MIP Modeling with Metaconstraints and Semantic Typing

André Ciré, John Hooker
Carnegie Mellon University
Tallys Yunes
University of Miami

MIP Workshop, July 2013
Two Perspectives on Optimization

• Reduce problems to **standard form** using atomistic constraints
  • Lose structure, but use highly engineered solvers
  • Current orthodoxy for MIP, SAT communities

• Solver must rely on specific **problem structure**
  • Use special purpose solver, or…
  • Convey structure to general solver with **global constraints**.
  • Current practice for CP solvers
In reality…

- You can’t solve NP-hard problems without exploiting special structure.
In reality…

• You can’t solve NP-hard problems without exploiting special structure.

• For SAT solvers:
  • Careful reduction of problem to SAT form
  • This has become a minor industry
In reality…

• You can’t solve NP-hard problems without exploiting special structure.

• For SAT solvers:
  • Careful reduction of problem to SAT form
  • This has become a minor industry

• For MIP solvers:
  • Careful choice of variables for tight formulation
  • Addition of valid inequalities
  • SOS1, SOS2, symmetry-breaking constraints, etc.
  • Solver parameters (e.g., which cuts?)
How to take advantage of structure

• Use a general solver that exploits structure directly
  • Such as a CP or integrated solver
  • ILOG CP Optimizer, CHIP, Gecode, Google CP Solver, ECLiPSe, G12, SIMPL, Xpress-Mosel

• Convey structure to MIP (or SAT) solver
  • Formulate problem with global constraints or \textit{metaconstraints} to reveal structure
  • Automatically convert these to optimal MIP formulation
Conveying structure to MIP

• Given the advanced state of MIP technology…
  • Perhaps a logical next step is to convey problem structure to the solver.
Conveying structure to MIP

• Given the advanced state of MIP technology…
  • Perhaps a logical next step is to convey problem structure to the solver.

• Advantages of using metaconstraints
  • Better MIP formulation, tighter LP relaxation
  • Opportunity to enhance solver with domain filtering, constraint propagation.
Conveying structure to MIP

• Given the advanced state of MIP solvers…
  • Perhaps a logical next step is to convey problem structure to the solver.

• Advantages of using metaconstraints
  • Better MIP formulation, tighter LP relaxation
  • Opportunity to enhance solver with domain filtering, constraint propagation.

• However, metaconstraints pose a fundamental problem of variable management…
Variable management problem

• MIP formulation typically introduces **new variables**
  - Different metaconstraints may introduce variables that are functionally **the same variable**
  - …or related in some other way.
  - Recognizing these relationships is essential to obtaining a good model (in particular, a tight relaxation)
  - How can the solver “understand” what is going on in the model?
Variable management problem

• MIP formulation typically introduces **new variables**
  • Different metaconstraints may introduce variables that are functionally **the same variable**
  • …or related in some other way.

• Recognizing these relationships is essential to obtaining a good model (in particular, a tight relaxation)

• How can the solver “understand” what is going on in the model?

• Proposal: Model with **semantic typing of variables**.
Metaconstraints and semantic typing

- **Metaconstraints** convey problem structure
  - Each represents a structured collection of more elementary constraints.

- **Semantic typing** assigns a meaning to each variable.
  - Allows solver to deduce relationships among variables.
  - Good modeling practice in general.
How variables are introduced

• The MIP solver may reformulate a constraint containing **general integer variable** $x_i$ in terms of 0-1 variables $y_{ij}$, where

$$x_i = \sum_j jy_{ij}$$

• $y_{ij}$s may be **equivalent to other variables that appear** in the model or MIP formulations of other constraints.
How variables are introduced

• The solver may reformulate a disjunction of linear systems

\[ \bigcup_{k} A_k x \geq b^k \]

using a convex hull (or big-M) formulation:

\[ A_k x^k \geq b^k y_k, \text{ all } k \]
\[ x = \sum_k x^k, \quad \sum_k y_k = 1 \]
\[ y_k \in \{0,1\}, \text{ all } k \]

• Other constraints may be based on same set of alternatives, and corresponding auxiliary variables (\(y_k\) etc.) should be equated.
How variables are introduced

• A nonlinear or global solver may use **McCormick factorization** to replace nonlinear subexpressions with auxiliary variables
  
  • … to obtain a linear relaxation.
How variables are introduced

• A nonlinear or global solver may use **McCormick factorization** to replace nonlinear subexpressions with auxiliary variables

• … to obtain a linear relaxation.

