertex Enumeration

- Vertex computation (simple “incremental” method)
 - ▶ Consider all combinations of d inequalities
 - ⇒ using backtracking and incremental LP solver
 - ▶ Turn them into equalities
 - ▶ Record vertex and activity domain if non-empty
 - ⇒ only record for lexmin inequalities
- Chamber decomposition (note: only full-dimensional chambers)

PolyLib:

- ▶ iterate over all activity domains
- ▶ compute differences and intersections with previous activity domains

isl:

- ▶ compute initial chamber (intersection of activity domains)
- ▶ pick unhandled internal facet
- ▶ intersect activity domains that contain facet and other side
 - ⇒ new chamber
- ▶ repeat while there are unhandled internal facets

Parametric Vertex Enumeration

- Vertex computation (simple “incremental” method)
 - ▶ Consider all combinations of d inequalities
 - ⇒ using backtracking and incremental LP solver
 - ▶ Turn them into equalities
 - ▶ Record vertex and activity domain if non-empty
 - ⇒ only record for lexmin inequalities
- Chamber decomposition (note: only full-dimensional chambers)

PolyLib:

- ▶ iterate over all activity domains
- ▶ compute **differences** and intersections with previous activity domains

isl:

- ▶ compute initial chamber (intersection of activity domains)
- ▶ pick unhandled internal facet
- ▶ intersect activity domains that contain facet and other side
 - ⇒ new chamber
- ▶ repeat while there are unhandled internal facets

Parametric Vertex Enumeration

- Vertex computation (simple “incremental” method)
 - ▶ Consider all combinations of d inequalities
 - ⇒ using backtracking and incremental LP solver
 - ▶ Turn them into equalities
 - ▶ Record vertex and activity domain if non-empty
 - ⇒ only record for lexmin inequalities
- Chamber decomposition (note: only full-dimensional chambers)

PolyLib:

- ▶ iterate over all activity domains
- ▶ compute **differences** and intersections with previous activity domains

isl:

- ▶ compute initial chamber (intersection of activity domains)
- ▶ pick unhandled internal facet
- ▶ intersect activity domains that contain facet and other side
 - ⇒ new chamber
- ▶ repeat while there are unhandled internal facets

⇒ much faster than PolyLib; similar to TOPCOM 0.16.2

Outline

8 Polyhedra

9 Operations

10 Vertex Enumeration

- Parametric Vertex Enumeration

11 Bounds on Quasi-Polynomials

12 Parametric Integer Programming

\mathcal{V} -Parametric Polytopes and Bounds on Polynomials

- \mathcal{V} -parametric polytopes

$$P : D \rightarrow \mathbb{Q}^n :$$

$$\mathbf{q} \mapsto P(\mathbf{q}) = \{ \mathbf{x} \mid \exists \alpha_i \in \mathbb{Q} : \mathbf{x} = \sum_i \alpha_i \mathbf{v}_i(\mathbf{q}), \alpha_i \geq 0, \sum_i \alpha_i = 1 \}$$

$D \subset \mathbb{Q}^r$: parameter domain

$\mathbf{v}_i(\mathbf{q}) \in \mathbb{Q}[\mathbf{q}]$ arbitrary polynomials in parameters

$\mathbf{v}_i(\mathbf{q})$ are generators of the polytope

Note: \mathcal{V} -parametric polytope can be computed from \mathcal{H} -parametric polytope through parameter vertex enumeration + chamber decomposition

\mathcal{V} -Parametric Polytopes and Bounds on Polynomials

- \mathcal{V} -parametric polytopes

$$P : D \rightarrow \mathbb{Q}^n :$$

$$\mathbf{q} \mapsto P(\mathbf{q}) = \{ \mathbf{x} \mid \exists \alpha_i \in \mathbb{Q} : \mathbf{x} = \sum_i \alpha_i \mathbf{v}_i(\mathbf{q}), \alpha_i \geq 0, \sum_i \alpha_i = 1 \}$$

$D \subset \mathbb{Q}^r$: parameter domain

$\mathbf{v}_i(\mathbf{q}) \in \mathbb{Q}[\mathbf{q}]$ arbitrary polynomials in parameters

$\mathbf{v}_i(\mathbf{q})$ are generators of the polytope

- Bounds on quasipolynomials (CFGV2009)

Input: Parametric polytope P and quasipolynomial $p(\mathbf{q}, \mathbf{x})$

Output: Bound $B(\mathbf{q})$ on quasipolynomial over polytope

$$B(\mathbf{q}) \geq \max_{\mathbf{x} \in P(\mathbf{q})} p(\mathbf{q}, \mathbf{x})$$

Note: \mathcal{V} -parametric polytope can be computed from \mathcal{H} -parametric polytope through parameter vertex enumeration + chamber decomposition

Bounds on Quasipolynomials: Example

$$p(x_1, x_2) = \frac{1}{2}x_1^2 + \frac{1}{2}x_1 + x_2 \quad P = \text{conv.hull}\{(0,0), (N,0), (N,N)\}$$

To compute:

$$M(N) = \max_{(x_1, x_2) \in P} p(x_1, x_2)$$

Bounds on Quasipolynomials: Example

$$p(x_1, x_2) = \frac{1}{2}x_1^2 + \frac{1}{2}x_1 + x_2 \quad P = \text{conv.hull}\{(0,0), (N,0), (N,N)\}$$

To compute:

$$B(N) \geq M(N) = \max_{(x_1, x_2) \in P} p(x_1, x_2)$$

Bounds on Quasipolynomials: Example

$$p(x_1, x_2) = \frac{1}{2}x_1^2 + \frac{1}{2}x_1 + x_2 \quad P = \text{conv.hull}\{(0,0), (N,0), (N,N)\}$$

To compute:

$$B(N) \geq M(N) = \max_{(x_1, x_2) \in P} p(x_1, x_2)$$

How? \Rightarrow Bernstein expansion

- Express $x \in P$ as convex combination of vertices

$$(x_1, x_2) = \alpha_1(0,0) + \alpha_2(N,0) + \alpha_3(N,N), \quad \alpha_i \geq 0, \quad \sum_i \alpha_i = 1$$

$$p(\alpha_1, \alpha_2, \alpha_3) = \frac{1}{2}N^2\alpha_2^2 + N^2\alpha_2\alpha_3 + \frac{1}{2}N^2\alpha_3^2 + \frac{1}{2}N\alpha_2 + \frac{3}{2}N\alpha_3$$

