An Integer Set Library for Program Analysis

Sven Verdoolaege

April 26, 2009

Program Analysis and Transformation

Most expensive part of a multimedia or signal processing program: manipulation of *arrays* inside *loops*
$\Rightarrow$ most interesting part to optimize

Program Analysis and Transformation

Most expensive part of a multimedia or signal processing program: manipulation of *arrays* inside *loops*
$\Rightarrow$ most interesting part to optimize

$\Rightarrow$ need for *compact representation* of
iterations of a loop/elements of an array
$\Rightarrow$ + *efficient to manipulate*

Program Analysis and Transformation

Most expensive part of a multimedia or signal processing program: manipulation of *arrays* inside *loops*
$\Rightarrow$ most interesting part to optimize

$\Rightarrow$ need for *compact representation* of
iterations of a loop/elements of an array
$\Rightarrow$ + *efficient to manipulate*

$\Rightarrow$ Integer points in polyhedra ("polyhedral model")
$\Rightarrow$ More generally: sets of integers bounded by affine inequalities

Representation Example: Iteration Domain

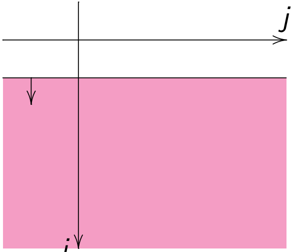
```
#define N 5
for (i = 1; i <= N; ++i)
    for (j = 1; j <= i; ++j)
        a[i][j]=
```

Assumptions on sequential code:

- iterators are integers
- loops with affine bounds
- affine conditions
- affine index expressions

Representation Example: Iteration Domain

```
#define N 5
for (i = 1; i <= N; ++i)
    for (j = 1; j <= i; ++j)
        a[i][j]=
```



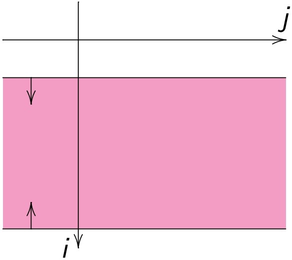
Iteration domain: $P = \{\, [i,j] \mid i \geq 1 \quad\quad\quad\quad\quad\quad \}$

Assumptions on sequential code:

- iterators are integers
- loops with affine bounds
- affine conditions
- affine index expressions

Representation Example: Iteration Domain

```
#define N 5
for (i = 1; i <= N; ++i)
    for (j = 1; j <= i; ++j)
        a[i][j]=
```



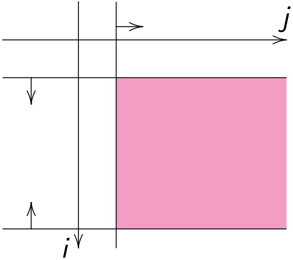
Iteration domain: $P = \{\, [i,j] \mid i \geq 1 \wedge i \leq N \qquad\qquad \}$

Assumptions on sequential code:

- iterators are integers
- loops with affine bounds
- affine conditions
- affine index expressions

Representation Example: Iteration Domain

```
#define N 5
for (i = 1; i <= N; ++i)
    for (j = 1; j <= i; ++j)
        a[i][j]=
```



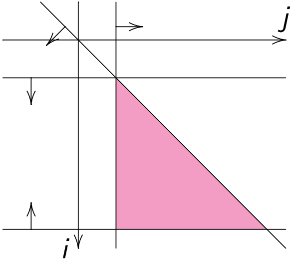
Iteration domain: $P = \{ [i, j] \mid i \geq 1 \wedge i \leq N \wedge j \geq 1 \qquad \}$

Assumptions on sequential code:

- iterators are integers
- loops with affine bounds
- affine conditions
- affine index expressions

Representation Example: Iteration Domain

```
#define N 5
for (i = 1; i <= N; ++i)
    for (j = 1; j <= i; ++j)
        a[i][j]=
```



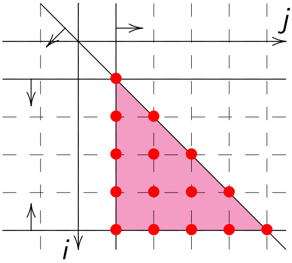
Iteration domain: $P = \{ [i, j] \mid i \geq 1 \wedge i \leq N \wedge j \geq 1 \wedge j \leq i \}$

Assumptions on sequential code:

- iterators are integers
- loops with affine bounds
- affine conditions
- affine index expressions

Representation Example: Iteration Domain

```
#define N 5
for (i = 1; i <= N; ++i)
    for (j = 1; j <= i; ++j)
        a[i][j]=
```



Iteration domain: $P = \{\, [i,j] \mid i \geq 1 \wedge i \leq N \wedge j \geq 1 \wedge j \leq i \,\}$

Assumptions on sequential code:

- iterators are integers
- loops with affine bounds
- affine conditions
- affine index expressions

Program Transformation Example

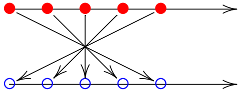
```
for (i = 0; i <= N; ++i)
    a[i] = ...
for (i = 0; i <= N; ++i)
    b[i] = f(a[N-i])
```

Execution order: top-down, left-right

- ●, ○ Statement iteration
- ⟶ Data flow dependence
- ● Executed statement
- ⟶ Data in memory

Program Transformation Example

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

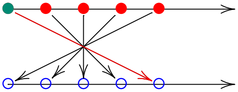


Execution order: top-down, left-right

- ●, ○ Statement iteration
- → Data flow dependence
- ● Executed statement
- → Data in memory

Program Transformation Example

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

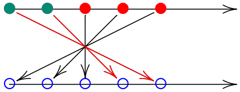


Execution order: top-down, left-right

●, ○	Statement iteration
→	Data flow dependence
●	Executed statement
→	Data in memory

Program Transformation Example

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

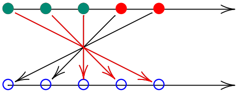


Execution order: top-down, left-right

- ●, ○ Statement iteration
- → Data flow dependence
- ● Executed statement
- → Data in memory

Program Transformation Example

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

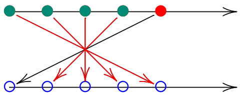


Execution order: top-down, left-right

- ●, ○ Statement iteration
- → Data flow dependence
- ● Executed statement
- → Data in memory

Program Transformation Example

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

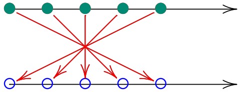


Execution order: top-down, left-right

- ●, ○ Statement iteration
- ⟶ Data flow dependence
- ● Executed statement
- ⟶ Data in memory

Program Transformation Example

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

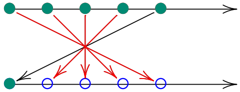


Execution order: top-down, left-right

- •, ○ Statement iteration
- → Data flow dependence
- • Executed statement
- → Data in memory

Program Transformation Example

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

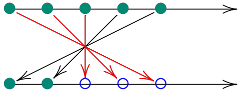


Execution order: top-down, left-right

- ●, ○ Statement iteration
- → Data flow dependence
- ● Executed statement
- → Data in memory

Program Transformation Example

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

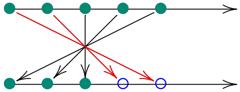


Execution order: top-down, left-right

- ●, ○ Statement iteration
- → Data flow dependence
- ● Executed statement
- → Data in memory

Program Transformation Example

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

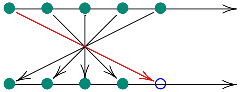


Execution order: top-down, left-right

- ●, ○ Statement iteration
- ⟶ Data flow dependence
- ● Executed statement
- ⟶ Data in memory

