A Polyhedral Compilation Library with Explicit Disequality Constraints

Sven Verdoolaege

Cerebras Systems

January 17, 2024
Outline

1 Motivation and Introduction

2 Related Work

3 Disequality Constraints
   - Internal Representation
   - Hidden Assumptions
   - Incremental LP Solver
   - Emptiness and Sampling
   - Redundant Local Variables
   - Parametric Integer Programming
   - Other Operations

4 Experimental Results

5 Conclusion
Outline

1 Motivation and Introduction

2 Related Work

3 Disequality Constraints
   - Internal Representation
   - Hidden Assumptions
   - Incremental LP Solver
   - Emptiness and Sampling
   - Redundant Local Variables
   - Parametric Integer Programming
   - Other Operations

4 Experimental Results

5 Conclusion
A Motivating Example (Kulkarni and Kruse June 2022)

```java
for (int i = 0; i < n; i+=1) {
    if (i == p0)
        continue;
    if (i == p1)
        continue;
    if (i == p2)
        continue;
    // ...
    Stmt(i);
}
```
A Motivating Example (Kulkarni and Kruse June 2022)

```java
for (int i = 0; i < n; i++) {
    if (i == p0)
        continue;
    if (i == p1)
        continue;
    if (i == p2)
        continue;
    // ...
    Stmt(i);
}
```

Instance set: \( \{ \text{Stmt}[i] : 0 \leq i < n \wedge i \neq p0 \wedge i \neq p1 \wedge i \neq p2 \wedge \ldots \} \)
A Motivating Example (Kulkarni and Kruse June 2022)

```java
for (int i = 0; i < n; i+=1) {
    if (i == p0)
        continue;
    if (i == p1)
        continue;
    if (i == p2)
        continue;
    // ...
    Stmt(i);
}
```

Instance set: \{ Stmt[i] : 0 \leq i < n \land i \neq p0 \land i \neq p1 \land i \neq p2 \land ... \}
\{ Stmt[i] : 0 \leq i < n \land (i < p0 \lor i > p0) \land (i < p1 \lor i > p1) \land (i < p2 \lor i > p2) \land ... \}

\Rightarrow \text{expansion causes explosion in representation}
Another Motivating Example (Klebanov 2015)

\[
\text{card} \{ [r1]; [r2]; [r3]; [r4]; [r5]; [r10] \}
\]
Another Motivating Example (Klebanov 2015)

card \{ [r1]; [r2]; [r3]; [r4]; [r5]; [r10] \}  

\[
\begin{aligned}
6 & \text{ if } r_1 \neq r_2, r_3, r_4, r_5, r_{10} \land r_2 \neq r_3, r_4, r_5, r_{10} \land r_3 \neq r_4, r_5, r_{10} \land r_4 \neq r_5, r_{10} \land r_5 \neq r_{10} \\
5 & \text{ if } (r_1 = r_2 \land r_2 \neq r_3, r_4, r_5, r_{10} \land r_3 \neq r_4, r_5, r_{10} \land r_4 \neq r_5, r_{10} \land r_5 \neq r_{10}) \lor \ldots \\
& \cdots \\
1 & \text{ if } r_1 = r_2 = r_3 = r_4 = r_5 = r_{10}
\end{aligned}
\]
Another Motivating Example (Klebanov 2015)

card \{ [r1]; [r2]; [r3]; [r4]; [r5]; [r10] \}

\[
\begin{align*}
6 & \text{ if } r1 \neq r2, r3, r4, r5, r10 \land r2 \neq r3, r4, r5, r10 \land r3 \neq r4, r5, r10 \land r4 \neq r5, r10 \land r5 \neq r10 \\
5 & \text{ if } (r1 = r2 \land r2 \neq r3, r4, r5, r10 \land r3 \neq r4, r5, r10 \land r4 \neq r5, r10 \land r5 \neq r10) \lor \ldots \\
\vdots \\
1 & \text{ if } r1 = r2 = r3 = r4 = r5 = r10 \\
\Rightarrow & \text{ large representation even with explicit disequality constraints} \\
\Rightarrow & \text{ a lot worse without}
\end{align*}
\]
Core Representation of Polyhedral Compilation Library

Conjunction of affine inequality constraints

\[ \{ z : A z + a \geq 0 \} \]

+ unions of such sets

No explicit representation for disequality constraints

This applies to libraries

- not supporting existentially quantified variables:
  - PolyLib (Wilde 1993)
  - PPL (Bagnara et al. 2008)

- supporting existentially quantified variables:
  - Omega (Kelly et al. Nov. 1996)
  - isl (V. 2010)
  - Omega+ (Chen June 2012)
Core Representation of Polyhedral Compilation Library

Conjunction of affine inequality constraints

\[ \{ z : Az + a \geq 0 \} \]

+ unions of such sets

No explicit representation for disequality constraints

How about equality constraints?
Core Representation of Polyhedral Compilation Library

Conjunction of affine inequality constraints

\[ \{ \mathbf{z} : A\mathbf{z} + \mathbf{a} \geq \mathbf{0} \land B\mathbf{z} + \mathbf{b} = \mathbf{0} \} \]

+ unions of such sets

No explicit representation for disequality constraints

How about equality constraints?
Core Representation of Polyhedral Compilation Library

Conjunction of affine inequality constraints

\[ \{ z : A z + a \geq 0 \land B z + b = 0 \} \]

+ unions of such sets

No explicit representation for disequality constraints

How about equality constraints? Not strictly needed but still used

- do not change expressivity
- \( n \) equality constraints replace \( n + 1 \) to \( 2n \) inequality constraints
- every (independent) equality constraint reduces effective dimensionality
Core Representation of Polyhedral Compilation Library

Conjunction of affine inequality constraints

\[ \{ \mathbf{z} : A\mathbf{z} + \mathbf{a} \geq \mathbf{0} \land B\mathbf{z} + \mathbf{b} = \mathbf{0} \} \]

+ unions of such sets

No explicit representation for disequality constraints

How about equality constraints? Not strictly needed but still used

- do not change expressivity
- \( n \) equality constraints replace \( n + 1 \) to \( 2n \) inequality constraints
- every (independent) equality constraint reduces effective dimensionality

Why not disequality constraints?