- Express $p(x)$ as convex combination of polynomials in parameters

Bounds on Quasipolynomials: Example

$$p(\alpha) = \frac{N^2}{2}\alpha_2^2 + N^2\alpha_2\alpha_3 + \frac{N^2}{2}\alpha_3^2 + \frac{N}{2}\alpha_2 + \frac{3N}{2}\alpha_3 \quad \alpha_i \geq 0, \quad \sum_i \alpha_i = 1$$

- Express $p(\mathbf{x})$ as convex combination of polynomials in parameters

$$p(\mathbf{x}) = \sum B_j(\alpha) b_j(\mathbf{N})$$

Bounds on Quasipolynomials: Example

$$p(\alpha) = \frac{N^2}{2}\alpha_2^2 + N^2\alpha_2\alpha_3 + \frac{N^2}{2}\alpha_3^2 + \frac{N}{2}\alpha_2 + \frac{3N}{2}\alpha_3 \quad \alpha_i \geq 0, \quad \sum_i \alpha_i = 1$$

- Express $p(\mathbf{x})$ as convex combination of polynomials in parameters

$$\min_j b_j(\mathbf{N}) \leq p(\mathbf{x}) = \sum B_j(\alpha) b_j(\mathbf{N}) \leq \max_j b_j(\mathbf{N})$$

Bounds on Quasipolynomials: Example

$$p(\alpha) = \frac{N^2}{2}\alpha_2^2 + N^2\alpha_2\alpha_3 + \frac{N^2}{2}\alpha_3^2 + \frac{N}{2}\alpha_2 + \frac{3N}{2}\alpha_3 \quad \alpha_i \geq 0, \quad \sum_i \alpha_i = 1$$

- Express $p(\mathbf{x})$ as convex combination of polynomials in parameters

$$\min_j b_j(\mathbf{N}) \leq p(\mathbf{x}) = \sum B_j(\alpha) b_j(\mathbf{N}) \leq \max_j b_j(\mathbf{N})$$

$$1 = (\alpha_1 + \alpha_2 + \alpha_3)^2 = \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + 2\alpha_1\alpha_2 + 2\alpha_3\alpha_3 + 2\alpha_3\alpha_1$$

Bounds on Quasipolynomials: Example

$$p(\alpha) = \frac{N^2}{2}\alpha_2^2 + N^2\alpha_2\alpha_3 + \frac{N^2}{2}\alpha_3^2 + \frac{N}{2}\alpha_2 + \frac{3N}{2}\alpha_3 \quad \alpha_i \geq 0, \quad \sum_i \alpha_i = 1$$

- Express $p(\mathbf{x})$ as convex combination of polynomials in parameters

$$\min_j b_j(\mathbf{N}) \leq p(\mathbf{x}) = \sum B_j(\alpha) b_j(\mathbf{N}) \leq \max_j b_j(\mathbf{N})$$

$$1 = (\alpha_1 + \alpha_2 + \alpha_3)^2 = \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + 2\alpha_1\alpha_2 + 2\alpha_3\alpha_3 + 2\alpha_3\alpha_1$$

$$\begin{aligned} p(\alpha_1, \alpha_2, \alpha_3) &= \alpha_1^2 0 + \alpha_2^2 \left(\frac{N^2 + N}{2} \right) + \alpha_3^2 \left(\frac{N^2 + 3N}{2} \right) \\ &\quad + (2\alpha_1\alpha_2) \frac{N}{4} + (2\alpha_1\alpha_3) \frac{3N}{2} + (2\alpha_2\alpha_3) \frac{N^2 + 2N}{2} \end{aligned}$$

Bounds on Quasipolynomials: Example

$$p(\alpha) = \frac{N^2}{2}\alpha_2^2 + N^2\alpha_2\alpha_3 + \frac{N^2}{2}\alpha_3^2 + \frac{N}{2}\alpha_2 + \frac{3N}{2}\alpha_3 \quad \alpha_i \geq 0, \quad \sum_i \alpha_i = 1$$

- Express $p(\mathbf{x})$ as convex combination of polynomials in parameters

$$\min_j b_j(\mathbf{N}) \leq p(\mathbf{x}) = \sum B_j(\alpha) b_j(\mathbf{N}) \leq \max_j b_j(\mathbf{N})$$

$$1 = (\alpha_1 + \alpha_2 + \alpha_3)^2 = \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + 2\alpha_1\alpha_2 + 2\alpha_3\alpha_3 + 2\alpha_3\alpha_1$$

$$\begin{aligned} p(\alpha_1, \alpha_2, \alpha_3) &= \alpha_1^2 \textcolor{red}{0} + \alpha_2^2 \left(\frac{N^2 + N}{2} \right) + \alpha_3^2 \left(\frac{\textcolor{blue}{N^2} + 3N}{2} \right) \\ &\quad + (2\alpha_1\alpha_2) \frac{N}{4} + (2\alpha_1\alpha_3) \frac{3N}{2} + (2\alpha_2\alpha_3) \frac{N^2 + 2N}{2} \end{aligned}$$

Outline

8 Polyhedra

9 Operations

10 Vertex Enumeration

- Parametric Vertex Enumeration

11 Bounds on Quasi-Polynomials

12 Parametric Integer Programming

Parametric Integer Programming

[17]

Assume:

- we want to compute the lexicographic *minimum*
- all unknown are non-negative
⇒ constraints $x_i \geq 0$ are (implicitly) imposed

Parametric Integer Programming

[17]

Assume:

- we want to compute the lexicographic *minimum*
- all unknown are non-negative
⇒ constraints $x_i \geq 0$ are (implicitly) imposed

Dual simplex (operations performed on simplex tableau)

- start from basic solution **0** (intersection of constraints $x_i \geq 0$)
- as long as there is any violated constraint, move to lexico-larger adjacent basic solution
(simple bookkeeping ensures lexico-positive move is always possible, if there is a solution)
- first feasible basic solution found is lexico-minimal solution

