Projection in Logic, CP, and Optimization

John Hooker
Carnegie Mellon University

Workshop on Logic and Search
Melbourne, 2017
Projection as a Unifying Concept

- **Projection** is a fundamental concept in **logic**, **constraint programming**, and **optimization**.
  - **Logical inference** is **projection** onto a subset of variables.
  - **Consistency maintenance** in CP is a **projection** problem.
  - **Optimization** is **projection** onto a cost variable.
Projection as a Unifying Concept

• **Projection** is a fundamental concept in **logic**, **constraint programming**, and **optimization**.
  - **Logical inference** is **projection** onto a subset of variables.
  - **Consistency maintenance** in CP is a **projection** problem.
  - **Optimization** is **projection** onto a cost variable.

• Recognizing this unity can lead to **faster search methods**.
  - In both logic and optimization.
Projection as a Unifying Concept

• Two fundamental projection methods occur across multiple fields.
Projection as a Unifying Concept

• Two fundamental projection methods occur across multiple fields.

• Fourier-Motzkin Elimination and generalizations.
  - Polyhedral projection.
  - Probability logic
  - Propositional logic (resolution)
  - Integer programming (cutting planes & modular arithmetic)
  - Some forms of consistency maintenance
Projection as a Unifying Concept

- Two fundamental projection methods occur across multiple fields.

- **Benders decomposition** and generalizations.
  - Optimization.
  - Probability logic (column generation)
  - Propositional logic (conflict clauses)
  - First-order logic (partial instantiation)
Outline

• Projection using **Fourier-Motzkin elimination**
• **Consistency maintenance** as projection
• Projection using **Benders decomposition**
What Is Projection?

• Projection yields a constraint set.
  - We project a constraint set onto a subset of its variables to obtain another constraint set.
What Is Projection?

• Projection yields a constraint set.
  - We project a constraint set onto a subset of its variables to obtain another constraint set.

• Formal definition
  - Let \( x = (x_1, \ldots, x_n) \)
  - Let \( \bar{x} = (x_1, \ldots, x_k), \ k < n \)
  - Let \( C \) be a constraint set.
  - The projection of \( C \) onto \( \bar{x} \) is a constraint set, containing only variables in \( \bar{x} \), whose satisfaction set is \( \{ \bar{x} \mid x \text{ satisfies } C \} \)
Projection Using Fourier-Motzkin Elimination and Its Generalizations
Polyhedral Projection

- We wish to project a polyhedron onto a subspace.
  - A method based on an idea of Fourier was proposed by Motzkin.
  - The basic idea of Fourier-Motzkin elimination can be used to compute projections in several contexts.

Fourier (1827)
Motzkin (1936)
Polyhedral Projection

- Eliminate variables we want to project out.
  - To project \( \{ x \mid Ax \geq b \} \) onto \( x_1, \ldots, x_k \)
    project out all variables except \( x_1, \ldots, x_k \)

  - To project out \( x_j \), eliminate it from pairs of inequalities:

    \[
    \begin{align*}
    c_0 x_j + c \bar{x} & \geq \gamma \left( \frac{1}{c_0} \right) \\
    -d_0 x_j + d \bar{x} & \geq \delta \left( \frac{1}{d_0} \right)
    \end{align*}
    \]

    \[
    \left( \frac{c}{c_0} + \frac{d}{d_0} \right) \bar{x} \geq \frac{\gamma}{c_0} + \frac{\delta}{d_0}
    \]

    where \( c_0, d_0 \geq 0 \)

  - Then remove all inequalities containing \( x_j \)
Polyhedral Projection

• Example
  - Project \(-2x_1 - x_2 \geq -4\) onto \(x_2\) by projecting out \(x_1\)

\[
\begin{align*}
-2x_1 - x_2 & \geq -4 \quad (1/2) \\
x_1 - x_2 & \geq -1 \quad (1) \\
\frac{-3}{2}x_2 & \geq -3 \\
\text{or} \\
x_2 & \leq 2
\end{align*}
\]
Optimization as Projection

• Optimization is projection onto a single variable.
  – To solve \( \min / \max \{ f(x) \mid x \in S \} \)
    project \( \{(x_0, x) \mid x_0 = f(x), x \in S\} \)
    onto \( x_0 \) to obtain an interval \( x_0^{\text{min}} \leq x_0 \leq x_0^{\text{max}} \)
Optimization as Projection

• Optimization is projection onto a single variable.
  – To solve \( \min / \max \{ f(x) \mid x \in S \} \)
  
  project \( \{(x_0, x) \mid x_0 = f(x), x \in S\} \)
  
  onto \( x_0 \) to obtain an interval \( x_0^{\min} \leq x_0 \leq x_0^{\max} \)

• Linear programming
  – We can in principle solve \( \min / \max \{ cx \mid Ax \geq b \} \)
    with Fourier-Motzkin elimination
    by projecting \( \{(x_0, x) \mid x_0 = cx, Ax \geq b\} \) onto \( x_0 \)
  – But this is extremely inefficient.
  – Use simplex or interior point method instead.
Probability Logic

• Inference in **probability logic** is a polyhedral projection problem
  – Originally stated by George Boole.
  – The **linear programming problem** can be solved, in principle, by Fourier-Motzkin elimination.