- do not change expressivity
- \( n \) disequality constraints avoid split into 2 to \( 2^n \) disjuncts
Outline

1 Motivation and Introduction

2 Related Work

3 Disequality Constraints
   - Internal Representation
   - Hidden Assumptions
   - Incremental LP Solver
   - Emptiness and Sampling
   - Redundant Local Variables
   - Parametric Integer Programming
   - Other Operations

4 Experimental Results

5 Conclusion
Seater and Wonnacott (2005)

Detect “inert” disequality constraints

⇒ disequality constraints that can be ignored (in terms of emptiness)
⇒ disequality constraints that involve unbounded direction
Seater and Wonnacott (2005)

Detect “inert” disequality constraints

⇒ disequality constraints that can be ignored (in terms of emptiness)
⇒ disequality constraints that involve unbounded direction
Seater and Wonnacott (2005)

Detect “inert” disequality constraints
⇒ disequality constraints that can be ignored (in terms of emptiness)
⇒ disequality constraints that involve unbounded direction
Seater and Wonnacott (2005)

Detect “inert” disequality constraints

⇒ disequality constraints that can be ignored (in terms of emptiness)
⇒ disequality constraints that involve unbounded direction

![Diagram showing inert disequality constraints]

None inert  Some inert
Seater and Wonnacott (2005)

Detect “inert” disequality constraints

⇒ disequality constraints that can be ignored (in terms of emptiness)
⇒ disequality constraints that involve unbounded direction

None inert

Some inert

All inert
An equivalent approach would be to simply allow negated equality constraints in simplified relations. This approach could be taken even further, to allow more general negated constraints, or other formulas that cannot be handled efficiently.

We do not currently have an implementation of our algorithms, and thus we do not have empirical verification that they are either fast or effective in practice. Given the nature of the changes discussed in the previous section, we do not expect to have an implementation any time soon.
Kulkarni and Kruse (June 2022)

\{ \text{Stmt}[i] : 0 \leq i < n \land i \neq p_0 \land i \neq p_1 \land i \neq p_2 \land \ldots \} 

Polyhedral binary decision diagram, PBDD
- internal nodes: affine (in)equality constraints
- terminal nodes: \textbf{IN}: in set; \textbf{OUT}: not in set

\Rightarrow \text{ allows negation of (conjunction of) affine constraints}
\text{(disequality constraint is special case)}

However
- limited number of supported operations (intersection, union, subtraction, complement)
- revert to isl (with expansion) for other operations
- no support for existentially quantified variables
Outline

1. Motivation and Introduction
2. Related Work
3. Disequality Constraints
   - Internal Representation
   - Hidden Assumptions
   - Incremental LP Solver
   - Emptiness and Sampling
   - Redundant Local Variables
   - Parametric Integer Programming
   - Other Operations
4. Experimental Results
5. Conclusion
Explicit Disequality Constraints

Main changes:

- extend internal representation
- resolve hidden assumptions
- adjust some core algorithms

For example:

\[
\{ S_5[i =0:99] \rightarrow T[i] : i \neq 57 \} \% \{ S_5[i =0:99] : i \neq 57 \};
\]

⇒

\[
\{ S_5[i] \rightarrow T[i] : i \geq 58 \text{ or } i \leq 56 \};
\]

new expression type in result of AST generation

for

\[
(int c0 = 1; c0 <= 9; c0 += 1) \{
\text{if } (c0 \neq 5) \{
\text{for } (int c1 = 1; c1 <= 9; c1 += 1)
S0(c0, c1);
\text{else}
\}
\]

// ...
Explicit Disequality Constraints

Main changes:
- extend internal representation
- resolve hidden assumptions
- adjust some core algorithms

Changes are (mostly) transparent to user of isl
- results of some heuristics-based operations may change
- new expression type in result of AST generation
Explicit Disequality Constraints

Main changes:

- extend internal representation
- resolve hidden assumptions
- adjust some core algorithms

Changes are (mostly) transparent to user of isl

- results of some heuristics-based operations may change

For example: pet test case

\[
\{ S_5[i=0:99] \rightarrow T[i] : i \neq 57 \} \% \{ S_5[i=0:99] : i \neq 57 \};
\Rightarrow \{ S_5[i] \rightarrow T[i] : i \geq 58 \text{ or } i \leq 56 \}; \text{ now: } \{ S_5[i] \rightarrow T[i] \}
\]

- new expression type in result of AST generation
Explicit Disequality Constraints

Main changes:
- extend internal representation
- resolve hidden assumptions
- adjust some core algorithms

Changes are (mostly) transparent to user of isl
- results of some heuristics-based operations may change
  For example: pet test case
  \[
  \{ S_5[i=0:99] \rightarrow T[i] : i \neq 57 \} \ % \ \{ S_5[i=0:99] : i \neq 57 \};
  \Rightarrow \{ S_5[i] \rightarrow T[i] : i \geq 58 \ or \ i \leq 56 \}; \ now: \ \{ S_5[i] \rightarrow T[i] \} \]
- new expression type in result of AST generation

```java
for (int c0 = 1; c0 <= 9; c0 += 1) {
    if (c0 != 5) {
        for (int c1 = 1; c1 <= 9; c1 += 1)
            s0(c0, c1);
    } else {
```
Extend Internal Representation