Parametric Integer Programming

[17]

Assume:

- we want to compute the lexicographic *minimum*
- all unknown are non-negative
⇒ constraints $x_i \geq 0$ are (implicitly) imposed

Dual simplex (operations performed on simplex tableau)

- start from basic solution **0** (intersection of constraints $x_i \geq 0$)
- as long as there is any violated constraint, move to lexico-larger adjacent basic solution
(simple bookkeeping ensures lexico-positive move is always possible, if there is a solution)
- first feasible basic solution found is lexico-minimal solution

Gomory cuts

- extra constraints added to exclude rational solutions
- may involve “new parameters” (extra integer divisions)

Parametric Integer Programming: Example

[17]

$$a : \quad x \geq y + 1$$

$$b : \quad 2y \geq p$$

$$c : \quad x \geq 0$$

$$d : \quad y \geq 0$$

Parametric Integer Programming: Example

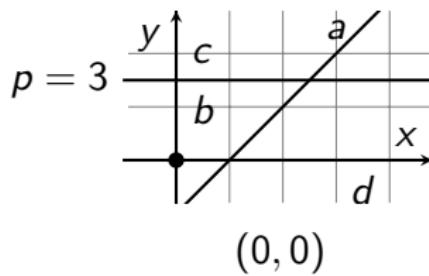
[17]

$$a : \quad x \geq y + 1$$

$$b : \quad 2y \geq p$$

$$c : \quad x \geq 0$$

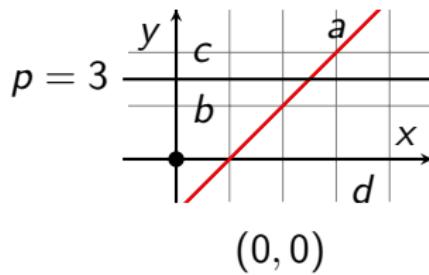
$$d : \quad y \geq 0$$



Parametric Integer Programming: Example

[17]

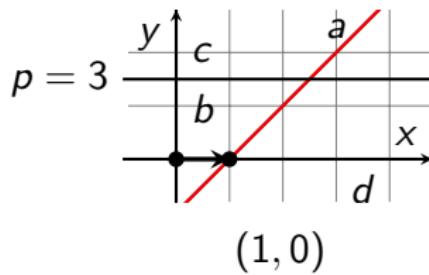
$a :$	$x \geq y + 1$	$0 \geq 1$	$-$
$b :$	$2y \geq p$	$0 \geq p$	$+ / -$
$c :$	$x \geq 0$	$0 \geq 0$	$+$
$d :$	$y \geq 0$	$0 \geq 0$	$+$



Parametric Integer Programming: Example

[17]

$a :$	$x \geq y + 1$	$0 \geq 1$	$-$
$b :$	$2y \geq p$	$0 \geq p$	$+ / -$
$c :$	$x \geq 0$	$0 \geq 0$	$+$
$d :$	$y \geq 0$	$0 \geq 0$	$+$



Parametric Integer Programming: Example

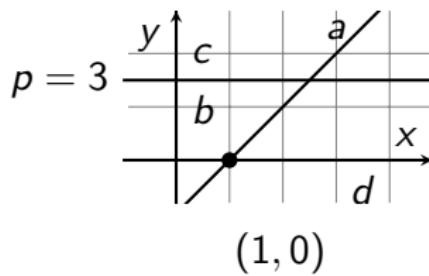
[17]

$$a : \quad x \geq y + 1$$

$$b : \quad 2y \geq p$$

$$c : \quad x \geq 0$$

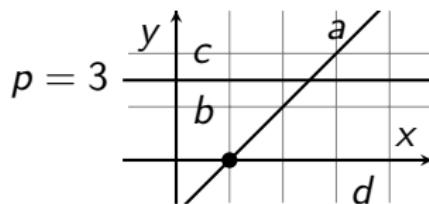
$$d : \quad y \geq 0$$



Parametric Integer Programming: Example

[17]

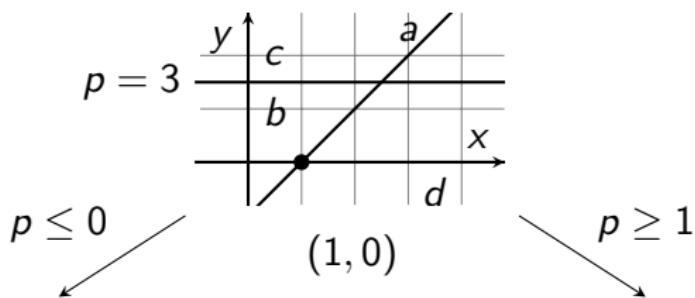
$a :$	$x \geq y + 1$	$1 \geq 1$	$+$
$b :$	$2y \geq p$	$0 \geq p$	$+ / -$
$c :$	$x \geq 0$	$1 \geq 0$	$+$
$d :$	$y \geq 0$	$0 \geq 0$	$+$

 $(1, 0)$

Parametric Integer Programming: Example

[17]

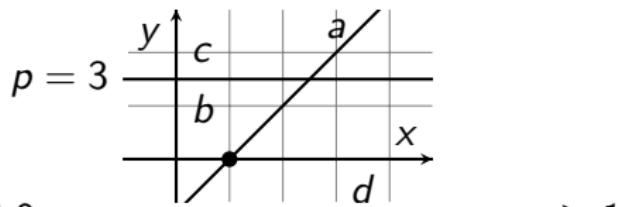
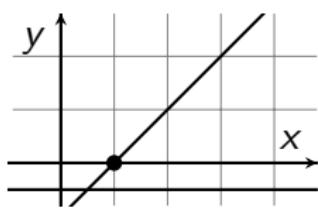
$a :$	$x \geq y + 1$	$1 \geq 1$	$+$
$b :$	$2y \geq p$	$0 \geq p$	$+ / -$
$c :$	$x \geq 0$	$1 \geq 0$	$+$
$d :$	$y \geq 0$	$0 \geq 0$	$+$



Parametric Integer Programming: Example

[17]