Program Transformation Example

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

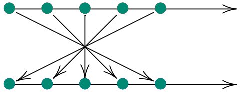


Execution order: top-down, left-right

●, ○	Statement iteration
→	Data flow dependence
●	Executed statement
→	Data in memory

Program Transformation Example

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

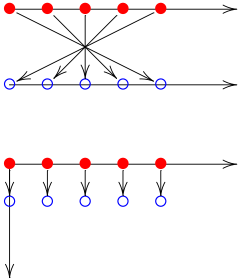


Execution order: top-down, left-right

- ●, ○ Statement iteration
- → Data flow dependence
- ● Executed statement
- → Data in memory

Program Transformation Example

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

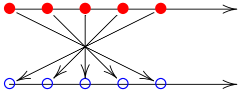


Execution order: top-down, left-right

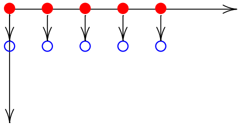
- ●, ○ Statement iteration
- → Data flow dependence
- ● Executed statement
- → Data in memory

Program Transformation Example

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

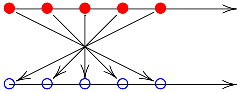


```
for (i = 0; i <= N; ++i)
    a[i] = ...
for (i = 0; i <= N; ++i)
    b[N-i] = f(a[i])
```

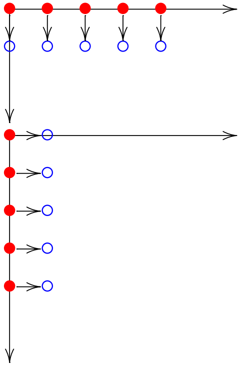


Program Transformation Example

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



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

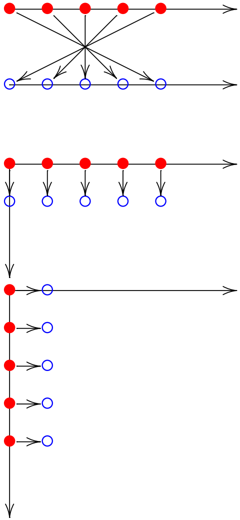


Program Transformation Example

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

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

```
for (i = 0; i <= N; ++i) {
    a[i] = ...
    b[N-i] = f(a[i])
}
```



Two Program Analysis and Transformation Tools

Why do we need an integer set library?

- Equivalence checker
  - Checks the equivalence of two programs represented in the polyhedral model
  - Proves output is the same given that input is the same
  - Maintains maps between statement iterations of both programs that should be proven to produce the same result

  Requirements
  - manipulations on *integer* sets/maps
  - explicit support for existentially quantified variables

Equivalence Checking Example

Given `in1 == in2`, can we prove `out1 == out2`?

```
a1[0] = in1;
for (i = 1; i <= N; ++i)
  a1[i] = f(a1[i - 1]);
out1 = a1[N];
```

```
a2[0] = in2;
for (i = 1; i <= N; ++i)
  a2[i] = f(a2[i - 1]);
out2 = a2[N];
```

Equivalence Checking Example

Given `in1 == in2`, can we prove `out1 == out2`?

```
a1[0] = in1;
for (i = 1; i <= N; ++i)
  a1[i] = f(a1[i - 1]);
out1 = a1[N];
```

```
a2[0] = in2;
for (i = 1; i <= N; ++i)
  a2[i] = f(a2[i - 1]);
out2 = a2[N];
```

`out1 == out2` requires `a1[N] == a2[N]`

Equivalence Checking Example

Given `in1 == in2`, can we prove `out1 == out2`?

```
a1[0] = in1;                    a2[0] = in2;
for (i = 1; i <= N; ++i)        for (i = 1; i <= N; ++i)
  a1[i] = f(a1[i - 1]);           a2[i] = f(a2[i - 1]);
out1 = a1[N];                   out2 = a2[N];
```

`out1 == out2` requires `a1[N] == a2[N]`

`a1[N] == a2[N]` requires `a1[N-1] == a2[N-1]`

Equivalence Checking Example

Given `in1 == in2`, can we prove `out1 == out2`?

```
a1[0] = in1;
for (i = 1; i <= N; ++i)
  a1[i] = f(a1[i - 1]);
out1 = a1[N];
```

```
a2[0] = in2;
for (i = 1; i <= N; ++i)
  a2[i] = f(a2[i - 1]);
out2 = a2[N];
```

`out1 == out2` requires `a1[N] == a2[N]`
`a1[N] == a2[N]` requires `a1[N-1] == a2[N-1]`
`a1[N-1] == a2[N-1]` requires `a1[N-2] == a2[N-2]`

Equivalence Checking Example

Given `in1 == in2`, can we prove `out1 == out2`?

```
a1[0] = in1;
for (i = 1; i <= N; ++i)
  a1[i] = f(a1[i - 1]);
out1 = a1[N];
```

```
a2[0] = in2;
for (i = 1; i <= N; ++i)
  a2[i] = f(a2[i - 1]);
out2 = a2[N];
```

`out1 == out2` requires `a1[N] == a2[N]`
`a1[N] == a2[N]` requires `a1[N-1] == a2[N-1]`
`a1[N-1] == a2[N-1]` requires `a1[N-2] == a2[N-2]`
...

Equivalence Checking Example

Given `in1 == in2`, can we prove `out1 == out2`?

```
a1[0] = in1;                    a2[0] = in2;
for (i = 1; i <= N; ++i)        for (i = 1; i <= N; ++i)
  a1[i] = f(a1[i - 1]);           a2[i] = f(a2[i - 1]);
out1 = a1[N];                   out2 = a2[N];
```

`out1 == out2` requires `a1[N] == a2[N]`

`a1[N] == a2[N]` requires `a1[N-1] == a2[N-1]`

`a1[N-1] == a2[N-1]` requires `a1[N-2] == a2[N-2]`

...

$\Rightarrow$ requires `a1[i] == a2[i]` for $1 \leq i \leq N$

$\Rightarrow$ induction for $2 \leq i \leq N$ + requires `a1[0] = a2[0]`

$\Rightarrow$ Integer affine hull of $\{ (N, N), (N-1, N-1) \}$: $\{ (i, i) \}$

Equivalence Checking Example

Given `in1 == in2`, can we prove `out1 == out2`?

```
a1[0] = in1;                    a2[0] = in2;
for (i = 1; i <= N; ++i)        for (i = 1; i <= N; ++i)
  a1[i] = f(a1[i - 1]);           a2[i] = f(a2[i - 1]);
out1 = a1[N];                   out2 = a2[N];
```

`out1 == out2` requires `a1[N] == a2[N]`
`a1[N] == a2[N]` requires `a1[N-1] == a2[N-1]`
`a1[N-1] == a2[N-1]` requires `a1[N-2] == a2[N-2]`
...
$\Rightarrow$ requires `a1[i] == a2[i]` for $1 \le i \le N$
$\Rightarrow$ induction for $2 \le i \le N$ + requires `a1[0] = a2[0]`
$\Rightarrow$ Integer affine hull of $\{ (N, N), (N-1, N-1) \}$: $\{ (i, i) \}$

`a1[0] == a2[0]` requires `in1 == in2`

Two Program Analysis and Transformation Tools

Why do we need an integer set library?

- Equivalence checker
  - Checks the equivalence of two programs represented in the polyhedral model
  - Proves output is the same given that input is the same
  - Maintains maps between statement iterations of both programs that should be proven to produce the same result

  Requirements
  - manipulations on *integer* sets/maps
  - explicit support for existentially quantified variables

Two Program Analysis and Transformation Tools

Why do we need an integer set library?