• The problem
  – Given a **probability interval** for each of several formulas in propositional logic,
  – Deduce a probability interval for a target formula.
### Example

<table>
<thead>
<tr>
<th>Formula</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.9</td>
</tr>
<tr>
<td>if $x_1$ then $x_2$</td>
<td>0.8</td>
</tr>
<tr>
<td>if $x_2$ then $x_3$</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Deduce probability range for $x_3$

---

Boole (1854)
# Probability Logic

## Example

<table>
<thead>
<tr>
<th>Formula</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.9</td>
</tr>
<tr>
<td>if $x_1$ then $x_2$</td>
<td>0.8</td>
</tr>
<tr>
<td>if $x_2$ then $x_3$</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Deduce probability range for $x_3$

Interpret if-then statements as material conditionals

---

Boole (1854)
## Probability Logic

### Example

<table>
<thead>
<tr>
<th>Formula</th>
<th>Probability</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>$\overline{x}_1 \lor x_2$</td>
<td>0.8</td>
<td>Interpret if-then statements as material conditionals</td>
</tr>
<tr>
<td>$\overline{x}_2 \lor x_3$</td>
<td>0.4</td>
<td></td>
</tr>
</tbody>
</table>

Deduce probability range for $x_3$

---

Boole (1854)
Probability Logic

Example

<table>
<thead>
<tr>
<th>Formula</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.9</td>
</tr>
<tr>
<td>$\overline{x}_1 \lor x_2$</td>
<td>0.8</td>
</tr>
<tr>
<td>$\overline{x}_2 \lor x_3$</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Deduce probability range for $x_3$

Linear programming model

\[
\begin{bmatrix}
01010101 \\
00001111 \\
11110011 \\
11011101 \\
11111111
\end{bmatrix} \begin{bmatrix}
p_{000} \\
p_{001} \\
p_{010} \\
p_{100} \\
p_{111}
\end{bmatrix} = \begin{bmatrix}
\pi_0 \\
0.9 \\
0.8 \\
0.4 \\
1
\end{bmatrix}
\]

$p_{000} = \text{probability that } (x_1, x_2, x_3) = (0, 0, 0)$

Hailperin (1976)
Nilsson (1986)
Probability Logic

Example

<table>
<thead>
<tr>
<th>Formula</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.9</td>
</tr>
<tr>
<td>$\bar{x}_1 \lor x_2$</td>
<td>0.8</td>
</tr>
<tr>
<td>$\bar{x}_2 \lor x_3$</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Deduce probability range for $x_3$

Linear programming model

$$\min/\max \quad \pi_0$$

$$\begin{bmatrix}
01010101 \\
00001111 \\
11110011 \\
11011101 \\
11111111
\end{bmatrix} \begin{bmatrix}
p_{000} \\
p_{001} \\
p_{010} \\
p_{110} \\
p_{111}
\end{bmatrix} = \begin{bmatrix}
\pi_0 \\
0.9 \\
0.8 \\
0.4 \\
1
\end{bmatrix}$$

$p_{000} = \text{probability that } (x_1, x_2, x_3) = (0,0,0)$

Solution: $\pi_0 \in [0.1, 0.4]$

Hailperin (1976)
Nilsson (1986)
Inference as Projection

• Projection can be viewed as the fundamental inference problem.
  – Deduce information that pertains to a desired subset of propositional variables.

• In propositional logic (SAT), this can be achieved by the **resolution** method.
  – CNF analog of Quine’s **consensus** method for DNF.
Inference as Projection

- Project onto propositional variables of interest
  - Suppose we wish to infer from these clauses everything we can about propositions \( x_1, x_2, x_3 \)

\[
\begin{array}{|c|c|}
\hline
x_1 & \lor x_4 \lor x_5 \\
\hline
x_1 & \lor x_4 \lor \overline{x}_5 \\
\hline
x_1 & \lor x_5 \lor x_6 \\
\hline
x_1 & \lor x_5 \lor \overline{x}_6 \\
\hline
x_2 & \lor \overline{x}_5 \lor x_6 \\
\hline
x_2 & \lor \overline{x}_5 \lor \overline{x}_6 \\
\hline
x_3 & \lor \overline{x}_4 \lor x_5 \\
\hline
x_3 & \lor \overline{x}_4 \lor \overline{x}_5 \\
\hline
\end{array}
\]
Inference as Projection

- Project onto propositional variables of interest
  - Suppose we wish to infer from these clauses everything we can about propositions \( x_1, x_2, x_3 \)

We can deduce

\[
\begin{align*}
x_1 & \lor x_2 \\
x_1 & \lor x_3
\end{align*}
\]

This is a projection onto \( x_1, x_2, x_3 \)
Inference as Projection

• Resolution as a projection method
  – Similar to Fourier-Motzkin elimination
    – Actually, identical to Fourier-Motzkin elimination + rounding
  – To project out $x_j$, eliminate it from pairs of clauses:

$$C \lor x_j \quad D \lor \overline{x_j}$$

$$C \lor D$$

  – Then remove all clauses containing $x_j$

Quine (1952,1955)
JH (1992,2012)
Inference as Projection

- Interpretation as Fourier-Motzkin + rounding
  - Project out $x_1$ using resolution:

\[
\begin{align*}
  x_1 \lor x_2 \lor x_3 \\
  \overline{x_1} \lor x_3 \lor x_4 \\
  \overline{x_1} \lor x_3 \lor x_4
\end{align*}
\]
Inference as Projection

• Interpretation as Fourier-Motzkin + rounding
  – Project out $x_1$ using resolution:
    \[
    x_1 \lor x_2 \lor x_3 \\
    \bar{x}_1 \lor x_3 \lor x_4 \\
    \hline
    \]
    \[
    x_2 \lor x_3 \lor x_4
    \]
  – Project out $x_1$ using Fourier-Motzkin + rounding
    \[
    x_1 + x_2 + x_3 \geq 1 \quad (1/2) \\
    -x_1 + x_3 + x_4 \geq 0 \quad (1/2) \\
    x_2 \geq 0 \quad (1/2) \\
    x_4 \geq 0 \quad (1/2)
    \]
    \[
    x_2 + x_3 + x_4 \geq \frac{1}{2}
    \]
    rounds to $x_2 + x_3 + x_4 \geq 1$
    \[
    \]
    since $x_j$s are integer

Williams (1987)
Projection and Cutting Planes

• A resolvent is a special case of a rank 1 Chvátal cut.
  – A general inference method for integer programming.
  – All rank 1 cuts can be obtained by taking nonnegative linear combinations and rounding.
  – We can deduce all valid inequalities by recursive generation of rank 1 cuts.
  – …including inequalities describing the projection onto a given subset of variables.
  – The minimum number of iterations necessary is the Chvátal rank of the constraint set.
  – There is no upper bound on the rank as a function of the number of variables.