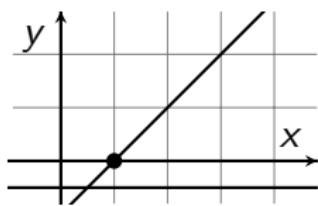
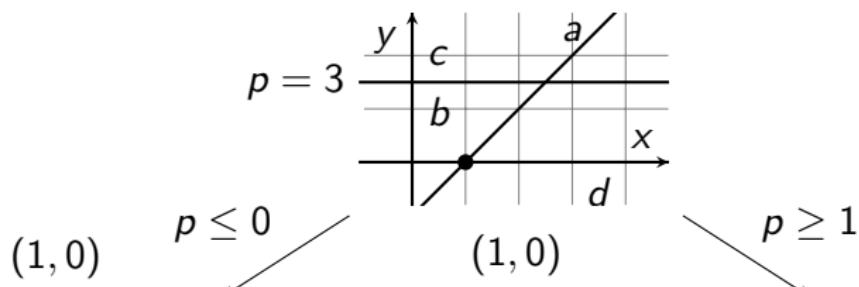
$a :$	$x \geq y + 1$	$1 \geq 1$	$+$
$b :$	$2y \geq p$	$0 \geq p$	$+ / -$
$c :$	$x \geq 0$	$1 \geq 0$	$+$
$d :$	$y \geq 0$	$0 \geq 0$	$+$

 $(1, 0)$ $p \leq 0$ $(1, 0)$ $p \geq 1$ 

Parametric Integer Programming: Example

[17]

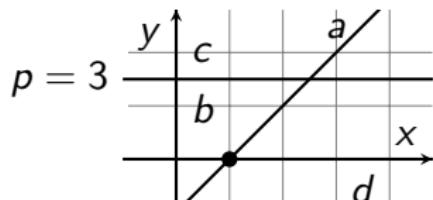
$$\begin{array}{llll} a : & x \geq y + 1 & 1 \geq 1 & + \\ b : & 2y \geq p & 0 \geq p & + \\ c : & x \geq 0 & 1 \geq 0 & + \\ d : & y \geq 0 & 0 \geq 0 & + \end{array}$$



Parametric Integer Programming: Example

[17]

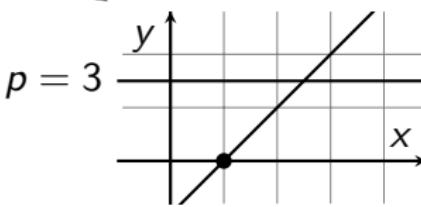
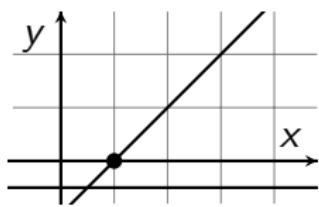
$a :$	$x \geq y + 1$	$1 \geq 1$	$+$
$b :$	$2y \geq p$	$0 \geq p$	$+ / -$
$c :$	$x \geq 0$	$1 \geq 0$	$+$
$d :$	$y \geq 0$	$0 \geq 0$	$+$



$(1, 0)$ $p \leq 0$

$(1, 0)$

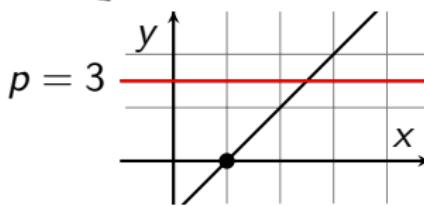
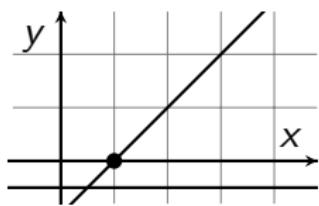
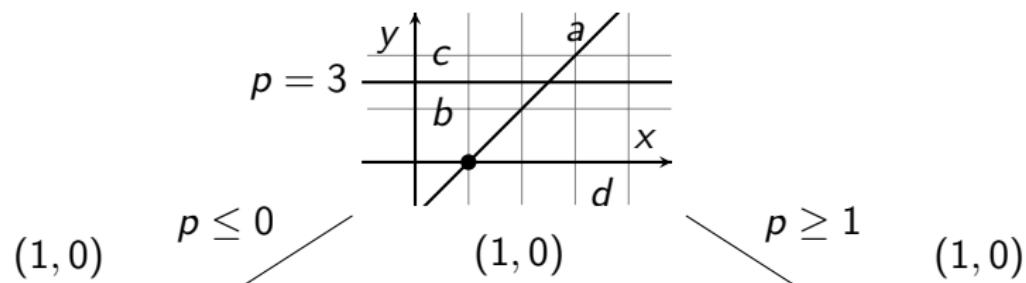
$p \geq 1$ $(1, 0)$



Parametric Integer Programming: Example

[17]

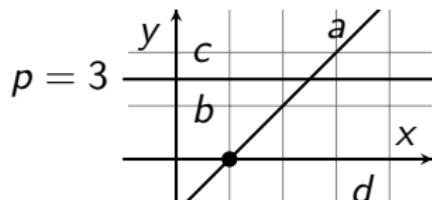
$$\begin{array}{llll} a : & x \geq y + 1 & 1 \geq 1 & + \\ b : & 2y \geq p & 0 \geq p & - \\ c : & x \geq 0 & 1 \geq 0 & + \\ d : & y \geq 0 & 0 \geq 0 & + \end{array}$$



Parametric Integer Programming: Example

[17]