- Equivalence checker
  - Checks the equivalence of two programs represented in the polyhedral model
  - Proves output is the same given that input is the same
  - Maintains maps between statement iterations of both programs that should be proven to produce the same result

  Requirements

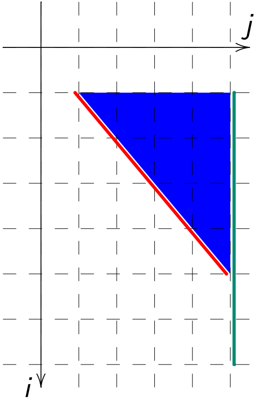
  - manipulations on *integer* sets/maps
  - explicit support for existentially quantified variables
- CLooG
  - Generates code for scanning integer points in polyhedra (iteration domains)

  Requirements

  - manipulations on *integer* sets
    $\Rightarrow$ remove redundant constraints/code
  - explicit support for existentially quantified variables
    $\Rightarrow$ replace some loops by guards

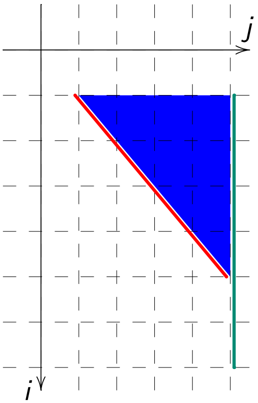
CLooG Example

S1: $\{(i, j) \mid 1 \leq i \leq n \leq m \wedge j = i\}$
S2: $\{(i, j) \mid 1 \leq i \leq n \leq m \wedge i \leq j \leq n\}$
S3: $\{(i, j) \mid 1 \leq i \leq m \wedge j = n \leq m\}$



CLooG Example

S1: $\{(i,j) \mid 1 \leq i \leq n \leq m \wedge j = i\}$
S2: $\{(i,j) \mid 1 \leq i \leq n \leq m \wedge i \leq j \leq n\}$
S3: $\{(i,j) \mid 1 \leq i \leq m \wedge j = n \leq m\}$




```
for (i=1;i<=m;i++) {
  if (i <= n) {
    S1(i,i);
  }
  for (j=i;j<=n;j++) {
    S2(i);
  }
  S3(i,n);
}
```

Required Operations

- Basic operations
  - Union
  - Intersection
  - Set difference
  - ...
- Operations required by equivalence checking
  - Integer affine hull
  - ...
- Operations required by CLooG
  - Projection (rational)
  - Ordering
  - Convex hull (rational)
  - Simplification
  - ...

Why not use a double description based library?

E.g., PolyLib, PPL, (polymake)

- Who needs vertices anyway?
  - Very useful for `LattE macchiato`/`barvinok` style counting

    (but neither equivalence checking or CLooG needs any counting)
  - Some operations can be performed more efficiently on explicit representation

    But:
    - Computing the dual can be costly
    - Double description requires more space

    ⇒ trade-off

    (sets used in equivalence checking and CLooG usually have few constraints)

Why not use a double description based library?

E.g., PolyLib, PPL, (polymake)

- Who needs vertices anyway?
  - Very useful for `LattE macchiato`/`barvinok` style counting

    (but neither equivalence checking or CLooG needs any counting)
  - Some operations can be performed more efficiently on explicit representation

    But:
    - Computing the dual can be costly
    - Double description requires more space

    $\Rightarrow$ trade-off

    (sets used in equivalence checking and CLooG usually have few constraints)
- Usually focus on *rational* values
- Little/no support for existentially quantified variables

Why do we need existentially quantified variables?

- Modeling some problems
  Which array elements are accessed in this loop?

```
for (j = 1; j <= p; ++j)
    for (i = 1; i <= 8; ++i)
        a[6i+9j-7] = a[6i+9j-7] + 5;
```

$S(s) = \{l \in \mathbb{Z} \mid \exists i, j \in \mathbb{Z} : l = 6i + 9j - 7 \wedge 1 \leq j \leq s \wedge 1 \leq i \leq 8\}$

Why do we need existentially quantified variables?

- Modeling some problems
  Which array elements are accessed in this loop?

  ```
  for (j = 1; j <= p; ++j)
      for (i = 1; i <= 8; ++i)
          a[6i+9j-7] = a[6i+9j-7] + 5;
  ```

  $S(s) = \{ l \in \mathbb{Z} \mid \exists i, j \in \mathbb{Z} : l = 6i + 9j - 7 \wedge 1 \leq j \leq s \wedge 1 \leq i \leq 8 \}$

- Especially integer divisions/remainders
  E.g., `i % 10 <= 6`

$$i - 10 \left\lfloor \frac{i}{10} \right\rfloor \leq 6$$

Why do we need existentially quantified variables?

- Modeling some problems
  Which array elements are accessed in this loop?

```
for (j = 1; j <= p; ++j)
    for (i = 1; i <= 8; ++i)
        a[6i+9j-7] = a[6i+9j-7] + 5;
```

$S(s) = \{l \in \mathbb{Z} \mid \exists i, j \in \mathbb{Z} : l = 6i + 9j - 7 \wedge 1 \leq j \leq s \wedge 1 \leq i \leq 8\}$

- Especially integer divisions/remainders
  E.g., `i % 10 <= 6`

$$i - 10\left\lfloor \frac{i}{10} \right\rfloor \leq 6$$

$$i - 10\alpha \leq 6$$

with $i - 9 \leq 10\alpha \leq i$

Why do we need existentially quantified variables?

- Modeling some problems
  Which array elements are accessed in this loop?

```
for (j = 1; j <= p; ++j)
    for (i = 1; i <= 8; ++i)
        a[6i+9j-7] = a[6i+9j-7] + 5;
```

$$S(s) = \{ l \in \mathbb{Z} \mid \exists i, j \in \mathbb{Z} : l = 6i + 9j - 7 \wedge 1 \le j \le s \wedge 1 \le i \le 8 \}$$

- Especially integer divisions/remainders
  E.g., `i % 10 <= 6`

$$i - 10 \left\lfloor \frac{i}{10} \right\rfloor \le 6$$

$$i - 10\alpha \le 6$$

  with $i - 9 \le 10\alpha \le i$

  - May appear in original code
  - May be introduced by (PIP-based) dependence analysis

Why not use the Omega library?

- focuses on *integer* values
- has explicit support for existentially quantified variables
- very fast on small problems due to extensive use of heuristics

Why not use the Omega library?

- focuses on *integer* values
- has explicit support for existentially quantified variables
- very fast on small problems due to extensive use of heuristics

But:

- not supported for many years (until recently)
- accuracy limited by machine precision
- different way of handling existentially quantified variables
- some heuristics favor speed over accuracy

Why not use the Omega library?