Chvátal 1973
Projection Methods

• Generalizations of resolution
  – For cardinality clauses  JH (1988)
  – For 0-1 linear inequalities  JH (1992)
  – For general integer linear inequalities  Williams & JH (2015)
Projection for Integer Programming

Example: solve

\[ \min x_2 \]
\[ 2x_1 + x_2 \geq 13 \quad \text{C1} \]
\[ -5x_1 - 2x_2 \geq -30 \quad \text{C2} \]
\[ -x_1 + x_2 \geq 5 \quad \text{C3} \]
\[ x_1, x_2 \in \mathbb{Z} \]
Projection for Integer Programming

Example: solve

\[
\begin{align*}
\text{min } & \quad x_2 \\
2x_1 + x_2 & \geq 13 \quad \text{C1} \\
-5x_1 - 2x_2 & \geq -30 \quad \text{C2} \\
x_1 + x_2 & \geq 5 \quad \text{C3} \\
x_1, x_2 \in \mathbb{Z}
\end{align*}
\]

To project out \(x_1\), first combine C1 and C2:

\[
\begin{align*}
2x_1 + x_2 & \geq 13 \quad (5) \\
-5x_1 - 2x_2 & \geq -30 \quad (2)
\end{align*}
\]

\[
\frac{5(x_2 - 13) + 2(-2x_2 + 30)}{5(x_2 - 13) + 2(-2x_2 + 30)} \geq 0
\]
Projection for Integer Programming

Example: solve
\[
\begin{align*}
\text{min } x_2 \\
2x_1 + x_2 & \geq 13 \quad \text{C1} \\
-5x_1 - 2x_2 & \geq -30 \quad \text{C2} \\
x_1 + x_2 & \geq 5 \quad \text{C3} \\
x_1, x_2 & \in \mathbb{Z}
\end{align*}
\]

To project out \(x_1\), first combine C1 and C2:
\[
\begin{align*}
2x_1 + x_2 & \geq 13 \quad (5) \\
-5x_1 - 2x_2 & \geq -30 \quad (2) \\
\hline
5(x_2 - 13) + 2(-2x_2 + 30) & \geq 0
\end{align*}
\]

Since 2\textsuperscript{nd} term is even, we can write this as
\[
5(x_2 - 13 - u) + 2(-2x_2 + 30) \geq 0, \quad x_2 - 13 - u \equiv 0 \pmod{2}
\]
where \(u \in \{0, 1\}\). This simplifies to
\[
x_2 \geq 5 + 5u, \quad x_2 \equiv u + 1 \pmod{2}
\]
Projection for Integer Programming

Example: solve

\[
\begin{align*}
\text{min } & \quad x_2 \\
2x_1 + x_2 & \geq 13 \quad \text{C1} \\
-5x_1 - 2x_2 & \geq -30 \quad \text{C2} \\
x_1 + x_2 & \geq 5 \quad \text{C3} \\
x_1, x_2 & \in \mathbb{Z}
\end{align*}
\]

After similarly combining C1 and C3, we get the problem with \( x_1 \) projected out:

\[
\begin{align*}
\text{min } & \quad x_2 \\
x_2 & \geq 5 + 5u, \quad 3x_2 \geq 23 + u \\
x_2 & \equiv u + 1 \pmod{2}, \quad u \in \{0, 1\}
\end{align*}
\]
Projection for Integer Programming

Example: solve

\[ \begin{align*}
\text{min } & x_2 \\
2x_1 + x_2 & \geq 13 \quad \text{C1} \\
-5x_1 - 2x_2 & \geq -30 \quad \text{C2} \\
-x_1 + x_2 & \geq 5 \quad \text{C3} \\
x_1, x_2 & \in \mathbb{Z}
\end{align*} \]

After similarly combining C1 and C3, we get the problem with \( x_1 \) projected out:

\[ \begin{align*}
\text{min } & x_2 \\
x_2 & \geq 5 + 5u, \quad 3x_2 \geq 23 + u \\
x_2 & \equiv u + 1 \pmod{2}, \ u \in \{0, 1\}
\end{align*} \]

This is equivalent to

\[ \begin{align*}
\text{min } & x_2(= 9) \\
x_2 & \geq 5, \quad 3x_2 \geq 23 \\
x_2 & \text{ odd}
\end{align*} \]

\[ \begin{align*}
\text{or } \quad \text{min } & x_2(= 10) \\
x_2 & \geq 10, \quad 3x_2 \geq 24 \\
x_2 & \text{ even}
\end{align*} \]

So optimal value = 9.
Projection for Integer Programming

Example: solve

\begin{align*}
\text{min } & x_2 \\
2x_1 + x_2 & \geq 13 \quad \text{C1} \\
-5x_1 - 2x_2 & \geq -30 \quad \text{C2} \\
-x_1 + x_2 & \geq 5 \quad \text{C3} \\
x_1, x_2 & \in \mathbb{Z}
\end{align*}

Number of iterations to compute a projection is bounded by number of variables projected out, unlike Chvátal cuts, for which number of iterations is unbounded.
Consistency Maintenance as Projection
Consistency as Projection

• Domain consistency
  – Domain of variable $x_j$ contains only values that $x_j$ assumes in some feasible solution.
  – Equivalently, domain of $x_j$ = projection of feasible set onto $x_j$. 
Consistency as Projection

• **Domain consistency**
  – Domain of variable $x_j$ contains only values that $x_j$ takes in some feasible solution.
  – Equivalently, domain of $x_j = \text{projection}$ of feasible set onto $x_j$.

**Example:**

Constraint set

\[
alldiff(x_1, x_2, x_3)
\]

\[
x_1 \in \{a, b\}
\]

\[
x_2 \in \{a, b\}
\]

\[
x_3 \in \{b, c\}
\]
Consistency as Projection

• Domain consistency
  – Domain of variable $x_j$ contains only values that $x_j$ takes in some feasible solution.
  – Equivalently, domain of $x_j$ = projection of feasible set onto $x_j$.