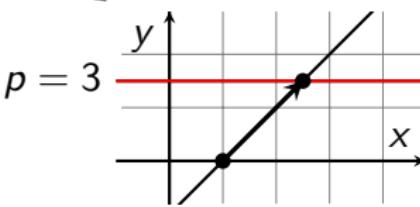
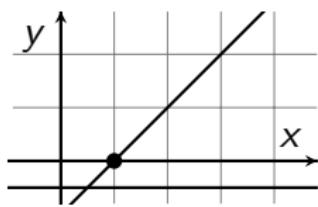
$a :$	$x \geq y + 1$	$1 \geq 1$	$+$
$b :$	$2y \geq p$	$0 \geq p$	$-$
$c :$	$x \geq 0$	$1 \geq 0$	$+$
$d :$	$y \geq 0$	$0 \geq 0$	$+$



$$(1, 0) \quad p \leq 0$$

$$(1, 0)$$

$$p \geq 1 \quad (1 + p/2, p/2)$$



Parametric Integer Programming: Example

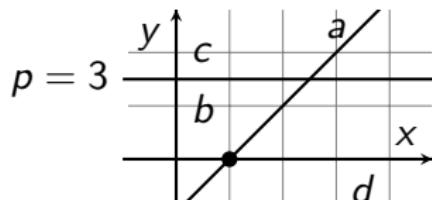
[17]

$$a : \quad x \geq y + 1$$

$$b : \quad 2y \geq p$$

$$c : \quad x \geq 0$$

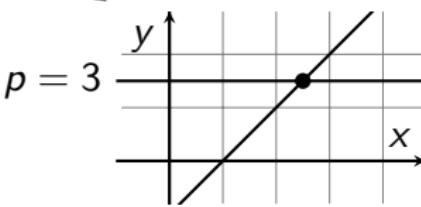
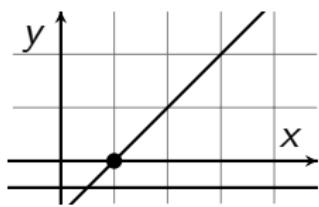
$$d : \quad y \geq 0$$



$$(1, 0) \quad p \leq 0$$

$$(1, 0)$$

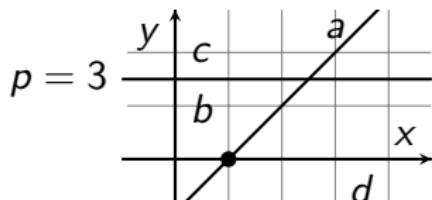
$$p \geq 1 \quad (1 + p/2, p/2)$$



Parametric Integer Programming: Example

[17]

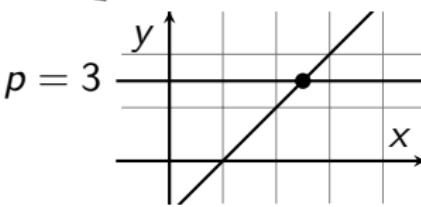
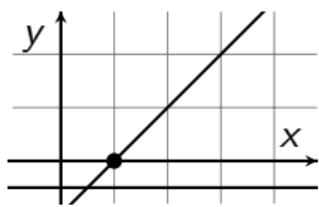
$$\begin{array}{lll} a : & x \geq y + 1 & 1 + p/2 \geq p/2 + 1 \\ b : & 2y \geq p & p \geq p \\ c : & x \geq 0 & 1 + p/2 \geq 0 \\ d : & y \geq 0 & p/2 \geq 0 \end{array} + + + +$$



$$(1, 0) \quad p \leq 0$$

$$(1, 0)$$

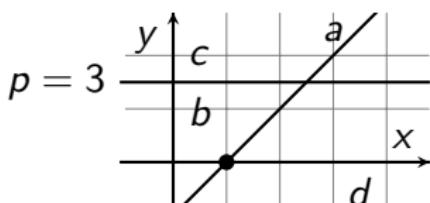
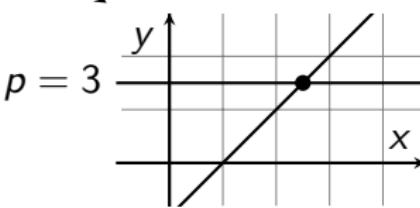
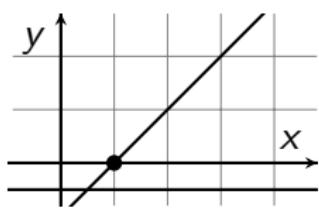
$$p \geq 1 \quad (1 + p/2, p/2)$$



Parametric Integer Programming: Example

[17]

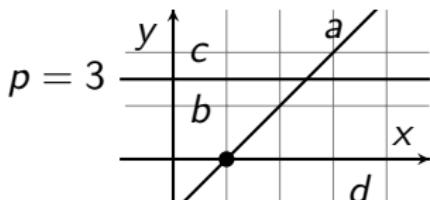
$$\begin{array}{lll} a : & x \geq y + 1 & 1 + p/2 \geq p/2 + 1 \\ b : & 2y \geq p & p \geq p \\ c : & x \geq 0 & 1 + p/2 \geq 0 \\ d : & y \geq 0 & p/2 \geq 0 \end{array} + + + +$$

 $(1, 0) \quad p \leq 0$ $(1, 0)$ $p \geq 1 \quad (1 + p/2, p/2)$ 

Parametric Integer Programming: Example

[17]

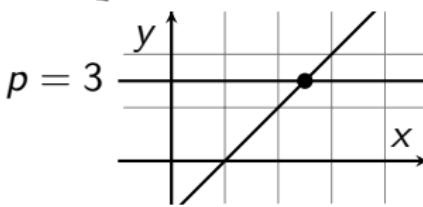
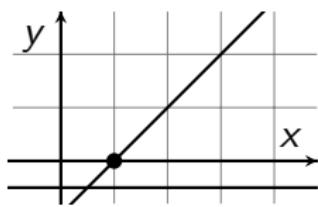
$a :$	$x \geq y + 1$	$1 + p/2 \geq p/2 + 1$	$+$
$b :$	$2y \geq p$	$p \geq p$	$+$
$c :$	$x \geq 0$	$1 + p/2 \geq 0$	$+$
$d :$	$y \geq 0$	$p/2 \geq 0$	$+$
$e :$	$y + \lfloor p/2 \rfloor \geq p$	$p/2 + \lfloor p + 2 \rfloor \geq p$	$-$



$(1, 0)$ $p \leq 0$

$(1, 0)$

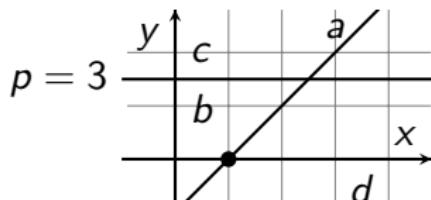
$p \geq 1$
 $(1 + p/2, p/2)$



Parametric Integer Programming: Example

[17]