Basic set:

\[
\{ z : Az + a \geq 0 \land Bz + b = 0 \}
\]

+ unions of basic sets
Extend Internal Representation

Basic set:

\[ \{ z : Az + a \geq 0 \land Bz + b = 0 \land Cz + c \neq 0 \} \]

+ unions of basic sets
Extend Internal Representation

Basic set:

$$\{ z : Az + a \geq 0 \land Bz + b = 0 \land Cz + c \neq 0 \}$$

+ unions of basic sets

Simplifications:

- $m f(z) + c \neq 0$
  
  \[ \Rightarrow \text{drop constraint if } m \text{ does not divide } c \]
Extend Internal Representation

Basic set:

\[ \{ z : Az + a \geq 0 \land Bz + b = 0 \land Cz + c \neq 0 \} \]

+ unions of basic sets

Simplifications:

- \( m f(z) + c \neq 0 \)
  - \( \Rightarrow \) drop constraint if \( m \) does not divide \( c \)

- \( c \neq 0 \)
  - \( \Rightarrow \) drop constraint if \( c \) is not zero
  - \( \Rightarrow \) mark basic set empty if \( c \) is zero
Extend Internal Representation

Basic set:
\[
\{ z : A z + a \geq 0 \land B z + b = 0 \land C z + c \neq 0 \}
\]
+ unions of basic sets

Simplifications:
- \( m f(z) + c \neq 0 \)
  \( \Rightarrow \) drop constraint if \( m \) does not divide \( c \)
- \( c \neq 0 \)
  \( \Rightarrow \) drop constraint if \( c \) is not zero
  \( \Rightarrow \) mark basic set empty if \( c \) is zero
- only exact duplicates (or opposites) of disequality constraints can be removed
Extend Internal Representation

Basic set:

\[
\left\{ \begin{array}{l}
z : A z + a \geq 0 \land B z + b = 0 \land C z + c \neq 0 \\
\end{array} \right. 
\]

+ unions of basic sets

Simplifications:

- \( m f(z) + c \neq 0 \)
  - \( \Rightarrow \) drop constraint if \( m \) does not divide \( c \)

- \( c \neq 0 \)
  - \( \Rightarrow \) drop constraint if \( c \) is not zero
  - \( \Rightarrow \) mark basic set empty if \( c \) is zero

- only exact duplicates (or opposites) of disequality constraints can be removed
  - \( f(z) + c \neq 0 \)
  - \( f(z) + a \geq 0 \)
    - \( \Rightarrow \) replace by \( f(z) + a - 1 \geq 0 \) if \( a = c \)
    - \( \Rightarrow \) drop disequality if \( a < c \)
Resolve Hidden Assumptions

Main hidden assumption in isl: basic set is convex

Implications:
- all integer values between min/max rational values are in basic set
- simple hull operation can convert 1-disjunct set into basic set
  - introduce special operation for conversion
  - simple hull operation drops disequality constraints
  - another operation for shared constraints needed?
Disequality Constraints in Tableau

Introduce a non-zero variable for each disequality constraint

- a non-zero variable does not participate in pivoting
  - ⇒ always a row variable
- if non-zero variable can attain only negative or only positive values
  - ⇒ non-zero variable is redundant and can be removed
- if non-zero variable can obviously only attain zero value
  - ▶ zero values for all remaining columns
  - ⇒ tableau is empty
- sample point only valid if all non-zero variables have non-zero value

\[
\begin{align*}
  f_1 &= x + y \geq 0 \\
  f_2 &= x - 10 \geq 0 \\
  f_3 &= y - 5 \geq 0 \\
  f_4 &= -y + 5 \geq 0 \\
  f_5 &= y - 5 \neq 0
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$f_2$</td>
<td>$-10$</td>
<td>1</td>
</tr>
<tr>
<td>$f_3$</td>
<td>$-5$</td>
<td>0</td>
</tr>
<tr>
<td>$f_4$</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>$f_5$</td>
<td>$-5$</td>
<td>0</td>
</tr>
</tbody>
</table>
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements

Procedure

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0)
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation
- perform backtracking search in tableau on bounded dimensions (can fail)

- pick some corresponding value for unbounded dimensions (always succeeds)
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements

Procedure

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0)
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation

\[
\begin{align*}
\{ [x, y] : 1 \leq 2x - y \leq 4 \}
\end{align*}
\]
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements

Procedure

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0)
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation

\[
\begin{align*}
\{ [x, y] : 1 \leq 2x - y \leq 4 \} \\
\{ [x, y] : 0 \leq 2x - y \leq 0 \}
\end{align*}
\]
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements

Procedure

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0)
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation

\[
\begin{align*}
\{ & [x, y] : 1 \leq 2x - y \leq 4 \} \\
\{ & [x, y] : 0 \leq 2x - y \leq 0 \} \\
x' &= 2x - y \\
y' &= y
\end{align*}
\]
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements

Procedure

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0)
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation

\[
\begin{align*}
\{ [x, y] : 1 \leq 2x - y \leq 4 \} \\
\{ [x, y] : 0 \leq 2x - y \leq 0 \}
\end{align*}
\]

\[
x' = 2x - y \\
y' = y
\]
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements

Procedure

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0)
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation

\[
\begin{align*}
  \{ [x, y] : 1 \leq 2x - y \leq 4 \} \\
  \{ [x, y] : 0 \leq 2x - y \leq 0 \}
\end{align*}
\]

\[
x' = 2x - y \\
y' = y
\]
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements

Procedure

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0)
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation

\[
\begin{align*}
\{ [x, y] : 1 \leq 2x - y \leq 4 \} \\
\{ [x, y] : 0 \leq 2x - y \leq 0 \}
\end{align*}
\]

\[
x' = 2x - y \\
y' = y
\]
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements

Procedure

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0)
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation

\[
\begin{align*}
\{ [x, y] & : 1 \leq 2x - y \leq 4 \} \\
\{ [x, y] & : 0 \leq 2x - y \leq 0 \} \\
x' &= 2x - y \\
y' &= y
\end{align*}
\]
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

⇒ restrict set to points that have entire unit cube included in original set

⇒ pick rational element in restricted set

⇒ round up
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

⇒ restrict set to points that have entire unit cube included in original set
⇒ pick rational element in restricted set
⇒ round up

\[ \{ [x, y] : y \leq 2x \land x \leq 2y - 1 \} \]
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

⇒ restrict set to points that have entire unit cube included in original set

⇒ pick rational element in restricted set

⇒ round up

\[ \{ [x, y] : y \leq 2x \land x \leq 2y - 1 \} \]
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

⇒ restrict set to points that have entire unit cube included in original set
⇒ pick rational element in restricted set
⇒ round up

\[
\begin{align*}
\{ [x, y] : y & \leq 2x \land x \leq 2y - 1 \} \\
\{ [x, y] : y & \leq 2x - 1 \land x \leq 2y - 2 \}
\end{align*}
\]
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

$\Rightarrow$ restrict set to points that have entire unit cube included in original set

$\Rightarrow$ pick rational element in restricted set $(4/3, 5/3)$

$\Rightarrow$ round up

$\{ [x, y] : y \leq 2x \land x \leq 2y - 1 \}$

$\{ [x, y] : y \leq 2x - 1 \land x \leq 2y - 2 \}$
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

⇒ restrict set to points that have entire unit cube included in original set
⇒ pick rational element in restricted set \((4/3, 5/3)\)
⇒ round up \((2, 2)\)
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements

Procedure

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0)
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation
- perform backtracking search in tableau on bounded dimensions (can fail)

- pick some corresponding value for unbounded dimensions (always succeeds)
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements

Procedure

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0) ignoring disequality constraints
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation
- perform backtracking search in tableau on bounded dimensions (can fail)
  - drop disequality constraints involving unbounded dimension ("inert")
  - skip values violating any other disequality constraint
- pick some corresponding value for unbounded dimensions (always succeeds)


**Emptiness and Sampling**

- Sampling picks an integer element
- Set is empty if it has no integer elements

**Procedure**

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0) ignoring *disequality* constraints
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation

\[
\begin{align*}
\{ & [x, y] : 1 \leq 2x - y \leq 4 \\
\{ & [x, y] : 0 \leq 2x - y \leq 0 \\
\end{align*}
\]

\[
\begin{align*}
x' &= 2x - y \\
y' &= y
\end{align*}
\]
Emptiness and Sampling

- Sampling picks an integer element
- Set is empty if it has no integer elements

Procedure

- trivial solution for 0D and 1D sets
- isolate bounded directions
  - compute recession cone (replace constant terms by 0) ignoring disequality constraints
  - (implicit) equality constraints determine bounded directions
  - perform unimodular transformation

\[
\begin{align*}
\{ [x, y] : 1 \leq 2x - y \leq 4 \} \\
\{ [x, y] : 0 \leq 2x - y \leq 0 \}
\end{align*}
\]

\[ x' = 2x - y \]
\[ y' = y \]
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

⇒ restrict set to points that have entire unit cube included in original set

⇒ pick rational element in restricted set

⇒ round up
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

⇒ restrict set to points that have entire \((1 + n^\neq)\)-cube included in original set

⇒ pick rational element in restricted set

⇒ round up (skipping violated disequality constraints)
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

⇒ restrict set to points that have entire \((1 + n^\neq)\)-cube included in original set

⇒ pick rational element in restricted set

⇒ round up (skipping violated disequality constraints)

\[
\{ [x, y] : y \leq 2x \land x \leq 2y - 1 \}
\]
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

⇒ restrict set to points that have entire \((1 + n^2)\)-cube included in original set

⇒ pick rational element in restricted set

⇒ round up (skipping violated disequality constraints)

\[ \{ [x, y] : y \leq 2x \land x \leq 2y - 1 \} \]
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

⇒ restrict set to points that have entire \((1 + n^\neq)-cube\) included in original set

⇒ pick rational element in restricted set

⇒ round up (skipping violated disequality constraints)

\[
\{ [x, y] : y \leq 2x \land x \leq 2y - 1 \}
\]

\[
\{ [x, y] : y \leq 2x - 3 \land x \leq 2y - 4 \}
\]
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)

⇒ restrict set to points that have entire \((1 + n\neq)\)-cube included in original set

⇒ pick rational element in restricted set \((10/3, 11/3)\)

⇒ round up (skipping violated disequality constraints)
Picking Element in Unbounded Set

Rational element can easily be picked in tableau (sample value, possibly non-integer values)
⇒ restrict set to points that have entire \((1 + n\neq)-\)cube included in original set
⇒ pick rational element in restricted set \((10/3, 11/3)\)
⇒ round up (skipping violated disequality constraints) \((4, 6)\)

\[
\begin{align*}
\{ [x, y] : y &\leq 2x \land x \leq 2y - 1 \} \\
\{ [x, y] : y &\leq 2x - 3 \land x \leq 2y - 4 \}
\end{align*}
\]
Redundant Local Variables

Basic set:

$$\{ \mathbf{x} : \exists \alpha : A_1 \mathbf{x} + A_2 \alpha + \mathbf{a} \geq \mathbf{0} \}$$

Some local variable $\alpha$ may be redundant

Some of these can be detected based purely on constraints

- consider all pairs of lower and upper bounds on variable $\alpha$

- if each pair admits an integer value
  - $\Rightarrow \alpha$ can be eliminated (using Fourier-Motzkin)
Redundant Local Variables