Example:

<table>
<thead>
<tr>
<th>Constraint set</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{alldiff}(x_1, x_2, x_3)$</td>
<td>$(x_1, x_2, x_3)$</td>
</tr>
<tr>
<td>$x_1 \in {a, b}$</td>
<td>$(a, b, c)$</td>
</tr>
<tr>
<td>$x_2 \in {a, b}$</td>
<td>$(b, a, c)$</td>
</tr>
<tr>
<td>$x_3 \in {b, c}$</td>
<td></td>
</tr>
</tbody>
</table>
Consistency as Projection

• Domain consistency
  – Domain of variable $x_j$ contains only values that $x_j$ takes in some feasible solution.
  – Equivalently, domain of $x_j = \text{projection}$ of feasible set onto $x_j$.

Example:

<table>
<thead>
<tr>
<th>Constraint set</th>
<th>Solutions</th>
<th>Projection onto $x_1$</th>
<th>Projection onto $x_2$</th>
<th>Projection onto $x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{alldiff}(x_1, x_2, x_3)$</td>
<td>$(x_1, x_2, x_3)$</td>
<td>$x_1 \in {a, b}$</td>
<td>$x_2 \in {a, b}$</td>
<td>$x_3 \in {c}$</td>
</tr>
<tr>
<td>$x_1 \in {a, b}$</td>
<td>$(a, b, c)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_2 \in {a, b}$</td>
<td>$(b, a, c)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3 \in {b, c}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consistency as Projection

- **Domain consistency**
  - Domain of variable $x_j$ contains only values that $x_j$ takes in some feasible solution.
  - Equivalently, domain of $x_j = \text{projection}$ of feasible set onto $x_j$.

**Example:**

<table>
<thead>
<tr>
<th>Constraint set</th>
<th>Solutions</th>
<th>Projection onto $x_1$</th>
<th>Projection onto $x_2$</th>
<th>Projection onto $x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{alldiff}(x_1, x_2, x_3)$</td>
<td>$(x_1, x_2, x_3)$</td>
<td>$x_1 \in {a, b}$</td>
<td>$x_2 \in {a, b}$</td>
<td>$x_3 \in {c}$</td>
</tr>
<tr>
<td>$x_1 \in {a, b}$</td>
<td>$(a, b, c)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_2 \in {a, b}$</td>
<td>$(b, a, c)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3 \in {b, c}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This achieves domain consistency.
Consistency as Projection

- **k-consistency**
  - Can be defined:
    - A constraint set $S$ is $k$-consistent if:
      - for every $J \subseteq \{1, \ldots, n\}$ with $|J| = k - 1$,
      - every assignment $x_J = v_J \in D_J$ for which $(x_J, x_j)$ does not violate $S$,
      - and every variable $x_j \notin x_J$, there is an assignment $x_j = v_j \in D_j$ for which $(x_J, x_j) = (v_J, v_j)$ does not violate $S$. 

\[ x_J = (x_j \mid j \in J) \]
Consistency as Projection

- **k-consistency**
  - Can be defined:
    - A constraint set $S$ is $k$-consistent if:
      - for every $J \subseteq \{1, \ldots, n\}$ with $|J| = k - 1$,
      - every assignment $x_J = v_J \in D_J$ for which $(x_J, x_j)$ does not violate $S$,
      - and every variable $x_j \notin x_J$,
        there is an assignment $x_j = v_j \in D_j$ for which $(x_J, x_j) = (v_J, v_j)$ does not violate $S$.
    - To achieve $k$-consistency:
      - Project the constraints containing each set of $k$ variables onto subsets of $k - 1$ variables.

$x_J = (x_j | j \in J)$
Consistency as Projection

• Consistency and backtracking:
  
  – Strong $k$-consistency for entire constraint set avoids backtracking…
    
    – if the primal graph has width $< k$ with respect to branching order.

  – No point in achieving strong $k$-consistency for individual constraints if we propagate through domain store.
    
    – Domain consistency has same effect.

Freuder (1982)
**J-Consistency**

- A type of consistency more directly related to projection.
  
  - Constraint set $S$ is **J-consistent** if it contains the projection of $S$ onto $x_j$.
  
  - $S$ is domain consistent if it is $\{j\}$-consistent for each $j$.

\[
x_j = (x_j \mid j \in J)
\]
\textbf{J-Consistency}

- \textit{J}-consistency and backtracking:
  - If we project a constraint onto $x_1, x_2, \ldots, x_k$, the constraint will not cause backtracking as we branch on the remaining variables.
    - A natural strategy is to project out $x_n, x_{n-1}, \ldots$ until computational burden is excessive.
**J-Consistency**

- **J-consistency and backtracking:**
  - If we project a constraint onto \(x_1, x_2, \ldots, x_k\), the constraint will not cause backtracking as we branch on the remaining variables.
    - A natural strategy is to project out \(x_n, x_{n-1}, \ldots\) until computational burden is excessive.
  - No point in achieving J-consistency for individual constraints if we propagate through a domain store.
    - However, J-consistency can be useful if we propagate through a richer data structure
    - …such as decision diagrams
    - …which can be more effective as a propagation medium.

JH & Hadžić (2006,2007)
Andersen, Hadžić, JH, Tiedemann (2007)
Bergman, Ciré, van Hoeve, JH (2014)
Propagating $J$-Consistency

Example:

among\((x_1, x_2),\{c, d\}, 1, 2\)  
\(x_1 = c \Rightarrow x_2 = d\)  
alldiff\((x_1, x_2, x_3, x_4)\)  
\(x_1, x_2 \in \{a, b, c, d\}\)  
\(x_3 \in \{a, b\}\)  
\(x_4 \in \{c, d\}\)

Already domain consistent for individual constraints.