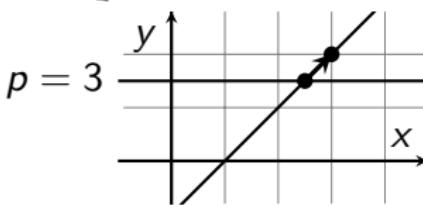
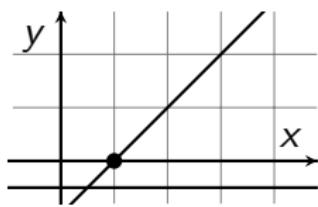
$a :$	$x \geq y + 1$	$1 + p/2 \geq p/2 + 1$	$+$
$b :$	$2y \geq p$	$p \geq p$	$+$
$c :$	$x \geq 0$	$1 + p/2 \geq 0$	$+$
$d :$	$y \geq 0$	$p/2 \geq 0$	$+$
$e :$	$y + \lfloor p/2 \rfloor \geq p$	$p/2 + \lfloor p + 2 \rfloor \geq p$	$-$



$$(1, 0) \quad p \leq 0$$

$$(1, 0)$$

$$p \geq 1 \quad (1+p-\lfloor p/2 \rfloor, p-\lfloor p/2 \rfloor)$$



Parametric Integer Programming: Example

[17]

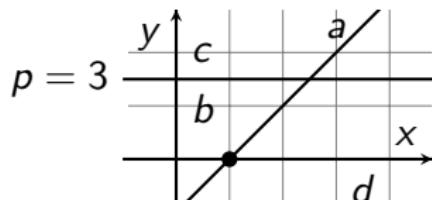
$$a : \quad x \geq y + 1$$

$$b : \quad 2y \geq p$$

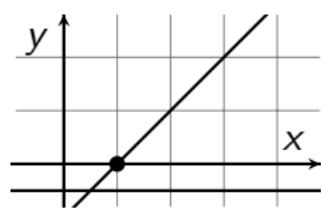
$$c : \quad x \geq 0$$

$$d : \quad y \geq 0$$

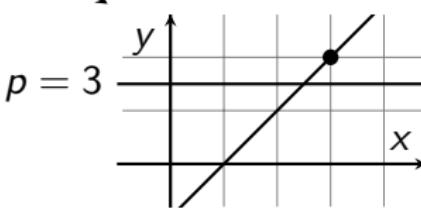
$$e : \quad y + \lfloor p/2 \rfloor \geq p$$



$$(1, 0) \quad p \leq 0$$



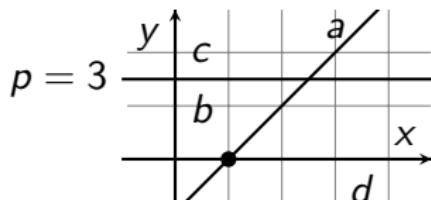
$$(1, 0) \quad p \geq 1 \quad (1+p-\lfloor p/2 \rfloor, p-\lfloor p/2 \rfloor)$$



Parametric Integer Programming: Example

[17]

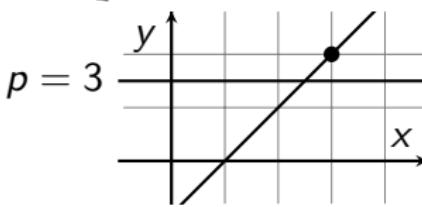
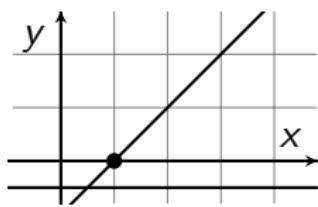
$a :$	$x \geq y + 1$	$1 + p - \lfloor p/2 \rfloor \geq p - \lfloor p/2 \rfloor + 1$	+
$b :$	$2y \geq p$	$2p - 2 \lfloor p/2 \rfloor \geq p$	+
$c :$	$x \geq 0$	$1 + p - \lfloor p/2 \rfloor \geq 0$	+
$d :$	$y \geq 0$	$p - \lfloor p/2 \rfloor \geq 0$	+
$e :$	$y + \lfloor p/2 \rfloor \geq p$	$p - \lfloor p/2 \rfloor + \lfloor p/2 \rfloor \geq p$	+



$(1, 0)$ $p \leq 0$

$(1, 0)$

$p \geq 1$
 $(1+p-\lfloor p/2 \rfloor, p-\lfloor p/2 \rfloor)$



Parametric Integer Programming

[17]

Parameters are assumed to be integral

Set of parameter values is called the *context*

Determining the sign (+, -, +/−) of a constraint

- + there are no values of the parameters where constraint is violated
- there are no values of the parameters where constraint is not violated
- +/- constraint is violated for only some values of the parameters

⇒ (non-parametric) integer *feasibility* problems

Parametric Integer Programming

[17]

Parameters are assumed to be integral

Set of parameter values is called the *context*

Determining the sign (+, -, +/−) of a constraint

- + there are no values of the parameters where constraint is violated
- there are no values of the parameters where constraint is not violated
- +/- constraint is violated for only some values of the parameters

⇒ (non-parametric) integer *feasibility* problems

piplib computes lexico-minimal value

- found ⇒ problem feasible
- not found ⇒ problem infeasible

Parametric Integer Programming

[17]

Basic method starts from **0**, assuming unknowns are non-negative

How to deal with unknowns/parameters of unrestricted sign?

⇒ “big parameter” trick

- A special parameter M is assumed to be arbitrarily large (think ∞)
- Problem is rewritten in terms of $\mathbf{x}' = M\mathbf{1} + \mathbf{x}$
- Initial value $\mathbf{x}' = \mathbf{0}$ corresponds to $\mathbf{x} = -M\mathbf{1}$ (think $\mathbf{x} = -\infty$)

Parametric Integer Programming

[17]

Basic method starts from $\mathbf{0}$, assuming unknowns are non-negative

How to deal with unknowns/parameters of unrestricted sign?