Basic set:
\[ \{ x : \exists \alpha : A_1 x + A_2 \alpha + a \geq 0 \} \]

Some local variable \( \alpha \) may be redundant

Some of these can be detected based purely on constraints

- consider all pairs of lower and upper bounds on variable \( \alpha \)
- \( \alpha \) is involved in \( n \) disequality constraints
- if each pair admits \( 1 + n \) integer values
  \( \Rightarrow \) \( \alpha \) can be eliminated (using Fourier-Motzkin)
Redundant Local Variables

Basic set:

\[ \{ \mathbf{x} : \exists \alpha : A_1 \mathbf{x} + A_2 \alpha + \mathbf{a} \geq \mathbf{0} \} \]

Some local variable \( \alpha \) may be redundant

Some of these can be detected based purely on constraints

- consider all pairs of lower and upper bounds on variable \( \alpha \)
- \( \alpha \) is involved in \( n \) disequality constraints
- if each pair admits \( 1 + n \) integer values
  \[ \Rightarrow \alpha \text{ can be eliminated (using Fourier-Motzkin)} \]

\[ \Rightarrow \text{potential trade-off between number of disjuncts and dimensionality of disjuncts} \]
Parametric Integer Programming

Compute lexicographic minimum of some variables $x$ in terms of other variables $n$

Two tableaux:
- main tableau in $x$ and $n$
- context tableau $n$

Pivoting in main tableau depends on sign of symbolic constant term in context tableau

$\Rightarrow$ requires context splits if constant term can attain both positive and negative values

$$R = \{ [i, j] : 0 \leq -i \leq N \land 0 \leq -j \leq -i \land 0 \leq k \leq 3N \land k = -i - 2j \}$$

lexmin $R =$

if $k < N$

$$[-k, 0]$$

else

$$\begin{cases} 
[k - 2 \left\lceil \frac{k+N}{2} \right\rceil, -k + \left\lceil \frac{k+N}{2} \right\rceil] 
\end{cases}$$
Parametric Integer Programming

Compute lexicographic minimum of some variables $x$ in terms of other variables $n$

Two tableaux:
- main tableau in $x$ and $n$
- context tableau $n$

Pivoting in main tableau depends on sign of symbolic constant term in context tableau
⇒ requires context splits if constant term can attain both positive and negative values

Keep track of disequality constraints in tableaux

If disequality constraint $g(n, x) \neq 0$ may be violated by potential solution
⇒ split context into 2 cases
  ▶ $f(n) \neq 0$ (implying $g(n, x) \neq 0$ is not violated)
    ⇒ proceed with other disequality constraints
  ▶ $f(n) = 0$ (implying $g(n, x) \neq 0$ is violated)
    ⇒ compute two solutions, for $g(n, x) \geq 1$ and $g(n, x) \leq -1$
    ⇒ take minimum of two solutions

Splitting $g(n, x) \neq 0$ up front computes same minimum but then cost is always incurred
Some Other Operations

- Preparation for counting using `barvinok` (V., Seghir, et al. June 2007)
Some Other Operations

- Preparation for counting using barvinok

\[
\begin{align*}
&= X \\
&= \bigtriangledown \\
&= \square \\
&= - - + .
\end{align*}
\]

- Transitive closure approximation
  Basic sets do not have to be split but result may be less accurate
Some Other Operations

- Preparation for counting using barvinok

- Transitive closure approximation
  Basic sets do not have to be split but result may be less accurate

- Scheduling
  Disequality constraints essentially ignored
Outline

1. Motivation and Introduction
2. Related Work
3. Disequality Constraints
   - Internal Representation
   - Hidden Assumptions
   - Incremental LP Solver
   - Emptiness and Sampling
   - Redundant Local Variables
   - Parametric Integer Programming
   - Other Operations
4. Experimental Results
5. Conclusion
PBDD versus isl with Explicit Disequality Constraints

\[
\{ \left[ i \right] : \bigwedge_{j \leq n \neq i} i \neq p_j \}\]

Note: construction times with PBDD and isl not directly comparable.
PBDD versus isl with Explicit Disequality Constraints

\[ \{ [i] : \bigwedge_{j \leq n} i \neq p_j \} \]

Note: construction times with PBDD and isl not directly comparable
Full Polyhedral Compilation Flow

```java
for (int i = 0; i < n; ++i) {
    if (i == p0 || i == p1 || i == p2)
        continue;
    A[i] = i;
}
for (int i = 0; i < n; ++i) {
    if (i == p0 || i == p1 || i == p2)
        continue;
    B[i] = A[i];
}
```

PPCG (V., Juega, et al. 2013) output:

```java
for (int c0 = 0; c0 < n; c0 += 1)
    if (c0 != p0 && c0 != p1 && c0 != p2) {
        A[c0] = (c0);
        B[c0] = A[c0];
    }
```

(No changes required to PPCG)

Involves
- construction of polyhedral model
- dependence analysis
- scheduling
- AST generation

![Graph showing time in seconds vs. number of disequality constraints]

- enable disequality
- disable disequality
Outline

1. Motivation and Introduction
2. Related Work
3. Disequality Constraints
   - Internal Representation
   - Hidden Assumptions
   - Incremental LP Solver
   - Emptiness and Sampling
   - Redundant Local Variables
   - Parametric Integer Programming
   - Other Operations
4. Experimental Results
5. Conclusion
Conclusion

Supporting explicit \textit{disequality} constraints in a polyhedral compilation library is feasible

- requires only conceptually minor adjustments
- in some cases simply delaying split to where it becomes relevant
- can dramatically reduce size of representation
- mostly transparent to the user