If we branch on \(x_1\) first, must consider all 4 branches \(x_1 = a, b, c, d\)
Propagating $J$-Consistency

Example:

Suppose we propagate through a relaxed decision diagram of width 2 for these constraints

\[ \text{among}\left( (x_1, x_2), \{c, d\}, 1, 2 \right) \]
\[ (x_1 = c) \Rightarrow (x_2 = d) \]
\[ \text{alldiff}\left( x_1, x_2, x_3, x_4 \right) \]
\[ x_1, x_2 \in \{a, b, c, d\} \]
\[ x_3 \in \{a, b\} \]
\[ x_4 \in \{c, d\} \]

52 paths from top to bottom represent assignments to $x_1, x_2, x_3, x_4$
36 of these are the feasible assignments.
Propagating J-Consistency

Example:

\[ \text{among}((x_1, x_2), \{c, d\}, 1, 2) \]
\[ (x_1 = c) \implies (x_2 = d) \]
\[ \text{alldiff} (x_1, x_2, x_3, x_4) \]
\[ x_1, x_2 \in \{a, b, c, d\} \]
\[ x_3 \in \{a, b\} \]
\[ x_4 \in \{c, d\} \]

Suppose we propagate through a relaxed decision diagram of width 2 for these constraints

52 paths from top to bottom represent assignments to \(x_1, x_2, x_3, x_4\)
36 of these are the feasible assignments.

Projection of alldiff onto \(x_1, x_2\) is

\[ \text{alldiff} (x_1, x_2) \]
\[ \text{atmost}((x_1, x_2), \{a, b\}, 1) \]
\[ \text{atmost}((x_1, x_2), \{c, d\}, 1) \]
Let’s propagate the $2^{nd}$ atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in \{c,d\}.

For each arc, indicate length of shortest path from top to that arc.

Projection of alldiff onto $x_1, x_2$ is

$$\text{alldiff}(x_1, x_2)$$

$$\text{atmost}((x_1, x_2), \{a, b\}, 1)$$

$$\text{atmost}((x_1, x_2), \{c, d\}, 1)$$
Let’s propagate the 2\textsuperscript{nd} atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in \{c,d\}.

For each arc, indicate length of shortest path from top to that arc.

Projection of \textit{alldiff} onto $x_1$, $x_2$ is

\[
alldiff(x_1, x_2)
\atmost((x_1, x_2), \{a, b\}, 1)
\atmost((x_1, x_2), \{c, d\}, 1)
\]
Propagating \( J \)-Consistency

Let’s propagate the 2\textsuperscript{nd} atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in \{c,d\}.

For each arc, indicate length of shortest path from top to that arc.

Remove arcs with label > 1

Projection of alldiff onto \( x_1, x_2 \) is

\[
\text{alldiff} \left( x_1, x_2 \right) \ \ \text{atmost} \left( \left( x_1, x_2 \right), \{a, b\}, 1 \right) \ \ \text{atmost} \left( \left( x_1, x_2 \right), \{c, d\}, 1 \right)
\]
Propagating $J$-Consistency

Let’s propagate the 2\textsuperscript{nd} atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in \{c,d\}.

For each arc, indicate length of shortest path from top to that arc.

Remove arcs with label > 1

Projection of alldiff onto $x_1$, $x_2$ is

\[
\text{alldiff}(x_1, x_2) \quad \text{atmost}((x_1, x_2), \{a, b\}, 1) \\
\text{atmost}((x_1, x_2), \{c, d\}, 1)
\]
Propagating $J$-Consistency

Let’s propagate the 2nd atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in \{c,d\}.

For each arc, indicate length of shortest path from top to that arc.

Projection of alldiff onto $x_1$, $x_2$ is

\[
\text{alldiff}(x_1, x_2) \atmost((x_1, x_2), \{a, b\}, 1) \atmost((x_1, x_2), \{c, d\}, 1)
\]

Remove arcs with label $> 1$

Clean up.
Propagating $J$-Consistency

Let's propagate the $2^{nd}$ atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in \{c,d\}.

For each arc, indicate length of shortest path from top to that arc.

Remove arcs with label $> 1$

Clean up.

Projection of alldiff onto $x_1$, $x_2$ is

\[
alldiff(x_1, x_2) \quad \text{atmost}((x_1, x_2), \{a, b\}, 1) \quad \text{atmost}((x_1, x_2), \{c, d\}, 1)
\]
Propagating $J$-Consistency

Let's propagate the 2$^{\text{nd}}$ atmost constraint in the projected alldiff through the relaxed decision diagram.

We need only branch on $a,b,d$ rather than $a,b,c,d$

Remove arcs with label $> 1$

Clean up.

Projection of alldiff onto $x_1, x_2$ is

\[
\text{alldiff}(x_1, x_2)
\]

\[
\text{atmost}((x_1, x_2), \{a, b\}, 1)
\]

\[
\text{atmost}((x_1, x_2), \{c, d\}, 1)
\]
Achieving $J$-consistency

<table>
<thead>
<tr>
<th>Constraint</th>
<th>How hard to project?</th>
</tr>
</thead>
<tbody>
<tr>
<td>among</td>
<td>Easy and fast.</td>
</tr>
<tr>
<td>sequence</td>
<td>More complicated but fast. Since polyhedron is integral, can write a formula based on <strong>Fourier-Motzkin</strong></td>
</tr>
<tr>
<td>regular</td>
<td>Easy and basically same labor as domain consistency.</td>
</tr>
<tr>
<td>alldiff</td>
<td>Quite complicated but practical for small domains.</td>
</tr>
</tbody>
</table>
Projection Using Benders Decomposition and Its Generalizations
Logic-Based Benders

- **Logic-based Benders decomposition** is a generalization of classical Benders decomposition.

  - Solves a problem of the form
    \[
    \min f(x, y) \\
    (x, y) \in S \\
    x \in D
    \]

Logic-Based Benders

- Decompose problem into master and subproblem.
  - Subproblem is obtained by fixing $x$ to solution value in master problem.

Master problem

\[
\min z \\
z \geq g_k(x) \quad \text{(Benders cuts)} \\
x \in D
\]

Minimize cost $z$ subject to bounds given by Benders cuts, obtained from values of $x$ attempted in previous iterations $k$.

Subproblem

\[
\min f(\bar{x}, y) \\
(\bar{x}, y) \in S
\]

Obtain proof of optimality (solution of inference dual). Use same proof to deduce cost bounds for other assignments, yielding Benders cut.
Logic-Based Benders

- Iterate until master problem value equals best subproblem value so far.
  - This yields optimal solution.

Master problem

\[ \min z \]
\[ z \geq g_k(x) \quad \text{(Benders cuts)} \]
\[ x \in D \]

Minimize cost \( z \) subject to bounds given by Benders cuts, obtained from values of \( x \) attempted in previous iterations \( k \).