⇒ “big parameter” trick

- A special parameter M is assumed to be arbitrarily large (think ∞)
- Problem is rewritten in terms of $\mathbf{x}' = M\mathbf{1} + \mathbf{x}$
- Initial value $\mathbf{x}' = \mathbf{0}$ corresponds to $\mathbf{x} = -M\mathbf{1}$ (think $\mathbf{x} = -\infty$)

Note: piplib uses *different* trick in context tableau

⇒ p is replaced by $p^+ - p^-$

Duality of Cones

Primal

 C

$$\rightarrow C^* = \{ \mathbf{y} : \forall \mathbf{x} \in C : \langle \mathbf{y}, \mathbf{x} \rangle \geq 0 \}$$

valid constraints

 \rightarrow

Dual

rays

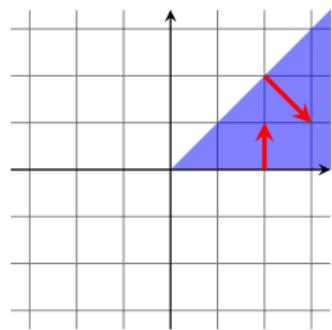
Duality of Cones

Primal

$$C$$

valid constraints

facet constraints



Dual

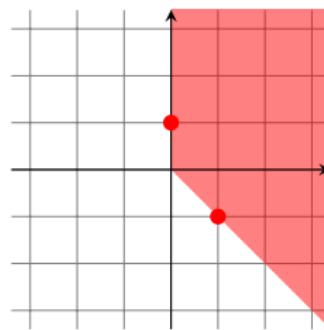
$$C^* = \{ \mathbf{y} : \forall \mathbf{x} \in C : \langle \mathbf{y}, \mathbf{x} \rangle \geq 0 \}$$

→

rays

→

extremal rays



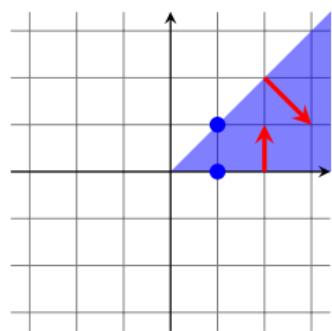
Duality of Cones

Primal

$$C \rightarrow C^* = \{ \mathbf{y} : \forall \mathbf{x} \in C : \langle \mathbf{y}, \mathbf{x} \rangle \geq 0 \}$$

valid constraints → rays

facet constraints → extremal rays



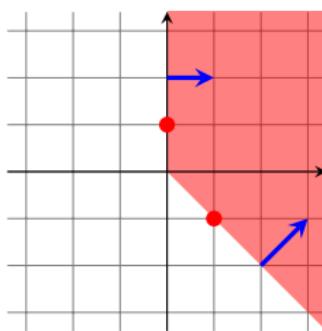
rays ← valid constraints

extremal rays ← facet constraints

Dual

rays

extremal rays



valid constraints ← rays

facet constraints ← extremal rays

duality

Vertex Enumeration by Elimination of Farkas Multipliers

Input polyhedron: $P = \{ \mathbf{x} : A\mathbf{x} + \mathbf{b} \geq 0 \}$

Application of Farkas expresses all valid constraints as

$$\left\{ \mathbf{y} : \exists \lambda \geq 0 : \mathbf{y}^T = \boldsymbol{\lambda}^T \begin{bmatrix} A & \mathbf{b} \\ \mathbf{0}^T & 1 \end{bmatrix} \right\}$$

\Rightarrow cone generated by constraints of $C = \{ \mathbf{x} : A\mathbf{x} + \mathbf{b}z \geq 0 \wedge z \geq 0 \}$

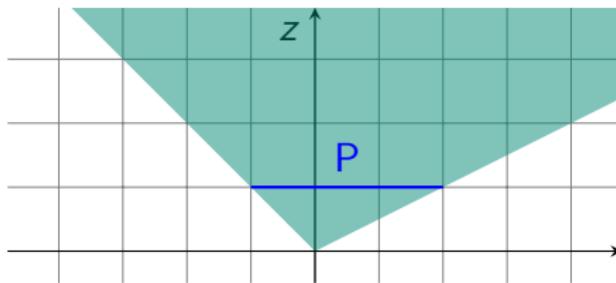
Vertex Enumeration by Elimination of Farkas Multipliers

Input polyhedron: $P = \{ \mathbf{x} : A\mathbf{x} + \mathbf{b} \geq 0 \}$

Application of Farkas expresses all valid constraints as

$$\left\{ \mathbf{y} : \exists \lambda \geq 0 : \mathbf{y}^T = \boldsymbol{\lambda}^T \begin{bmatrix} A & \mathbf{b} \\ \mathbf{0}^T & 1 \end{bmatrix} \right\}$$

- ⇒ cone generated by constraints of $C = \{ \mathbf{x} : A\mathbf{x} + \mathbf{b}z \geq 0 \wedge z \geq 0 \}$
- ⇒ elimination of Farkas multipliers yields dual cone C^*
- ⇒ facet constraints are extremal rays of C
 - ▶ rays with non-zero z -coordinate
 - ⇒ vertex of P (z is denominator)
 - ▶ rays with zero z -coordinate
 - ⇒ ray of P



References I

- [1] D. Avis.
lrs: A revised implementation of the reverse search vertex enumeration algorithm.
In G. Kalai and G. Ziegler, editors, *Polytopes — Combinatorics and Computation*, pages 177–198. Birkhäuser-Verlag, 2000.
DMV Seminar Band 29.
- [2] D. Avis, D. Bremner, and R. Seidel.
How good are convex hull algorithms?
Computational Geometry: Theory and Applications, 7:265–301, 1997.
- [3] R. Baghadi, A. Cohen, S. Verdoolaege, and K. Trifunovic.
Improved loop tiling based on the removal of spurious false dependences.
TACO, 9(4):52, 2013.

References II

- [4] R. Bagnara, P. M. Hill, and E. Zaffanella.
The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems.
Science of Computer Programming, 72(1–2):3–21, 2008.
- [5] V. Baldoni, N. Berline, and M. Vergne.
Summing a polynomial function over integral points of a polygon.
user's guide, May 2009.
- [6] A. Barvinok and J. Pommersheim.
An algorithmic theory of lattice points in polyhedra.
New Perspectives in Algebraic Combinatorics, 38:91–147, 1999.
- [7] A. Barvinok and K. Woods.
Short rational generating functions for lattice point problems.
J. Amer. Math. Soc., 16:957–979, Apr. 2003.