Some trade-offs involved, e.g.,

- elimination of redundant local variables
- accuracy of transitive closure approximation

Perhaps useful to consider other explicit constraints, e.g.,

- lexicographic constraints
Appendix

Outline

6 Appendix

- Incremental LP Solver
- References
Incremental LP Solver

Core representation: tableau

Given

\{ z : Az + a \geq 0 \}

with \( n \) variables \( z \) and \( m \) constraints \( Az + a \geq 0 \)

- introduce a non-negative variable \( f_i \) for each affine expression
- tableau writes \( m \) variables in terms of \( n \) variables
- initially, \( f \) in terms of \( z \)

\[
\begin{align*}
f_1 &= x + y \geq 0 \\
f_2 &= x - 10 \geq 0 \\
f_3 &= y - 5 \geq 0 \\
f_4 &= -y + 5 \geq 0
\end{align*}
\]

sample value: assign zero to all column variables

\[
\begin{array}{ccc}
x & y & \\
f_1 & 0 & 1 & 1 \\
f_2 & -10 & 1 & 0 \\
f_3 & -5 & 0 & 1 \\
f_4 & 5 & 0 & -1
\end{array}
\]
Incremental LP Solver
Core representation: tableau

Given

\[ \{ z : A z + a \geq 0 \} \]

with \( n \) variables \( z \) and \( m \) constraints \( A z + a \geq 0 \)
- introduce a non-negative variable \( f_i \) for each affine expression
- tableau writes \( m \) variables in terms of \( n \) variables
- initially, \( f \) in terms of \( z \)

\[
\begin{align*}
f_1 &= x + y \geq 0 \\
f_2 &= x - 10 \geq 0 \\
f_3 &= y - 5 \geq 0 \\
f_4 &= -y + 5 \geq 0
\end{align*}
\]

- sample value: assign zero to all column variables
  \( x = 0, y = 0, f_1 = 0, f_2 = -10, f_3 = -5, f_4 = 5 \)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>-10</td>
<td>1</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>-5</td>
<td>0</td>
</tr>
<tr>
<td>( f_4 )</td>
<td>5</td>
<td>-1</td>
</tr>
</tbody>
</table>
## Incremental LP Solver

\[
\begin{align*}
  f_1 &= x + y \geq 0 \\
  f_2 &= x - 10 \geq 0 \\
  f_3 &= y - 5 \geq 0 \\
  f_4 &= -y + 5 \geq 0
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$f_2$</td>
<td>-10</td>
<td>1</td>
</tr>
<tr>
<td>$f_3$</td>
<td>-5</td>
<td>0</td>
</tr>
<tr>
<td>$f_4$</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

- a pivot interchanges a row and a column variable

\[
\begin{align*}
  f_3 &= y - 5 \geq 0 \\
  \Rightarrow & \\
  y &= 5 + f_3 \\
  f_1 &= x + y + 5 \geq 0 \\
  f_2 &= x - 10 \geq 0 \\
  f_4 &= -y + 5 \geq 0
\end{align*}
\]

- a column variable that is known to be zero (e.g., $f_3$) can be "killed" ⇒ fixed to zero ⇒ no longer participates in pivoting

- sample value is "valid" if
  - all non-negative variables have non-negative value
  - all variables have integer value

⇒ current sample value not valid because

\[
f_2 = -10
\]
Incremental LP Solver

\[ f_1 = x + y \geq 0 \]
\[ f_2 = x - 10 \geq 0 \]
\[ f_3 = y - 5 \geq 0 \]
\[ f_4 = -y + 5 \geq 0 \]

<table>
<thead>
<tr>
<th></th>
<th>( x )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>-10</td>
<td>1</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>-5</td>
<td>0</td>
</tr>
<tr>
<td>( f_4 )</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

- a pivot interchanges a row and a column variable

\[ f_3 = -5 + y \Rightarrow y = 5 + f_3, \quad f_1 = 5 + x + f_3, \quad f_4 = -f_3 \]

A column variable that is known to be zero (e.g., \( f_3 \)) can be "killed" \( \Rightarrow \) fixed to zero \( \Rightarrow \) no longer participates in pivoting

A sample value is "valid" if
- all non-negative variables have non-negative value
- all variables have integer value

\[ \Rightarrow \] current sample value not valid because \( f_2 = -10 \)
Incremental LP Solver

\[ f_1 = x + y \geq 0 \]
\[ f_2 = x - 10 \geq 0 \]
\[ f_3 = y - 5 \geq 0 \]
\[ f_4 = -y + 5 \geq 0 \]

<table>
<thead>
<tr>
<th></th>
<th>( x )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>-10</td>
<td>1</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>-5</td>
<td>0</td>
</tr>
<tr>
<td>( f_4 )</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

- a pivot interchanges a row and a column variable
  \[ f_3 = -5 + y \Rightarrow y = 5 + f_3, \quad f_1 = 5 + x + f_3, \quad f_4 = -f_3 \]
### Incremental LP Solver

\[
\begin{align*}
  f_1 &= x + y \geq 0 \\
  f_2 &= x - 10 \geq 0 \\
  f_3 &= y - 5 \geq 0 \\
  f_4 &= -y + 5 \geq 0
\end{align*}
\]

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f_1)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f_2)</td>
<td>-10</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f_3)</td>
<td>-5</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f_4)</td>
<td>5</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A pivot interchanges a row and a column variable:

\[
\begin{align*}
  f_3 &= -5 + y \\  y &= 5 + f_3 \\
  f_1 &= 5 + x + f_3 \\
  f_4 &= -f_3
\end{align*}
\]
Incremental LP Solver

\[
\begin{align*}
\mathbf{f}_1 &= x + y \geq 0 && \mathbf{f}_1 &\quad 0 \quad 1 \quad 1 \\
\mathbf{f}_2 &= x - 10 \geq 0 && \mathbf{f}_2 &\quad -10 \quad 1 \quad 0 \\
\mathbf{f}_3 &= y - 5 \geq 0 && \mathbf{f}_3 &\quad -5 \quad 0 \quad 1 \\
\mathbf{f}_4 &= -y + 5 \geq 0 && \mathbf{f}_4 &\quad 5 \quad 0 \quad -1
\end{align*}
\]