Subproblem

\[ \min f(\bar{x},y) \]
\[ (\bar{x},y) \in S \]

Obtain proof of optimality (solution of \textit{inference dual}). Use same proof to deduce cost bounds for other assignments, yielding Benders cut.
Logic-Based Benders

- The Benders cuts define the projection of the feasible set onto $(z,x)$.
  - If all possible cuts are generated.

Master problem

\[
\begin{align*}
\text{min } z \\
z &\geq g_k(x) \quad \text{(Benders cuts)} \\
x &\in D
\end{align*}
\]

Minimize cost $z$ subject to bounds given by Benders cuts, obtained from values of $x$ attempted in previous iterations $k$.

Subproblem

\[
\begin{align*}
\text{min } f(\bar{x}, y) \\
(\bar{x}, y) &\in S
\end{align*}
\]

Obtain proof of optimality (solution of inference dual). Use same proof to deduce cost bounds for other assignments, yielding Benders cut.

Trial value $\bar{x}$ that solves master

Benders cut $z \geq g_k(x)$
Logic-Based Benders

- **Fundamental concept:** inference duality

**Primal problem:**
- Optimization
  
  \[
  \min f(x) \\
  x \in S
  \]
  
  Find **best** feasible solution by searching over values of \(x\).

**Dual problem:**
- Inference
  
  \[
  \max \nu \\
  x \in S \quad \overset{P}{\Rightarrow} \quad f(x) \geq \nu \\
  P \in \mathcal{P}
  \]
  
  Find a proof of optimal value \(\nu^*\) by searching over proofs \(P\).
Logic-Based Benders

• Popular optimization duals are special cases of the inference dual.
  – Result from different choices of inference method.
  – For example....
    – Linear programming dual (gives classical Benders cuts)
    – Lagrangean dual
    – Surrogate dual
    – Subadditive dual
Classical Benders

- **Linear programming dual** results in classical Benders method.
  - The problem is

\[
\begin{align*}
\min & \quad cx + dy \\
\text{subject to} & \quad Ax + By \geq b
\end{align*}
\]

**Master problem**

- Minimize cost \(z\) subject to bounds given by Benders cuts, obtained from values of \(x\) attempted in previous iterations \(k\).

**Subproblem**

- Trial value \(\bar{x}\) that solves master

\[
\begin{align*}
\min & \quad c\bar{x} + dy \\
By & \geq b - A\bar{x}
\end{align*}
\]

- Obtain proof of optimality by solving **LP dual**:

\[
\begin{align*}
\max & \quad u(b - A\bar{x}) \\
UB & \leq d, \quad u \geq 0
\end{align*}
\]

Benders (1962)
Application to Planning & Scheduling

- Assign tasks in master, schedule in subproblem.
  - Combine **mixed integer programming** and **constraint programming**

Master problem

- Assign tasks to resources to minimize cost.
- Solve by **mixed integer programming**.

Subproblem

- Schedule jobs on each machine, subject to time windows.
- **Constraint programming** obtains proof of optimality (dual solution).
- Use **same proof** to deduce cost for some other assignments, yielding Benders cut.

\[ z \geq g_k(x) \]
Application to Planning & Scheduling

• Objective function
  – Cost is based on task assignment only.

\[
\text{cost} = \sum_{ij} c_{ij} x_{ij}, \quad x_{ij} = 1 \text{ if task } j \text{ assigned to resource } i
\]

  – So cost appears only in the master problem.
  – Scheduling subproblem is a feasibility problem.
Application to Planning & Scheduling

• **Objective function**
  – Cost is based on **task assignment only**.
    $$\text{cost} = \sum_{ij} c_{ij} x_{ij}, \quad x_{ij} = 1 \text{ if task } j \text{ assigned to resource } i$$
    – So cost appears only in the **master problem**.
    – Scheduling subproblem is a **feasibility problem**.

• **Benders cuts**
  – They have the form
    $$\sum_{j \in J_i} (1 - x_{ij}) \geq 1, \quad \text{all } i$$
    – where $J_i$ is a set of tasks that create infeasibility when assigned to resource $i$. 
Application to Planning & Scheduling

- Resulting Benders decomposition:

Master problem

\[
\min z \\
z = \sum_{ij} c_{ij} x_{ij} \\
\text{Benders cuts}
\]

Subproblem

Trial assignment \( \bar{x} \)

Benders cuts

\[
\sum_{j \in d_i} (1 - x_{ij}) \geq 1, \\
\text{for infeasible resources } i
\]

Schedule jobs on each resource.

**Constraint programming** may obtain proof of infeasibility on some resources (dual solution).

Use **same proof** to deduce infeasibility for some other assignments, yielding Benders cut.
Performance profile

50 instances
Application to Probability Logic

Exponentially many variables in LP model. What to do?

<table>
<thead>
<tr>
<th>Formula</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.9</td>
</tr>
<tr>
<td>$\overline{x}_1 \lor x_2$</td>
<td>0.8</td>
</tr>
<tr>
<td>$\overline{x}_2 \lor x_3$</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Deduce probability range for $x_3$

Linear programming model

$$\min/\max \quad \pi_0$$

\[
\begin{bmatrix}
01010101 \\
00001111 \\
11110011 \\
11011101 \\
11111111
\end{bmatrix} \begin{bmatrix}
p_{000} \\
p_{001} \\
p_{010} \\
p_{110} \\
p_{111}
\end{bmatrix} = \begin{bmatrix}
\pi_0 \\
0.9 \\
0.8 \\
0.4 \\
1
\end{bmatrix}
\]

$\rho_{000} =$ probability that $(x_1,x_2,x_3) = (0,0,0)$
Application to Probability Logic

Exponentially many variables in LP model. What to do? Apply classical Benders to linear programming dual!