• For example, bilinear term $xy$ can be linearized by replacing it with new variable $z$ and constraints

  $L_y x + L_x y - L_x L_y \leq z \leq L_y x + U_x y - L_x U_y$

  $U_y x + U_x y - U_x U_y \leq z \leq U_y x + L_x y - U_x L_y$

  where $x \in [L_x, U_x]$, $y \in [L_y, U_y]$

• Factorization of different constraints may create variables for identical subexpressions.

• They should be identified to get a tight relaxation.
How variables are introduced

• The MIP solver may reformulate global constraints from CP by introducing variables that have the same meaning.
How variables are introduced

• The MIP solver may reformulate global constraints from CP by introducing variables that have the same meaning.

• For example, sequence constraint limits how many jobs of a given type can occur in given time interval:

\[
\text{sequence}(x), \quad x_i = \text{job in position } i
\]

and cardinality constraint limits how many times a given job appears

\[
\text{cardinality}(x), \quad x_j = \text{job in position } j
\]

Both may introduce variables

\[
y_{ij} = 1 \text{ when job } j \text{ occurs in position } i
\]

that should be identified.
How variables are introduced

• Popular global constraints include:

  all-different  lex greater
  among        nvalues
  cardinality  path
  circuit      range
  clique       regular
  cumulative   roots
  cutset       same
  cycle        sort
  diffn        spread
  element      sum
  flow         symmetric alldiff
How variables are introduced

- The solver may introduce equivalent variables while interpreting metaconstraints designed for classical MIP modeling situations:
  - Fixed-charge network flow
  - Facility location
  - Lot sizing
  - Job shop scheduling
  - Assignment (3-dim, quadratic, etc.)
  - Piecewise linear
How variables are introduced

• A model may include two formulations of the problem that use related variables.
  • Common in CP, because it strengthens propagation.
How variables are introduced

• A model may include **two formulations** of the problem that use related variables.
  
  • Common in CP, because it strengthens **propagation**.
  
  • For example,

\[
x_i = \text{job assigned to worker } i
\]

\[
y_j = \text{worker assigned to job } j
\]

• Solver should generate **channeling constraints** to relate the variables to each other:

\[
j = x_y, \quad i = y_x
\]
Motivating example

• Allocate 10 advertising spots to 5 products

\[ x_i = \text{how many spots allocated to product } i \]

\[ y_{ij} = 1 \text{ if } j \text{ spots allocated to product } i \]
Motivating example

• Allocate 10 advertising spots to 5 products

\[ x_i = \text{how many spots allocated to product } i \]

\[ y_{ij} = 1 \text{ if } j \text{ spots allocated to product } i \]

\[ \leq 4 \text{ spots per product} \]
Motivating example

- Allocate 10 advertising spots to 5 products

\[ x_i = \text{how many spots allocated to product } i \]
\[ y_{ij} = \text{1 if } j \text{ spots allocated to product } i \]

\[ \leq 4 \text{ spots per product} \]
\[ \text{Advertise } \leq 3 \text{ products} \]
Motivating example

• Allocate 10 advertising spots to 5 products

\(x_i = \text{how many spots allocated to product } i\)

\(y_{ij} = 1 \text{ if } j \text{ spots allocated to product } i\)

\(\leq 4 \text{ spots per product}\)

Advertise \(\leq 3 \text{ products}\)

\(\geq 4 \text{ spots for at least one product}\)
Motivating example

• Allocate 10 advertising spots to 5 products

\[ x_i = \text{how many spots allocated to product } i \]
\[ y_{ij} = 1 \text{ if } j \text{ spots allocated to product } i \]

\[ P_{ij} = \text{profit from } \text{allocating } j \text{ spots to product } i \]

- \( \leq 4 \text{ spots per product} \)
- Advertise \( \leq 3 \text{ products} \)
- \( \geq 4 \text{ spots for at least one product} \)