- focuses on *integer* values
- has explicit support for existentially quantified variables
- very fast on small problems due to extensive use of heuristics

But:

- not supported for many years (until recently)
- accuracy limited by machine precision
- different way of handling existentially quantified variables
- some heuristics favor speed over accuracy

```
# Omega Calculator v2.1 (based on Omega Library 2.1, Ju
AffineHull {[a,b] : a=b && 1 <= a <= 162};
# AffineHull {[a,b] : a=b && 1 <= a <= 162};
{[a,a]}
AffineHull {[a,b] : a=b && 1 <= a <= 163};
# AffineHull {[a,b] : a=b && 1 <= a <= 163};
{[In_1,In_2]}
```

$\Rightarrow$ completely unacceptable for equivalence checking

Internal Representation

$$S(\mathbf{s}) = \{\, \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : A\mathbf{x} + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \,\}$$

$$R(\mathbf{s}) = \{\, (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2} \mid \exists \mathbf{z} \in \mathbb{Z}^e : A_1\mathbf{x}_1 + A_2\mathbf{x}_2 + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \,\}$$

- "basic" types: "convex" sets and maps (relations)
  - equality + inequality constraints
  - parameters $\mathbf{s}$
  - (optional) explicit representation of existentially quantified variables as integer divisions
    - ⇒ useful for aligning dimensions when performing set operations (e.g., set difference)
    - ⇒ can be computed using PIP
    - ⇒ already available if obtained from PIP-based dependence analysis
- union types: sets and maps
  - ⇒ (disjoint) unions of basic sets/maps

Parametric Integer Programming

$$R(\mathbf{s}) = \{\, (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2} \mid \exists \mathbf{z} \in \mathbb{Z}^e : A_1\mathbf{x}_1 + A_2\mathbf{x}_2 + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \,\}$$

Lexicographic minimum of R:

$$\operatorname{lexmin} R = \{\, (\mathbf{x}_1, \mathbf{x}_2) \in R \mid \forall \mathbf{x}_2' \in R(\mathbf{s}, \mathbf{x}_1) : \mathbf{x}_2 \preccurlyeq \mathbf{x}_2' \,\}$$

Parametric Integer Programming

$$R(\mathbf{s}) = \{\, (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2} \mid \exists \mathbf{z} \in \mathbb{Z}^e : A_1\mathbf{x}_1 + A_2\mathbf{x}_2 + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \,\}$$

Lexicographic minimum of R:

$$\operatorname{lexmin} R = \{\, (\mathbf{x}_1, \mathbf{x}_2) \in R \mid \forall \mathbf{x}_2' \in R(\mathbf{s}, \mathbf{x}_1) : \mathbf{x}_2 \preccurlyeq \mathbf{x}_2' \,\}$$

Parametric integer programming computes $\operatorname{lexmin} R$ in the form

$$\begin{aligned} \operatorname{lexmin} R = \bigcup_i \{\, (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2} \mid \exists \mathbf{z}' \in \mathbb{Z}^{e'} : A_i\mathbf{x}_1 + B_i\mathbf{s} \geq \mathbf{c}_i \wedge \\ \mathbf{z}' = \left\lfloor \frac{P_i\mathbf{x}_1 + Q_i\mathbf{s} + \mathbf{r}_i}{m} \right\rfloor \wedge \\ \mathbf{x}_2 = T_i\mathbf{x}_1 + U_i\mathbf{s} + V_i\mathbf{z}' + \mathbf{w}_i \,\} \end{aligned}$$

- explicit representation of existentially quantified variables
- explicit representation of range variables

Technique: dual simplex + Gomory cuts

Parametric Integer Programming

$$R(\mathbf{s}) = \{\, (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2} \mid \exists \mathbf{z} \in \mathbb{Z}^e : A_1\mathbf{x}_1 + A_2\mathbf{x}_2 + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \,\}$$

Lexicographic minimum of R:

$$\operatorname{lexmin} R = \{\, (\mathbf{x}_1, \mathbf{x}_2) \in R \mid \forall \mathbf{x}_2' \in R(\mathbf{s}, \mathbf{x}_1) : \mathbf{x}_2 \preccurlyeq \mathbf{x}_2' \,\}$$

Parametric integer programming computes $\operatorname{lexmin} R$ in the form

$$\operatorname{lexmin} R = \bigcup_i \{\, (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2} \mid \exists \mathbf{z}' \in \mathbb{Z}^{e'} : A_i\mathbf{x}_1 + B_i\mathbf{s} \geq \mathbf{c}_i \wedge \mathbf{z}' = \left\lfloor \frac{P_i\mathbf{x}_1 + Q_i\mathbf{s} + \mathbf{r}_i}{m} \right\rfloor \wedge \mathbf{x}_2 = T_i\mathbf{x}_1 + U_i\mathbf{s} + V_i\mathbf{z}' + \mathbf{w}_i \,\}$$

- explicit representation of existentially quantified variables
- explicit representation of range variables

Technique: dual simplex + Gomory cuts

Parametric Integer Programming

$$R(\mathbf{s}) = \{\, (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2} \mid \exists \mathbf{z} \in \mathbb{Z}^e : A_1\mathbf{x}_1 + A_2\mathbf{x}_2 + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \,\}$$

Lexicographic minimum of R:

$$\operatorname{lexmin} R = \{\, (\mathbf{x}_1, \mathbf{x}_2) \in R \mid \forall \mathbf{x}_2' \in R(\mathbf{s}, \mathbf{x}_1) : \mathbf{x}_2 \preccurlyeq \mathbf{x}_2' \,\}$$

Parametric integer programming computes $\operatorname{lexmin} R$ in the form

$$\operatorname{lexmin} R = \bigcup_i \{\, (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2} \mid \exists \mathbf{z}' \in \mathbb{Z}^{e'} : A_i\mathbf{x}_1 + B_i\mathbf{s} \geq \mathbf{c}_i \wedge \mathbf{z}' = \left\lfloor \frac{P_i\mathbf{x}_1 + Q_i\mathbf{s} + \mathbf{r}_i}{m} \right\rfloor \wedge \mathbf{x}_2 = T_i\mathbf{x}_1 + U_i\mathbf{s} + V_i\mathbf{z}' + \mathbf{w}_i \,\}$$

- explicit representation of existentially quantified variables
- explicit representation of range variables

Technique: dual simplex + Gomory cuts

isl Operation: Set Difference

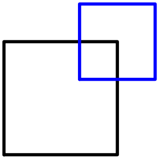
$$S(\mathbf{s}) = \{\, \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : A\mathbf{x} + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \,\}$$

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

$$S(\mathbf{s}) = \{\, \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : A\mathbf{x} + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \,\}$$

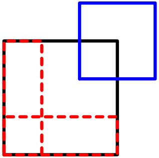
Set difference $S_1 \setminus S_2$

- 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

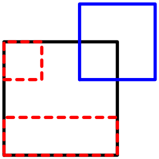
$$S(\mathbf{s}) = \{\, \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{z} \in \mathbb{Z}^e : A\mathbf{x} + B\mathbf{s} + D\mathbf{z} \geq \mathbf{c} \,\}$$

Set difference $S_1 \setminus S_2$

- no existentially quantified variables

$$S_2(\mathbf{s}) = \{\, \mathbf{x} \in \mathbb{Z}^d \mid \bigwedge_i \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i \,\}$$

$$\begin{aligned} S_1 \setminus S_2 &= \bigcup_i (S_1 \cap \{\, \mathbf{x} \mid \neg(\langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i) \,\}) \\ &= \bigcup_i (S_1 \cap \{\, \mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \leq c_i - 1 \,\}) \\ &= \bigcup_i (S_1 \cap \bigcap_{j<i} \{\, \mathbf{x} \mid \langle \mathbf{a}_j, \mathbf{x} \rangle + \langle \mathbf{b}_j, \mathbf{s} \rangle \geq c_j \,\} \\ &\qquad \cap \{\, \mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \leq c_i - 1 \,\}) \end{aligned}$$



`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

$$S_2(\mathbf{s}) = \{\, \mathbf{x} \in \mathbb{Z}^d \mid \bigwedge_i \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i \,\}$$

$$S_1 \setminus S_2 = \bigcup_i \Big(S_1 \cap \bigcap_{j<i} \{\, \mathbf{x} \mid \langle \mathbf{a}_j, \mathbf{x} \rangle + \langle \mathbf{b}_j, \mathbf{s} \rangle \geq c_j \,\} \cap \{\, \mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \leq c_i - 1 \,\}\Big)$$