<table>
<thead>
<tr>
<th>Formula</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.9</td>
</tr>
<tr>
<td>$\overline{x}_1 \lor x_2$</td>
<td>0.8</td>
</tr>
<tr>
<td>$\overline{x}_2 \lor x_3$</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Deduce probability range for $x_3$

### Linear programming model

$$\begin{array}{ccc}
\text{min/ max} & \pi_0 \\
\begin{bmatrix}
01010101 \\
00001111 \\
11100011 \\
11011101 \\
11111111 \\
\end{bmatrix} & \begin{bmatrix}
\rho_{000} \\
\rho_{001} \\
\rho_{010} \\
\vdots \\
\rho_{111} \\
\end{bmatrix} & \begin{bmatrix}
\pi_0 \\
0.9 \\
0.8 \\
0.4 \\
1 \\
\end{bmatrix}
\end{array}$$

$\rho_{000} = \text{probability that } (x_1, x_2, x_3) = (0,0,0)$
### Application to Probability Logic

**Exponentially many** variables in LP model. What to do?

*Apply classical Benders to **linear programming dual**!*

This results in a **column generation** method that introduces variables into LP only as needed to find optimum.

#### Linear programming model

\[
\begin{align*}
\text{min/ max } & \quad \pi_0 \\
\begin{bmatrix}
01010101 \\
00001111 \\
11110011 \\
11011101 \\
11111111 \\
\end{bmatrix} & \begin{bmatrix}
p_{000} \\
p_{001} \\
p_{010} \\
p_{101} \\
p_{111} \\
\end{bmatrix} = \begin{bmatrix}
\pi_0 \\
0.9 \\
0.8 \\
0.4 \\
1 \\
\end{bmatrix}
\end{align*}
\]

\[\rho_{000} = \text{probability that } (x_1, x_2, x_3) = (0, 0, 0)\]
Inference as Projection

- Recall that logical inference is a projection problem.
  - We wish to infer from these clauses everything we can about propositions \( x_1, x_2, x_3 \)

We can deduce

\[
\begin{align*}
x_1 & \lor x_2 \\
x_1 & \lor x_3
\end{align*}
\]

This is a projection onto \( x_1, x_2, x_3 \)
Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1, x_2, x_3$
Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1, x_2, x_3$

Current Master problem

\[ x_1 \lor x_2 \]

solution of master $(x_1, x_2, x_3) = (0, 1, 0)$

Resulting subproblem

\[
\begin{align*}
x_4 & \lor x_5 \\
x_4 & \lor \bar{x}_5 \\
x_5 & \lor x_6 \\
x_5 & \lor \bar{x}_6 \\
\bar{x}_4 & \lor x_5 \\
\bar{x}_4 & \lor \bar{x}_5
\end{align*}
\]
Inference as Projection

• Benders decomposition computes the projection!
  – Benders cuts describe projection onto $x_1, x_2, x_3$

Current
Master problem

\[ x_1 \lor x_2 \]

solution of master
\[ (x_1, x_2, x_3) = (0,1,0) \]

Resulting
subproblem

\[ x_4 \lor x_5 \]
\[ x_4 \lor \bar{x}_5 \]
\[ x_5 \lor x_6 \]
\[ x_5 \lor \bar{x}_6 \]
\[ \bar{x}_4 \lor x_5 \]
\[ \bar{x}_4 \lor \bar{x}_5 \]

Subproblem is infeasible.
\[ (x_1, x_3) = (0,0) \]
creates infeasibility
Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto \( x_1, x_2, x_3 \)

Current Master problem

\[
\begin{align*}
  x_1 \lor x_2 & \\
  x_1 \lor x_3 &
\end{align*}
\]

solution of master \((x_1, x_2, x_3) = (0, 1, 0)\)

Benders cut (nogood)

Subproblem is infeasible. \((x_1, x_3) = (0, 0)\) creates infeasibility

Resulting subproblem

\[
\begin{align*}
  x_4 \lor x_5 & \\
  x_4 \lor \overline{x}_5 & \\
  x_5 \lor x_6 & \\
  x_5 \lor \overline{x}_6 & \\
  \overline{x}_4 \lor x_5 & \\
  \overline{x}_4 \lor \overline{x}_5 &
\end{align*}
\]
Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1, x_2, x_3$
Inference as Projection

• Benders decomposition computes the projection!
  – Benders cuts describe projection onto $x_1, x_2, x_3$

Current Master problem

$\begin{array}{l}
  x_1 \lor x_2 \\
  x_1 \lor x_3
\end{array}$

solution of master $(x_1,x_2,x_3) = (0,1,1)$

Subproblem is feasible

Resulting subproblem

$\begin{array}{l}
  x_4 \lor \overline{x}_5 \\
  x_4 \lor \overline{x}_5 \\
  \overline{x}_5 \lor x_6 \\
  x_5 \lor \overline{x}_6
\end{array}$
Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1$, $x_2$, $x_3$

Current Master problem

```
X_1 \lor X_2
X_1 \lor X_3
X_1 \lor \overline{X_2} \lor \overline{X_3}
```

solution of master $(x_1, x_2, x_3) = (0, 1, 1)$

Enumerative Benders cut

Subproblem is feasible

Resulting subproblem

```
x_4 \lor x_5
x_4 \lor \overline{x_5}
\overline{x_5} \lor x_6
\overline{x_5} \lor \overline{x_6}
```
Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1, x_2, x_3$

Current Master problem

\[
\begin{align*}
X_1 \lor X_2 \\
X_1 \lor X_3 \\
X_1 \lor \overline{X}_2 \lor \overline{X}_3
\end{align*}
\]

solution of master $(x_1, x_2, x_3) = (0, 1, 1)$

Enumerative Benders cut

Resulting subproblem

\[
\begin{align*}
x_4 \lor x_5 \\
x_4 \lor \overline{x}_5 \\
x_5 \lor x_6 \\
x_5 \lor \overline{x}_6
\end{align*}
\]

Continue until master is infeasible.

Black Benders cuts describe projection.

JH (2000, 2012)
Inference as Projection

• Benders cuts = **conflict clauses** in a SAT algorithm!
  – Branch on $x_1$, $x_2$, $x_3$ first.
Inference as Projection

• Benders cuts = **conflict clauses** in a SAT algorithm!
  – Branch on $x_1, x_2, x_3$ first.
Inference as Projection

- Benders cuts = **conflict clauses** in a SAT algorithm!
  - Branch on $x_1$, $x_2$, $x_3$ first.
Inference as Projection

• Benders cuts = **conflict clauses** in a SAT algorithm!
  – Branch on $x_1$, $x_2$, $x_3$ first.