Objective: maximize profit
Motivating example

spots in \{0..4\}
product in \{A,B,C,D,E\}

Index sets
Motivating example

spots in \{0..4\}
product in \{A,B,C,D,E\}

\texttt{data P\{product,spots\}}

Data input
Motivating example

spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)

Declaration of variable $x_i$
Motivating example

spots in {0..4}
product in {A, B, C, D, E}
data P{product, spots}

\[ x[i] \] is how many spots allocate(product i)

Declaration of variable \( x_i \)

This makes it a variable declaration
Motivating example

spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is how many spots allocate (product i)

Declaration of variable $x_i$

This is the semantic type
Motivating example

spots in \{0..4\}
product in \{A,B,C,D,E\}
data P\{product,spots\}
\(x[i] \text{ is } \boxed{\text{howmany}}\) spots allocate(product \(i\))

Indicates an integer quantity

Other keywords:
howmuch
whether
Motivating example

spots in \{0..4\}
product in \{A,B,C,D,E\}
data P\{product,spots\}

x[i] is how many \textbf{spots} allocate(product i)

How many of what?
Motivating example

spots in \{0..4\}
product in \{A,B,C,D,E\}
data P\{product,spots\}
\(x[i]\) is how many spots allocate\((product\ i)\)

Declaration of variable \(x_i\)

Predicate associated with variable \(x\)

Every variable is associated with a predicate that gives it meaning
Motivating example

spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is how many spots allocate(product i)
Motivating example

spots in \{0..4\}
product in \{A,B,C,D,E\}
data P\{product,spots\}
\[ x[i] \text{ is how many spots allocate}(product[i]) \]
Motivating example

\[ \text{max } \sum_{i} P_{ix_i} \]

spots in \{0..4\}
product in \{A,B,C,D,E\}
data \ P\{\text{product}, \text{spots}\}
x[i] \text{ is how many spots allocate(product i)}
\text{maximize sum\{product i\} } P[i,x[i]] \quad \text{Objective function}
Motivating example

\[
\begin{align*}
\text{max} & \quad \sum_i P_{ix_i} \\
\sum_i x_i & \leq 10
\end{align*}
\]

spots in \{0..4\}  
product in \{A,B,C,D,E\}  
data \text{P\{product,spots\}} 
\text{x[i] is how many spots allocate(product i)}  
maximize \text{sum\{product i\} P[i,x[i]]}  
\text{sum\{product i\} x[i] \leq 10} \quad 10 \text{ spots available}
Motivating example

\[
\begin{align*}
\text{max} & \quad \sum_i P_{ix_i} \\
\sum_i x_i & \leq 10
\end{align*}
\]

spots in \{0..4\}
product in \{A,B,C,D,E\}
data \ P\{\text{product},\text{spots}\}
x[i] \text{ is how many spots allocate(product } i) 
maximize \ \sum\{\text{product } i\} \ P[I,x[i]] 
sum\{\text{product } i\} \ x[i] \leq 10 
y[i,j] \text{ is whether allocate(product } i, \text{ spots } j) 

\text{Declare } y_{ij} 

Indicates 0-1 variable

\text{Indicates 0-1 variable}
Motivating example

\[
\begin{align*}
\text{max} & \sum_{i} P_{ix_i} \\
\sum_{i} x_i & \leq 10
\end{align*}
\]

spots in \{0..4\}
product in \{A,B,C,D,E\}
data \ P\{product,spots\}
x[i] is how many spots allocate(product i)
maximize sum\{product i\} \ P[i,x[i]]
sum\{product i\} \ x[i] \leq 10
y[i,j] is whether allocate(product i, spots j)

Declare \( y_{ij} \)

Associated with same predicate as \( x[i] \)
Motivating example

\[
\begin{align*}
\text{max} & \sum_i P_{ix_i} \\
\sum_i x_i & \leq 10, \quad \sum_i y_{i0} \geq 2
\end{align*}
\]

spots in \{0..4\}
product in \{A,B,C,D,E\}
data P\{product,spots\}
x[i] is how many spots allocate(product i)
maximize sum\{product i\} P[i,x[i]]
sum\{product i\} x[i] <= 10
y[i,j] is whether allocate(product i, spots j)
sum\{product i\} y[i,0] >= 2 \quad \text{At most 3 products advertised}
Motivating example