References III

- [8] C. Bastoul.
Code generation in the polyhedral model is easier than you think.
In *PACT '04: Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 7–16, Washington, DC, USA, 2004. IEEE Computer Society.
- [9] N. Berline and M. Vergne.
Local euler-maclaurin formula for polytopes, July 2006.
<http://arXiv.org/abs/math/0507256>.
- [10] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan.
A practical automatic polyhedral parallelizer and locality optimizer.
SIGPLAN Not., 43(6):101–113, 2008.
- [11] C. Chen.
Polyhedra scanning revisited.
SIGPLAN Not., 47(6):499–508, June 2012.

References IV

- [12] P. Clauss.
Counting solutions to linear and nonlinear constraints through Ehrhart polynomials: Applications to analyze and transform scientific programs.
In *International Conference on Supercomputing*, pages 278–285, 1996.
- [13] P. Clauss, F. J. Fernandez, D. Garberetsky, and S. Verdoollaeghe.
Symbolic polynomial maximization over convex sets and its application to memory requirement estimation.
IEEE Transactions on VLSI Systems, 17(8):983–996, Aug. 2009.

References V

- [14] W. Cook, T. Rutherford, H. E. Scarf, and D. F. Shallcross.
An implementation of the generalized basis reduction algorithm for
integer programming.
Cowles Foundation Discussion Papers 990, Cowles Foundation, Yale
University, Aug. 1991.
available at <http://ideas.repec.org/p/cwl/cwldpp/990.html>.
- [15] J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida.
Effective lattice point counting in rational convex polytopes.
The Journal of Symbolic Computation, 38(4):1273–1302, 2004.
- [16] D. Detlefs, G. Nelson, and J. B. Saxe.
Simplify: a theorem prover for program checking.
J. ACM, 52(3):365–473, 2005.

References VI

- [17] P. Feautrier.
Parametric integer programming.
RAIRO Recherche Opérationnelle, 22(3):243–268, 1988.
- [18] P. Feautrier.
Dataflow analysis of array and scalar references.
International Journal of Parallel Programming, 20(1):23–53, 1991.
- [19] P. Feautrier, J. Collard, and C. Bastoul.
PIP/PipLib: A solver for parametric integer programming problems, 2007.
- [20] K. Fukuda, T. M. Liebling, and C. Lütolf.
Extended convex hull.
In *Proceedings of the 12th Canadian Conference on Computational Geometry*, pages 57–63, 2000.

References VII

- [21] T. Grosser, A. Groesslinger, and C. Lengauer.
Polly - performing polyhedral optimizations on a low-level
intermediate representation.
Parallel Processing Letters, 22(04), 2012.
- [22] W. Kelly, W. Pugh, E. Rosser, and T. Shpeisman.
Transitive closure of infinite graphs and its applications.
Int. J. Parallel Program., 24(6):579–598, 1996.
- [23] M. Köppe.
A primal Barvinok algorithm based on irrational decompositions.
SIAM Journal on Discrete Mathematics, 21(1):220–236, 2007.
- [24] V. Loechner and D. K. Wilde.
Parameterized polyhedra and their vertices.
International Journal of Parallel Programming, 25(6):525–549, Dec.
1997.

References VIII

- [25] V. Maslov.
Lazy array data-flow dependence analysis.
In H.-J. Boehm, B. Lang, and D. M. Yellin, editors, *POPL*, pages 311–325. ACM Press, 1994.
- [26] W. Pugh.
Counting solutions to Presburger formulas: How and why.
In *SIGPLAN Conference on Programming Language Design and Implementation (PLDI'94)*, pages 121–134, 1994.
- [27] J. Rambau.
TOPCOM: Triangulations of point configurations and oriented matroids.
In A. M. Cohen, X.-S. Gao, and N. Takayama, editors, *Mathematical Software—ICMS 2002*, pages 330–340, 2002.

References IX

- [28] N. Tawbi.
Estimation of nested loops execution time by integer arithmetic in convex polyhedra.
In *Proceedings of the 8th International Parallel Processing Symposium*, pages 217–221. IEEE Computer Society Press, 1994.
- [29] S. Verdoolaege.
barvinok: User guide, 2008.
- [30] S. Verdoolaege.
isl: An integer set library for the polyhedral model.
In K. Fukuda, J. Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 299–302. Springer, 2010.

References X

- [31] S. Verdoolaege.
Counting affine calculator and applications.
In *First International Workshop on Polyhedral Compilation Techniques (IMPACT'11)*, Chamonix, France, Apr. 2011.
- [32] S. Verdoolaege and M. Bruynooghe.
Algorithms for weighted counting over parametric polytopes: A survey and a practical comparison.
In M. Beck and T. Stoll, editors, *The 2008 International Conference on Information Theory and Statistical Learning*, July 2008.
- [33] S. Verdoolaege, J. Carlos Juega, A. Cohen, J. Ignacio Gómez, C. Tenllado, and F. Catthoor.
Polyhedral parallel code generation for CUDA.
ACM TACO, 9(4):54, 2013.

References XI

- [34] S. Verdoolaege and T. Grosser.
Polyhedral extraction tool.
In *Second International Workshop on Polyhedral Compilation Techniques (IMPACT'12)*, Paris, France, Jan. 2012.
- [35] S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner, and M. Bruynooghe.
Counting integer points in parametric polytopes using Barvinok's rational functions.
Algorithmica, 48(1):37–66, June 2007.
- [36] S. Verdoolaege, K. Woods, M. Bruynooghe, and R. Cools.
Computation and manipulation of enumerators of integer projections of parametric polytopes.
Report CW 392, Dept. of Computer Science, K.U.Leuven, Leuven, Belgium, 2005.

References XII

- [37] D. K. Wilde.
A library for doing polyhedral operations.
Technical Report 785, IRISA, Rennes, France, 1993.
- [38] D. Wonnacott.
Constraint-Based Array Dependence Analysis.
PhD thesis, The University of Maryland, 1995.