- a pivot interchanges a row and a column variable
  \[
  \mathbf{f}_3 = -5 + y \Rightarrow y = 5 + \mathbf{f}_3, \quad \mathbf{f}_1 = 5 + x + \mathbf{f}_3, \quad \mathbf{f}_4 = -\mathbf{f}_3
  \]
- a column variable that is known to be zero (e.g., \( \mathbf{f}_3 \)) can be “killed”
  \[
  \Rightarrow \text{fixed to zero} \\
  \Rightarrow \text{no longer participates in pivoting}
  \]
Appendix
Incremental LP Solver

In scalar form, the modified tableau is:

\[
\begin{align*}
\mathbf{f}_1 &= x + y \geq 0 \\
\mathbf{f}_2 &= x - 10 \geq 0 \\
\mathbf{f}_3 &= y - 5 \geq 0 \\
\mathbf{f}_4 &= -y + 5 \geq 0
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>(x)</th>
<th>(y)</th>
<th>(x)</th>
<th>(f_3)</th>
<th>(f_3)</th>
<th>(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_1)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(f_2)</td>
<td>-10</td>
<td>1</td>
<td>0</td>
<td>-10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(f_3)</td>
<td>-5</td>
<td>0</td>
<td>1</td>
<td>y</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>(f_4)</td>
<td>5</td>
<td>0</td>
<td>-1</td>
<td>(f_4)</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

- a pivot interchanges a row and a column variable
  \[
  f_3 = -5 + y \Rightarrow y = 5 + f_3, \quad f_1 = 5 + x + f_3, \quad f_4 = -f_3
  \]
- a column variable that is known to be zero (e.g., \(f_3\)) can be “killed”
  \[
  \Rightarrow \text{fixed to zero} \\
  \Rightarrow \text{no longer participates in pivoting}
  \]
### Incremental LP Solver

Let's consider the following system of linear inequalities:

\[
\begin{align*}
    f_1 &= x + y \geq 0 \\
    f_2 &= x - 10 \geq 0 \\
    f_3 &= y - 5 \geq 0 \\
    f_4 &= -y + 5 \geq 0
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>(x)</th>
<th>(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(f_2)</td>
<td>-10</td>
<td>1</td>
</tr>
<tr>
<td>(f_3)</td>
<td>-5</td>
<td>0</td>
</tr>
<tr>
<td>(f_4)</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(x)</th>
<th>(f_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_1)</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>(f_2)</td>
<td>-10</td>
<td>0</td>
</tr>
<tr>
<td>(f_3)</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>(f_4)</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

- A pivot interchanges a row and a column variable:
  \[f_3 = -5 + y \Rightarrow y = 5 + f_3, \ f_1 = 5 + x + f_3, \ f_4 = -f_3\]

- A column variable that is known to be zero (e.g., \(f_3\)) can be “killed”:
  - \(f_3\) fixed to zero
  - \(f_3\) no longer participates in pivoting

- Sample value is “valid” if:
  - All non-negative variables have non-negative values
  - All variables have integer value

  \(\Rightarrow\) current sample value not valid because \(f_2 = -10\)
Disequality Constraints in Tableau

Introduce a non-zero variable for each disequality constraint

- a non-zero variable does not participate in pivoting
  ⇒ always a row variable

- if non-zero variable can attain only negative or only positive values
  (while non-negative variables have non-negative values)
  ⇒ non-zero variable is redundant and can be removed

- if non-zero variable can obviously only attain zero value
  ▶ zero sample value
  ▶ all coefficients in non-killed columns are zero
  ⇒ tableau is empty

Effect on sample value validity

- all non-zero variables have non-zero value
- all non-negative variables have non-negative value
- all variables have integer value
Disequality Constraints in Tableau

Introduce a **non-zero** variable for each **disequality** constraint
- a **non-zero** variable does not participate in pivoting
  - \(\Rightarrow\) always a row variable
- if **non-zero** variable can attain only negative or only positive values
  (while **non-negative** variables have non-negative values)
  - \(\Rightarrow\) **non-zero** variable is redundant and can be removed
- if **non-zero** variable can obviously only attain zero value
  - zero sample value
  - all coefficients in non-killed columns are zero
  - \(\Rightarrow\) tableau is empty

\[
\begin{align*}
  f_1 &= x + y \geq 0 \\
  f_2 &= x - 10 \geq 0 \\
  f_3 &= y - 5 \geq 0 \\
  f_4 &= -y + 5 \geq 0
\end{align*}
\]

\[
\begin{array}{c|cc}
  \text{f} & x & y \\
  \hline
  f_1 & 0 & 1 \\
  f_2 & -10 & 1 \\
  f_3 & -5 & 1 \\
  f_4 & 5 & 0
\end{array}
\quad
\begin{array}{c|cc}
  \text{f} & x & f_3 \\
  \hline
  f_1 & 5 & 1 \\
  f_2 & -10 & 1 \\
  y & 5 & 0 \\
  f_4 & 0 & -1
\end{array}
\quad
\begin{array}{c|cc}
  \text{f} & f_3 & x \\
  \hline
  f_1 & 5 & 1 \\
  f_2 & -10 & 0 \\
  y & 5 & 1 \\
  f_4 & 0 & -1
\end{array}
\]