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

$$S_2(\mathbf{s}) = \{\, \mathbf{x} \in \mathbb{Z}^d \mid \bigwedge_i \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \geq c_i \,\}$$

$$S_1 \setminus S_2 = \bigcup_i (S_1 \cap \bigcap_{j<i} \{\, \mathbf{x} \mid \langle \mathbf{a}_j, \mathbf{x} \rangle + \langle \mathbf{b}_j, \mathbf{s} \rangle \geq c_j \,\} \cap \{\, \mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle + \langle \mathbf{b}_i, \mathbf{s} \rangle \leq c_i - 1 \,\})$$

- with existentially quantified variables
  $\Rightarrow$ 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: 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

$\Rightarrow$ try to combine several basic sets into a single basic set

$$S_1 = \{ \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \} \qquad S_2 = \{ \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \}$$

PolyLib way:

1. Compute $H = \mathrm{conv.hull}(S_1 \cup S_2)$
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
$\Rightarrow$ try to combine several basic sets into a single basic set

$$S_1 = \{ \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \} \qquad S_2 = \{ \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \}$$

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

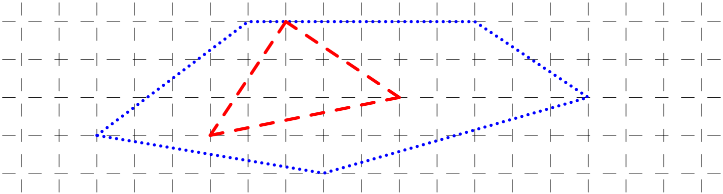
`isl` Operation: Set Coalescing

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

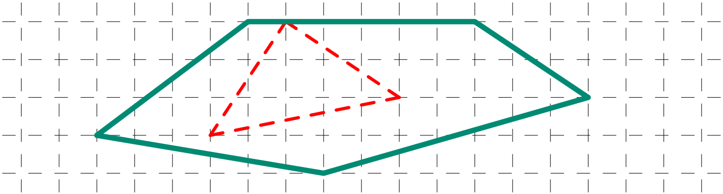
1. Classify constraints
   - redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
   - valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
   - separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
     special cases:
     - 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. Recognize cases
   - 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. Recognize cases
   - 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

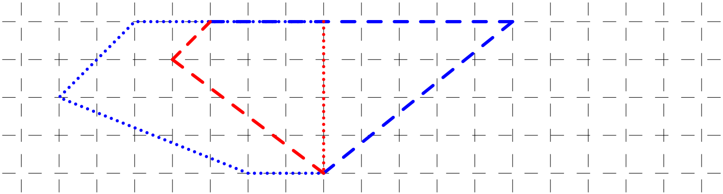


2. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow S_2$ can be dropped

`isl` Operation: Set Coalescing

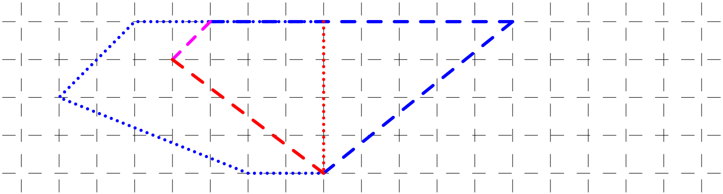
1. Classify constraints
   - redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
   - valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
   - separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
     special cases:
     - 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. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - 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



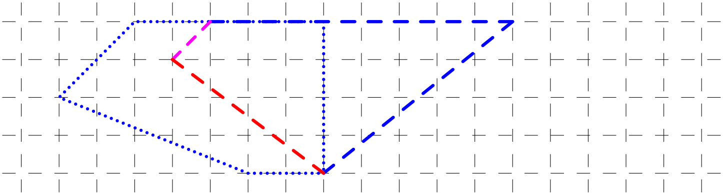
2. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints

isl Operation: Set Coalescing



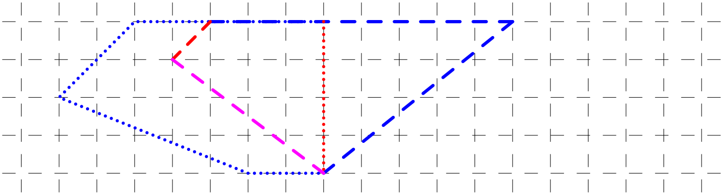
2. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints

isl Operation: Set Coalescing



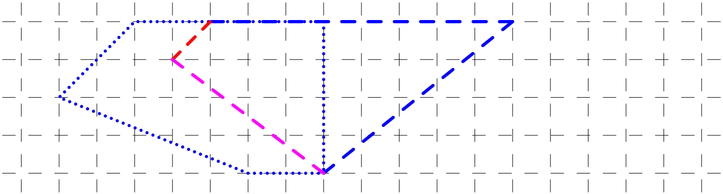
2. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints

isl Operation: Set Coalescing



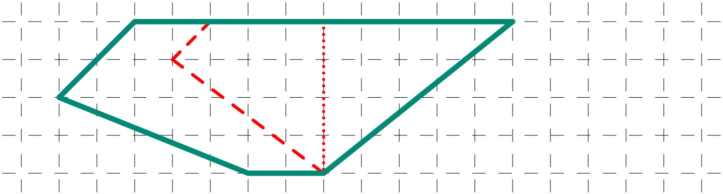
2. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow S_2$ can be dropped
   - no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints

isl Operation: Set Coalescing



2. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - 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

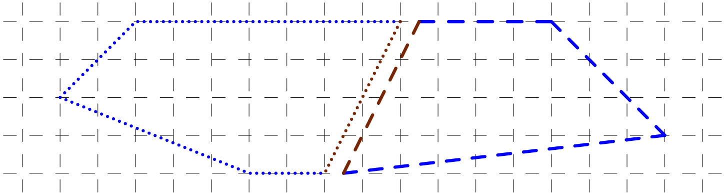


2. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$ $\Rightarrow$ S_2 can be dropped
   - no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001) $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints

`isl` Operation: Set Coalescing

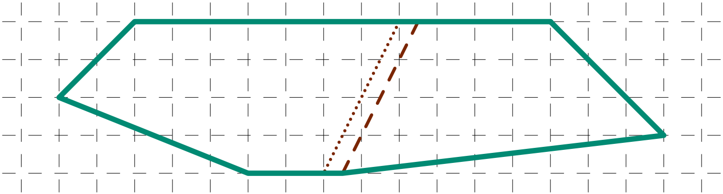
1. Classify constraints
   - redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
   - valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
   - separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
     special cases:
     - 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. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - 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
   - 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. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - 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
   - 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

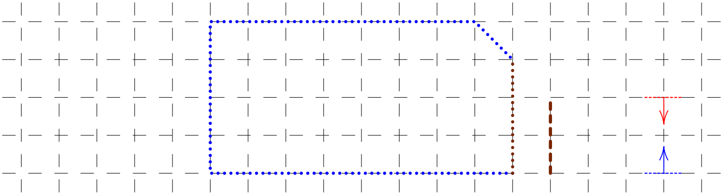


2. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints
   - 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