Conflict clauses containing $x_1$, $x_2$, $x_3$ describe projection
• Logic-based Benders can speed up search in several domains.
  – Several orders of magnitude relative to state of the art.
• Some applications:
  – Circuit verification
  – Chemical batch processing (BASF, etc.)
  – Steel production scheduling
  – Auto assembly line management (Peugeot-Citroën)
  – Automated guided vehicles in flexible manufacturing
  – Allocation and scheduling of multicore processors (IBM, Toshiba, Sony)
  – Facility location-allocation
  – Stochastic facility location and fleet management
  – Capacity and distance-constrained plant location
Logic-Based Benders

• Some applications…
  – Transportation network design
  – Traffic diversion around blocked routes
  – Worker assignment in a queuing environment
  – Single- and multiple-machine allocation and scheduling
  – Permutation flow shop scheduling with time lags
  – Resource-constrained scheduling
  – Wireless local area network design
  – Service restoration in a network
  – Optimal control of dynamical systems
  – Sports scheduling
First-Order Logic

- **Partial instantiation methods** for first-order logic can be viewed as Benders methods
  - The **master problem** is a SAT problem for the current formula $F$,
  - The solution of the master finds a **satisfier mapping** that makes one literal of each clause of $F$ (the satisfier of the clause) true.

JH, Rago, Chandru, Shrivastava (2002)
Partial instantiation methods for first-order logic can be viewed as Benders methods

- The master problem is a SAT problem for the current formula $F$,
  - The solution of the master finds a satisfier mapping that makes one literal of each clause of $F$ (the satisfier of the clause) true.
- The subproblem checks whether a satisfier mapping is blocked.
  - This means atoms assigned true and false can be unified.

JH, Rago, Chandru, Shrivastava (2002)
First-Order Logic

• **Partial instantiation methods** for first-order logic can be viewed as Benders methods
  
  – The **master problem** is a SAT problem for the current formula $F$,
    
    – The solution of the master finds a **satisfier mapping** that makes one literal of each clause of $F$ (the satisfier of the clause) true.
  
  – The subproblem checks whether a satisfier mapping is **blocked**.
    
    – This means atoms assigned true and false can be **unified**.
  
  – In case of blockage, more complete instantiations of the blocked clauses are added to $F$ as **Benders cuts**.

JH, Rago, Chandru, Shrivastava (2002)
Resulting Benders decomposition:

Master problem

Current partially instantiated formula \( F \).

Solve SAT problem for a satisfier mapping.

Subproblem

Check if the satisfier mapping is \textbf{blocked} by unifying atoms that receive different truth values.

The \textbf{dual} solution is the most general unifier.

Use \textbf{same unifier} to create \textbf{Benders cuts}: fuller instantiations of the relevant clauses.
Consider the formula

\[ F = \forall x C_1 \land \forall y C_2 \]

where

\[ C_1 = P(a, x) \lor Q(a) \lor \neg R(x) \quad 
C_2 = \neg Q(y) \lor \neg P(y, b) \]
First-Order Logic

Consider the formula

$$F = \forall x C_1 \land \forall y C_2$$

where

$$C_1 = P(a, x) \lor Q(a) \lor \neg R(x)$$

$$C_2 = \neg Q(y) \lor \neg P(y, b)$$

Solution of master problem yields satisfiers shown.
First-Order Logic

Consider the formula

\[ F = \forall x C_1 \land \forall y C_2 \]

where

- \( C_1 = P(a, x) \lor Q(a) \lor \neg R(x) \)
- \( C_2 = \neg Q(y) \lor \neg P(y, b) \)

Solution of master problem yields satisfiers shown.

The satisfier mapping is **blocked** because the atoms \( P(a, x) \) and \( P(y, b) \) can be unified.
First-Order Logic

Consider the formula \( F = \forall x C_1 \land \forall y C_2 \)

True

\[
\begin{align*}
C_1 &= P(a, x) \lor Q(a) \lor \neg R(x) \\
C_2 &= \neg Q(y) \lor \neg P(y, b)
\end{align*}
\]

False

The satisfier mapping is blocked because the atoms \( P(a, x) \) and \( P(y, b) \) can be unified.

Generate **Benders cuts** by applying the most general unifier of the atoms to the clauses containing them, and adding the result to \( F \).

Now, \( F = \forall x C_1 \land \forall y C_2 \land C_3 \land C_4 \)

where \( C_3 = P(a, b) \lor Q(a) \lor \neg R(b) \) \hspace{1cm} \( C_4 = \neg Q(y) \lor \neg P(y, b) \)
Consider the formula \( F = \forall x C_1 \land \forall y C_2 \) where

\[
C_1 = P(a, x) \lor Q(a) \lor \neg R(x) \quad \quad C_2 = \neg Q(y) \lor \neg P(y, b)
\]

Solution of master problem yields satisfiers shown.

The satisfier mapping is **blocked** because the atoms \( P(a, x) \) and \( P(y, b) \) can be unified.

Generate **Benders cuts** by applying the most general unifier of the atoms to the clauses containing them, and adding the result to \( F \). Now,

\[
F = \forall x C_1 \land \forall y C_2 \land C_3 \land C_4
\]

where

\[
C_3 = P(a, b) \lor Q(a) \lor \neg R(b) \quad \quad C_4 = \neg Q(y) \lor \neg P(y, b)
\]

Solution of the new master problem yields a satisfier mapping that is **not blocked** in the subproblem, and the procedure terminates with satisfiability.
First-Order Logic

• We can accommodate full first-order logic with functions
  – If we replace blocked with $M$-blocked
    – Meaning that the satisfier mapping is blocked within a nesting depth of $M$.
  – The procedure always terminates if $F$ is unsatisfiable.
    – It may not terminate if $F$ is satisfiable, since first-order logic is semidecidable.
    – The master problem has infinitely many variables, because the Herbrand base is infinite.
THE END IS NEAR HERE