\[
\max \sum_i P_{ix_i} \\
\sum_i x_i \leq 10, \quad \sum_i y_{i0} \geq 2, \quad \sum_i y_{i4} \geq 1
\]

spots in \{0..4\}
product in \{A,B,C,D,E\}
data P\{product,spots\}
x[i] is howmany spots allocate(product i)
maximize sum\{product i\} P[i,x[i]]
sum\{product i\} x[i] \leq 10
y[i,j] is whether allocate(product i, spots j)
sum\{product i\} y[i,0] \geq 2
sum\{product i\} y[i,4] \geq 1  \quad \text{At least 1 product gets } \geq 4 \text{ spots}
Motivating example

spots in \{0..4\}
product in \{A,B,C,D,E\}
data \(P\{\text{product}, \text{spots}\}\)
\(x[i]\) is how many spots allocate \(\text{product} \ i\)
maximize \(\sum \{\text{product} \ i\} \ P[i,x[i]]\)
\(\sum\{\text{product} \ i\} \ x[i] \leq 10\)
y\([i,j]\) is whether allocate \(\text{product} \ i, \text{spots} \ j\)
\(\sum\{\text{product} \ i\} \ y[i,0] \geq 2\)
\(\sum\{\text{product} \ i\} \ y[i,4] \geq 1\)
\(\{\text{product} \ i\} \ \text{sum}\{\text{spots} \ j\} \ y[i,j] = 1\)
\(\{\text{product} \ i\} \ x[i] = \text{sum}\{\text{spots} \ j\} \ j*y[i,j]\)

Solver generates linking constraints because \(x[i]\) and \(y[i,j]\) are associated with the same predicate.

\[
\begin{align*}
\text{max} & \quad \sum_i P_{ix_i} \\
\sum_i x_i & \leq 10, \quad \sum_i y_{i0} \geq 2, \quad \sum_i y_{i4} \geq 1 \\
\sum_j y_{ij} & = 1, \quad x_i = \sum_j jy_{ij}, \text{ all } i
\end{align*}
\]
Motivating example

spots in \{0..4\}

product in \{A,B,C,D,E\}

data \( P\{\text{product, spots}\} \)

\( x[i] \) is how many spots allocate \((\text{product } i)\)

maximize sum\(\{\text{product } i\} P[i,x[i]]\)

This constraint must be linearized. Solver generates

\[
\begin{align*}
\max & \sum_i P_{ix_i} \\
\sum_i x_i & \leq 10, \quad \sum_i y_{i0} \geq 2, \quad \sum_i y_{i4} \geq 1 \\
\sum_j y_{ij} & = 1, \quad x_i = \sum_j jy_{ij}, \text{ all } i
\end{align*}
\]

\[
\begin{align*}
z_i = & \sum_{j=0}^{4} P_{ij} y'_{ij}, \quad \sum_{j=0}^{4} y'_i = 1, \quad x_i = \sum_{j=0}^{4} jy'_{ij}, \text{ all } i \\
y'[i,j] & \text{ is whether allocate(} \text{product } i, \text{ spots } j \text{)}
\end{align*}
\]
Motivating example

spots in \{0,\ldots,4\}
product in \{A, B, C, D, E\}
data \(P\{\text{product, spots}\}\)
x\[i\] is how many spots allocate (product \(i\))

\[
\begin{align*}
\max & \sum_i P_{ix_i} \\
\sum_i x_i & \leq 10, \quad \sum_i y_{i0} \geq 2, \quad \sum_i y_{i4} \geq 1 \\
\sum_j y_{ij} & = 1, \quad x_i = \sum_j jy_{ij}, \text{ all } i
\end{align*}
\]

This constraint must be linearized. Solver generates

\[
Z_i = \sum_{j=0}^{4} P_{ij} y'_j, \quad \sum_{j=0}^{4} y'_j = 1, \quad x_i = \sum_{j=0}^{4} jy'_j, \text{ all } i
\]

\(y'_[i,j]\) is whether allocate(product \(i\), spots \(j\))

\(y\) and \(y'\) are identified because they have the same type:

\(y[i,j]\) is whether allocate(product \(i\), spots \(j\))
Predicates and relations