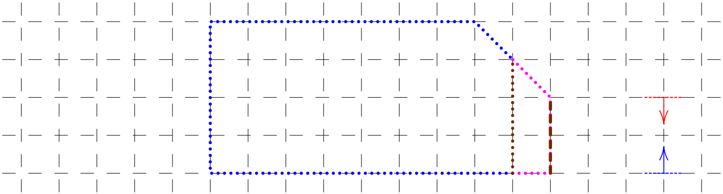
1. Classify constraints
   - redundant: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over remaining constraints
   - valid: $\min \langle \mathbf{a}_i, \mathbf{x} \rangle > c_i - 1$ over S_2
   - separating: $\max \langle \mathbf{a}_i, \mathbf{x} \rangle < c_i$ over S_2
     special cases:
     - 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. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - 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
   - single pair of adjacent inequalities (other constraints valid)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints
   - 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



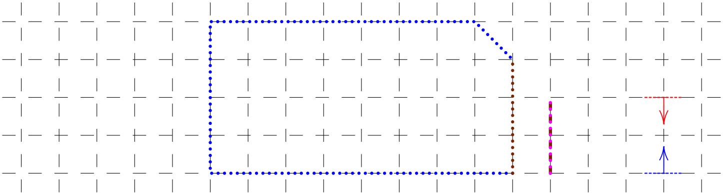
2. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints
   - single pair of adjacent inequalities (other constraints valid)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints
   - 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



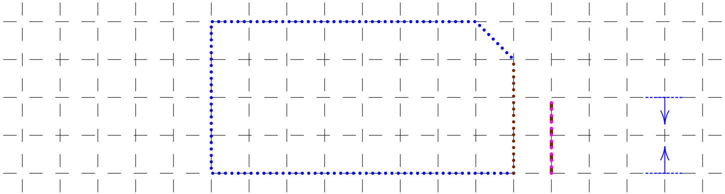
2. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints
   - single pair of adjacent inequalities (other constraints valid)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints
   - 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



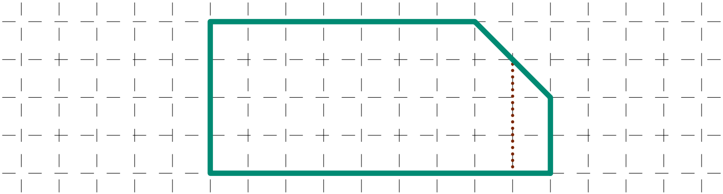
2. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - no separating constraints and cut constraints of S_2 are valid for cut facets of S_1 (similar to BFT2001)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints
   - single pair of adjacent inequalities (other constraints valid)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints
   - 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. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow$ S_2 can be dropped
   - 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
   - single pair of adjacent inequalities (other constraints valid)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints
   - 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. Recognize cases
   - non-redundant constraints of S_1 are valid for S_2, i.e., $S_2 \subseteq S_1$
     $\Rightarrow S_2$ can be dropped
   - 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
   - single pair of adjacent inequalities (other constraints valid)
     $\Rightarrow$ replace S_1 and S_2 by basic set with all valid constraints
   - 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: Closed Convex Hull

$$H = \mathrm{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

1. using elimination
   - convex hull of polyhedra
     $\Rightarrow$ sum of cones in homogeneous space

     $$H = \{\, \mathbf{x} \mid \exists \mathbf{x}_1, \mathbf{x}_2, z_1, z_2 : \mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 \wedge 1 = z_1 + z_2 \wedge A\mathbf{x}_1 \geq \mathbf{c} z_1 \wedge z_1 \geq 0 \wedge B\mathbf{x}_2 \geq \mathbf{d} z_2 \wedge z_2 \geq 0 \,\}$$

   - eliminate $\mathbf{x}_1, \mathbf{x}_2, z_1, z_2$ using Fourier-Motzkin elimination

isl Operation: Closed Convex Hull

$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

1. using elimination
   - convex hull of polyhedra
     $\Rightarrow$ sum of cones in homogeneous space

     $$H = \{\, \mathbf{x} \mid \exists \mathbf{x}_1, \mathbf{x}_2, z_1, z_2 : \mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 \wedge 1 = z_1 + z_2 \wedge A\mathbf{x}_1 \geq \mathbf{c} z_1 \wedge z_1 \geq 0 \wedge B\mathbf{x}_2 \geq \mathbf{d} z_2 \wedge z_2 \geq 0 \,\}$$

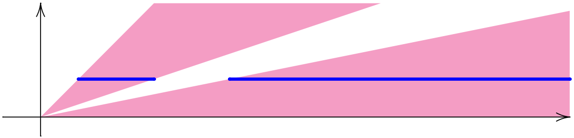
   - eliminate $\mathbf{x}_1, \mathbf{x}_2, z_1, z_2$ using Fourier-Motzkin elimination

isl Operation: Closed Convex Hull

$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

1. using elimination
   - convex hull of polyhedra
     $\Rightarrow$ sum of cones in homogeneous space

     $$H = \{\, \mathbf{x} \mid \exists \mathbf{x}_1, \mathbf{x}_2, z_1, z_2 : \mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 \wedge 1 = z_1 + z_2 \wedge A\mathbf{x}_1 \geq \mathbf{c} z_1 \wedge z_1 \geq 0 \wedge B\mathbf{x}_2 \geq \mathbf{d} z_2 \wedge z_2 \geq 0 \,\}$$
   - eliminate $\mathbf{x}_1, \mathbf{x}_2, z_1, z_2$ using Fourier-Motzkin elimination



isl Operation: Closed Convex Hull

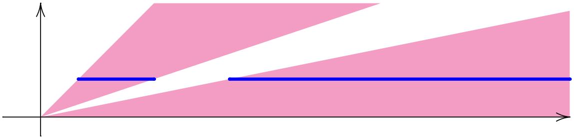
$$H = \mathrm{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

1. using elimination
   - convex hull of polyhedra
     $\Rightarrow$ sum of cones in homogeneous space

     $$H = \{\, \mathbf{x} \mid \exists \mathbf{x}_1, \mathbf{x}_2, z_1, z_2 : \mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 \wedge 1 = z_1 + z_2 \wedge A\mathbf{x}_1 \geq \mathbf{c} z_1 \wedge z_1 \geq 0 \wedge B\mathbf{x}_2 \geq \mathbf{d} z_2 \wedge z_2 \geq 0 \,\}$$

   - eliminate $\mathbf{x}_1, \mathbf{x}_2, z_1, z_2$ using Fourier-Motzkin elimination

   $\Rightarrow$ very inefficient!



isl Operation: Closed Convex Hull

$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

1. using elimination
   - convex hull of polyhedra
     $\Rightarrow$ sum of cones in homogeneous space
     $$H = \{\, \mathbf{x} \mid \exists \mathbf{x}_1, \mathbf{x}_2, z_1, z_2 : \mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 \wedge 1 = z_1 + z_2 \wedge A\mathbf{x}_1 \geq \mathbf{c} z_1 \wedge z_1 \geq 0 \wedge B\mathbf{x}_2 \geq \mathbf{d} z_2 \wedge z_2 \geq 0 \,\}$$
   - eliminate $\mathbf{x}_1, \mathbf{x}_2, z_1, z_2$ using Fourier-Motzkin elimination