Effect on sample value validity
- sample value is "valid" if
  - all non-zero variables have non-zero value
  - all non-negative variables have non-negative value
  - all variables have integer value
Disequality Constraints in Tableau

Introduce a non-zero variable for each disequality constraint

- a non-zero variable does not participate in pivoting
  \[ \Rightarrow \text{ always a row variable} \]

- if non-zero variable can attain only negative or only positive values
  (while non-negative variables have non-negative values)
  \[ \Rightarrow \text{ non-zero variable is redundant and can be removed} \]

- if non-zero variable can obviously only attain zero value
  - zero sample value
  - all coefficients in non-killed columns are zero
  \[ \Rightarrow \text{ tableau is empty} \]

\[
\begin{align*}
f_1 &= x + y \geq 0 \\
f_2 &= x - 10 \geq 0 \\
f_3 &= y - 5 \geq 0 \\
f_4 &= -y + 5 \geq 0 \\
f_5 &= y - 5 \neq 0
\end{align*}
\]
Disequality Constraints in Tableau

Introduce a non-zero variable for each disequality constraint

- a non-zero variable does not participate in pivoting
  \[ \implies \text{always a row variable} \]
- if non-zero variable can attain only negative or only positive values
  (while non-negative variables have non-negative values)
  \[ \implies \text{non-zero variable is redundant and can be removed} \]
- if non-zero variable can obviously only attain zero value
  - zero sample value
  - all coefficients in non-killed columns are zero
  \[ \implies \text{tableau is empty} \]

\[
\begin{align*}
  f_1 &= x + y \geq 0 \\
  f_2 &= x - 10 \geq 0 \\
  f_3 &= y - 5 \geq 0 \\
  f_4 &= -y + 5 \geq 0 \\
  f_5 &= y - 5 \neq 0
\end{align*}
\]

\[
\begin{array}{ccc|c|c|c}
  & x & y & f_1 & x & f_3 & f_3 & x \\
  f_1 & 0 & 1 & 1 & f_1 & 5 & 1 & 1 & f_1 & 5 & 1 & 1 \\
  f_2 & -10 & 1 & 0 & f_2 & -10 & 1 & 0 & f_2 & -10 & 0 & 1 \\
  f_3 & -5 & 0 & 1 & y & 5 & 0 & 1 & y & 5 & 1 & 0 \\
  f_4 & 5 & 0 & -1 & f_4 & 0 & 0 & -1 & f_4 & 0 & -1 & 0 \\
  f_5 & -5 & 0 & 1 & & & & & & & & \\
\end{array}
\]

Effect on sample value validity

- sample value is "valid" if
  - all non-zero variables have non-zero value
  - all non-negative variables have non-negative value
  - all variables have integer value
Disequality Constraints in Tableau

Introduce a non-zero variable for each disequality constraint

- a non-zero variable does not participate in pivoting
  \[ \Rightarrow \text{ always a row variable} \]
- if non-zero variable can attain only negative or only positive values
  (while non-negative variables have non-negative values)
  \[ \Rightarrow \text{ non-zero variable is redundant and can be removed} \]
- if non-zero variable can obviously only attain zero value
  ▶ zero sample value
  ▶ all coefficients in non-killed columns are zero
  \[ \Rightarrow \text{ tableau is empty} \]

\[
\begin{align*}
f_1 &= x + y \geq 0 \\
f_2 &= x - 10 \geq 0 \\
f_3 &= y - 5 \geq 0 \\
f_4 &= -y + 5 \geq 0 \\
f_5 &= y - 5 \neq 0
\end{align*}
\]

\[
\begin{array}{c|cc}
f_1 & 0 & 1 & 1 \\
f_2 & -10 & 1 & 0 \\
f_3 & -5 & 0 & 1 \\
f_4 & 5 & 0 & -1 \\
f_5 & -5 & 0 & 1
\end{array}
\]

\[
\begin{align*}
f_1 & \quad x & f_3 \\
f_2 & \quad y & f_3 \\
f_3 & \quad y & f_3 \\
f_4 & \quad y & f_3 \\
f_5 & \quad y & f_3
\end{align*}
\]

Effect on sample value validity

- sample value is "valid" if
  ▶ all non-zero variables have non-zero value
  ▶ all non-negative variables have non-negative value
  ▶ all variables have integer value
Disequality Constraints in Tableau

Introduce a non-zero variable for each disequality constraint

- a non-zero variable does not participate in pivoting
  ⇒ always a row variable
- if non-zero variable can attain only negative or only positive values
  (while non-negative variables have non-negative values)
  ⇒ non-zero variable is redundant and can be removed
- if non-zero variable can obviously only attain zero value
  ▶ zero sample value
  ▶ all coefficients in non-killed columns are zero
  ⇒ tableau is empty

\[
\begin{align*}
f_1 &= x + y \geq 0 \\
f_2 &= x - 10 \geq 0 \\
f_3 &= y - 5 \geq 0 \\
f_4 &= -y + 5 \geq 0 \\
f_5 &= y - 5 \neq 0 \\
\end{align*}
\]
Disequality Constraints in Tableau

Introduce a non-zero variable for each disequality constraint

- a non-zero variable does not participate in pivoting
  - ⇒ always a row variable
- if non-zero variable can attain only negative or only positive values
  (while non-negative variables have non-negative values)
  - ⇒ non-zero variable is redundant and can be removed
- if non-zero variable can obviously only attain zero value
  - zero sample value
  - all coefficients in non-killed columns are zero
  - ⇒ tableau is empty

Effect on sample value validity

- sample value is “valid” if
  - all non-zero variables have non-zero value
  - all non-negative variables have non-negative value
  - all variables have integer value
References I


Klebanov, Vladimir (2015). *Personal communication*.

References II


References III