   $\Rightarrow$ very inefficient!
2. using "wrapping"
   - S_1 and S_2 are polytopes
     $\Rightarrow$ wrap facets around ridges until all facets found (FLL2000)
   - H is pointed
     $\Rightarrow$ change perspective
   - S_1 and S_2 are pointed (R_i recession cone of S_i)
     $\Rightarrow$ project out $\text{lineality } H = \text{lin.hull}(R_1 \cap -R_2)$
   - S_1 or S_2 has non-trivial lineality space
     $\Rightarrow$ project out $\text{lineality } S_1$ and $\text{lineality } S_2$

isl Operation: Closed Convex Hull—Wrapping

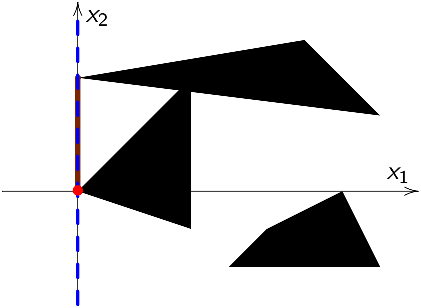
$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \ge \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \ge \mathbf{d} \,\}$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \ge 0$ defines a facet and $x_2 \ge 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \ge a x_1$

isl Operation: Closed Convex Hull—Wrapping

$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$

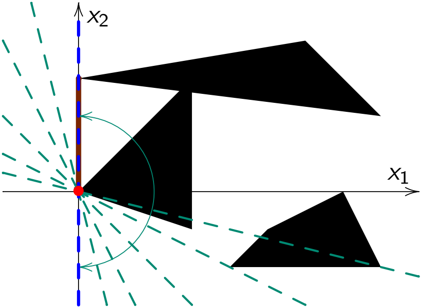
- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$



isl Operation: Closed Convex Hull—Wrapping

$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$

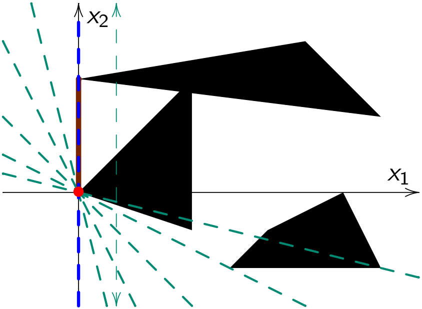
- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$



`isl` Operation: Closed Convex Hull—Wrapping

$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

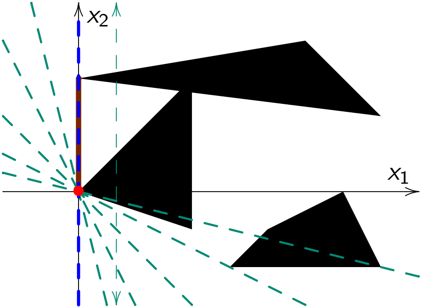
- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$



isl Operation: Closed Convex Hull—Wrapping

$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$



Compute $a = \min x_2 + y_2$ s.t.

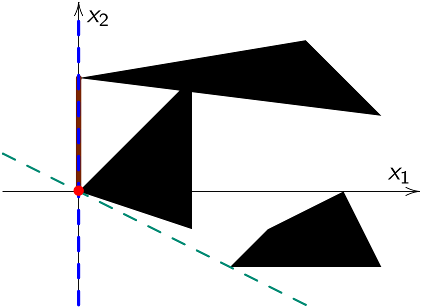
$$x_1 + y_1 = 1 \wedge A\mathbf{x} \geq \mathbf{c}x_0 \wedge x_0 \geq 0 \wedge B\mathbf{y} \geq \mathbf{d}y_0 \wedge y_0 \geq 0$$

(Cone of hull is sum of cones in homogeneous space)

isl Operation: Closed Convex Hull—Wrapping

$$H = \mathrm{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge ⇒ new facet constraint $x_2 \geq a x_1$



Compute $a = \min x_2 + y_2$ s.t.

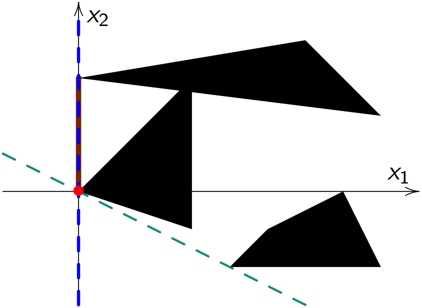
$$x_1 + y_1 = 1 \wedge A\mathbf{x} \geq \mathbf{c}x_0 \wedge x_0 \geq 0 \wedge B\mathbf{y} \geq \mathbf{d}y_0 \wedge y_0 \geq 0$$

(Cone of hull is sum of cones in homogeneous space)

isl Operation: Closed Convex Hull—Wrapping

$H = \text{conv.hull}(S_1 \cup S_2)$ $\quad S_1 = \{ \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \}$ $\quad S_2 = \{ \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \}$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$



  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found

isl Operation: Closed Convex Hull—Wrapping

$$H = \mathrm{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed $\Rightarrow$ change perspective

isl Operation: Closed Convex Hull—Wrapping

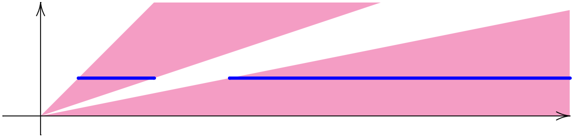
$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge ⇒ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed ⇒ change perspective

isl Operation: Closed Convex Hull—Wrapping

$H = \mathrm{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed $\Rightarrow$ change perspective

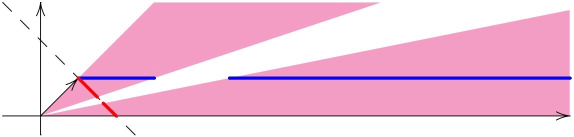


  - Consider cones in homogeneous space

isl Operation: Closed Convex Hull—Wrapping

$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge ⇒ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed ⇒ change perspective

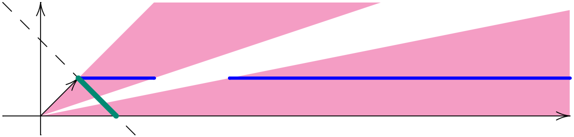


  - Consider cones in homogeneous space
  - Take other homogeneous direction ⇒ union of polytopes

isl Operation: Closed Convex Hull—Wrapping

$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge ⇒ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed ⇒ change perspective

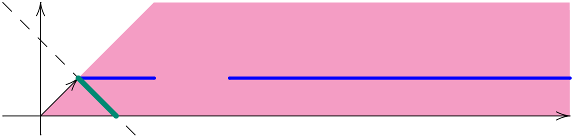


  - Consider cones in homogeneous space
  - Take other homogeneous direction ⇒ union of polytopes
  - Compute convex hull

`isl` Operation: Closed Convex Hull—Wrapping

$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge ⇒ new facet constraint $x_2 \geq ax_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed ⇒ change perspective



  - Consider cones in homogeneous space
  - Take other homogeneous direction ⇒ union of polytopes
  - Compute convex hull
  - Convert back

isl Operation: Closed Convex Hull—Wrapping

$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge ⇒ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed ⇒ change perspective

  - Consider cones in homogeneous space
  - Take other homogeneous direction ⇒ union of polytopes
  - Compute convex hull
  - Convert back

isl Operation: Closed Convex Hull—Wrapping

$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge ⇒ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed ⇒ change perspective

  - Consider cones in homogeneous space
  - Take other homogeneous direction ⇒ union of polytopes
  - Compute convex hull
  - Convert back

isl Operation: Closed Convex Hull—Wrapping

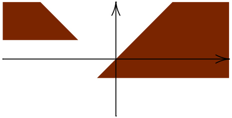
$$H = \operatorname{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed $\Rightarrow$ change perspective
  - Consider cones in homogeneous space
  - Take other homogeneous direction $\Rightarrow$ union of polytopes
  - Compute convex hull
  - Convert back
- S_1 and S_2 are pointed (R_i recession cone of S_i)
  $\Rightarrow$ project out lineality $H = \operatorname{lin.hull}(R_1 \cap -R_2)$

isl Operation: Closed Convex Hull—Wrapping

$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

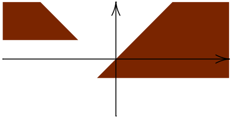
- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed $\Rightarrow$ change perspective
  - Consider cones in homogeneous space
  - Take other homogeneous direction $\Rightarrow$ union of polytopes
  - Compute convex hull
  - Convert back
- S_1 and S_2 are pointed (R_i recession cone of S_i)
  $\Rightarrow$ project out lineality $H = \text{lin.hull}(R_1 \cap -R_2)$

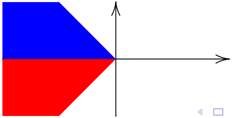


isl Operation: Closed Convex Hull—Wrapping

$$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed $\Rightarrow$ change perspective
  - Consider cones in homogeneous space
  - Take other homogeneous direction $\Rightarrow$ union of polytopes
  - Compute convex hull
  - Convert back
- S_1 and S_2 are pointed (R_i recession cone of S_i)
  $\Rightarrow$ project out lineality $H = \text{lin.hull}(R_1 \cap -R_2)$

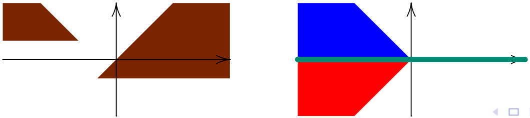




isl Operation: Closed Convex Hull—Wrapping

$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$

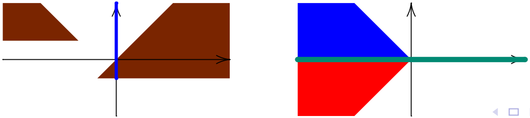
- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed $\Rightarrow$ change perspective
  - Consider cones in homogeneous space
  - Take other homogeneous direction $\Rightarrow$ union of polytopes
  - Compute convex hull
  - Convert back
- S_1 and S_2 are pointed (R_i recession cone of S_i)
  $\Rightarrow$ project out lineality $H = \text{lin.hull}(R_1 \cap -R_2)$



isl Operation: Closed Convex Hull—Wrapping

$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$

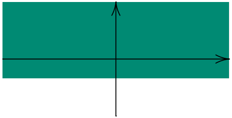
- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed $\Rightarrow$ change perspective
  - Consider cones in homogeneous space
  - Take other homogeneous direction $\Rightarrow$ union of polytopes
  - Compute convex hull
  - Convert back
- S_1 and S_2 are pointed (R_i recession cone of S_i)
  $\Rightarrow$ project out lineality $H = \text{lin.hull}(R_1 \cap -R_2)$

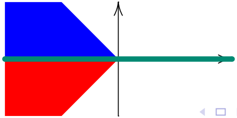


isl Operation: Closed Convex Hull—Wrapping

$H = \text{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{ \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \} \qquad S_2 = \{ \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \}$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq a x_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed $\Rightarrow$ change perspective
  - Consider cones in homogeneous space
  - Take other homogeneous direction $\Rightarrow$ union of polytopes
  - Compute convex hull
  - Convert back
- S_1 and S_2 are pointed (R_i recession cone of S_i)
  $\Rightarrow$ project out lineality $H = \text{lin.hull}(R_1 \cap -R_2)$





isl Operation: Closed Convex Hull—Wrapping

$$H = \mathrm{conv.hull}(S_1 \cup S_2) \qquad S_1 = \{\, \mathbf{x} \mid A\mathbf{x} \geq \mathbf{c} \,\} \qquad S_2 = \{\, \mathbf{x} \mid B\mathbf{x} \geq \mathbf{d} \,\}$$

- S_1 and S_2 are polytopes (FLL2000)
  - Assume $x_1 \geq 0$ defines a facet and $x_2 \geq 0$ a ridge on the facet
  - Wrap facet around ridge $\Rightarrow$ new facet constraint $x_2 \geq ax_1$
  - Repeat for all ridges
  - Ridges found through recursive application
  - Repeat for new facets until all facets found
- H is pointed $\Rightarrow$ change perspective
  - Consider cones in homogeneous space
  - Take other homogeneous direction $\Rightarrow$ union of polytopes
  - Compute convex hull
  - Convert back
- S_1 and S_2 are pointed (R_i recession cone of S_i)
  $\Rightarrow$ project out $\mathrm{lineality}\, H = \mathrm{lin.hull}(R_1 \cap -R_2)$
- S_1 or S_2 has non-trivial lineality space
  $\Rightarrow$ project out $\mathrm{lineality}\, S_1$ and $\mathrm{lineality}\, S_2$

Improved Code Generation using CLooG

Using PolyLib as a backend:

```
for (p1=0;p1<=floord(8*N+63,32);p1++) {
  for (p3=max(max(max(max(0,ceild(-32*p1-27,4)),
       ceild(512*p1-128*N-975,16)),ceild(28*p1-7*N-20,36)),
       ceild(60*p1-15*N-44,68));
       p3<=min(min(floord(4*M+47,16),floord(24*p1+5*M+36,20)),
       floord(136*p1+31*M+224,124));p3++) {
    if ((p1 >= 0) && (p1 <= floord(N-1,4))) {
      for (p5=max(0,4*p3);p5<=min(M-1,4*p3+3);p5++) {
        for (p7=max(0,4*p1);p7<=min(N-1,4*p1+3);p7++) {
          S9(p3,p5,p1,p7); /* ... */
```

Improved Code Generation using CLooG

Using PolyLib as a backend:

```
for (p1=0;p1<=floord(8*N+63,32);p1++) {
  for (p3=max(max(max(max(0,ceild(-32*p1-27,4)),
       ceild(512*p1-128*N-975,16)),ceild(28*p1-7*N-20,36)),
       ceild(60*p1-15*N-44,68));
       p3<=min(min(floord(4*M+47,16),floord(24*p1+5*M+36,20)),
       floord(136*p1+31*M+224,124));p3++) {
    if ((p1 >= 0) && (p1 <= floord(N-1,4))) {
      for (p5=max(0,4*p3);p5<=min(M-1,4*p3+3);p5++) {
        for (p7=max(0,4*p1);p7<=min(N-1,4*p1+3);p7++) {
          S9(p3,p5,p1,p7); /* ... */
```

Using isl as a backend:

```
for (p1=0;p1<=floord(N+7,4);p1++) {
  for (p3=max(0,ceild(4*p1-N+1,4));
       p3<=min(floord(M+11,4),floord(4*p1+M+3,4));p3++) {
    if (p1 <= floord(N-1,4)) {
      for (p5=4*p3;p5<=min(M-1,4*p3+3);p5++) {
        for (p7=4*p1;p7<=min(N-1,4*p1+3);p7++) {
          S9(p3,p5,p1,p7); /* ... */
```

CLooG Speed Comparison

	PolyLib-64	PolyLib-gmp	isl-gmp
Example from previous slide (from Harald Devos)	0.15s	0.31s	0.18s
CLooG test suite	5.1s	11.4s	7.5s
Simple tiling example	1.11s	2.63s	1.11s
Extreme tiling example	14.6s	28.5s	5.15s
LU example	0.86s	1.88s	0.35s
Sobel example (from Harald Devos)	0.62s	1.64s	0.15s

(Tiling examples from Uday K Bondhugula)

Conclusion

- `isl`: a new integer set library
- currently used in
  - equivalence checking tool
  - `CLooG`
    - Produces better code than PolyLib backend
    - Comparable in speed or faster than PolyLib backend
- explicit support for existentially quantified variables
- uses PIP for solving (P)ILP problems
- all computations in exact integer arithmetic using GMP
- built-in incremental LP solver
- released under LGPL license