Predicate **allocate** denotes 2-place **relation** (set of tuples). Schematically indicated by:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>$x_i$</td>
</tr>
<tr>
<td><strong>product</strong></td>
<td><strong>spots</strong></td>
</tr>
</tbody>
</table>
Predicates and relations

Predicate \textit{allocate} denotes 2-place \textit{relation} (set of tuples). Schematically indicated by:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>(x_i)</td>
</tr>
</tbody>
</table>

Column corresponding to a variable must be a \textit{function} of other columns.
## Predicates and relations

Predicate `allocate` denotes 2-place relation (set of tuples). Schematically indicated by:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>(x_i)</td>
</tr>
</tbody>
</table>

Declaration of \(y[i,j]\) as whether `allocate (product \(i\), spots \(j\))` creates the 3-place relation

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>(x_i)</td>
<td>(y_{ij})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>howmany</th>
<th>whether</th>
</tr>
</thead>
<tbody>
<tr>
<td>product</td>
<td>spots</td>
</tr>
</tbody>
</table>
Predicates and relations

Relation table reveals channeling constraints. For example,

\[ x[i] \text{ is which job assign(worker i)} \]
\[ y[j] \text{ is which worker assign (job i)} \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( j, x_i )</td>
<td>( i, y_j )</td>
</tr>
<tr>
<td>which</td>
<td>which</td>
<td></td>
</tr>
<tr>
<td>job</td>
<td>worker</td>
<td></td>
</tr>
</tbody>
</table>

We can read off the channeling constraints

\[ j = x_i = x_{y_i} \]
\[ i = y_j = y_{x_i} \]
Previous work

- **Model management** uses semantic typing to help combine models and use inheritance.
  - Originally inspired by object-oriented programming
    Bradley & Clemence (1988)
  - *Quiddity*: a rigorous attempt to analyze conditions for variable identification
    Bhargava, Kimbrough & Krishnan (1991)
  - **SML** uses typing in a structured modeling framework
    Geoffrion (1992)
  - **Ascend** uses strongly-typed, object-oriented modeling
    Bhargava, Krishnan & Piela (1998)
Previous work

• Our semantic typing differs:
  • **Less ambitious** because it doesn’t attempt model management.
    • There is only one model.
  • **More ambitious** because we recognize relationships other than equivalence.
  • We manage variables *introduced by solver*. 
Previous work

• Modeling systems that convey some structure to solver:
  • All CP modelers (OPL, CHIP, etc.) use global constraints.
  • AIMMS uses typed index sets.
  • OPL, Xpress-Kalis, Comet, etc., use interval variables.
  • SAT solver SymChaff uses high-level AI planning language PDDL.
  • Lopes and Fourer (2009) use UML (Unified Modeling Language) to model multistage stochastic LPs with recourse.
  • SIMPL has full metaconstraint capability.
Previous work

• However, none of these systems deals systematically with the variable management problem.
  
  • We address it with semantic typing of variables.
Assignment problem

\[
\min \sum_{i} c_{ix_i}
\]
\[
\text{alldiff} \left( x_1, \ldots, x_n \right)
\]

worker in \{1..m\}
job in \{1..n\}
data C\{worker, job\}
\(x[i]\) is which job assign(worker i)
minimize sum\{worker i\} C[i,x[i]]
alldiff\{x[*]\}
Assignment problem

\[
\begin{align*}
\text{min } & \sum_{i} c_{ix_i} \\
\text{alldiff } & (x_1, \ldots, x_n)
\end{align*}
\]

worker in \{1..m\}
job in \{1..n\}
data \ C\{\text{worker,job}\}
x[i] \text{ is which job assign(} \text{worker i})
minimize \ \sum \{\text{worker i} \} C[i,x[i]]
alldiff\{x[*]\}

Objective function is formulated

\[
\begin{align*}
\max & \sum_{i} c_{ij} y_{ij}, \ x_i = \sum_{j} y_{ij}, \text{ all } i \\
y[i,j] \text{ is whether assign(} \text{worker i, job j})
\end{align*}
\]
Assignment problem

\[
\min \sum_{i} c_{ix_i}
\]
\[
\text{alldiff}\left(x_1, \ldots, x_n\right)
\]

worker \text{ in } \{1..m\}
job \text{ in } \{1..n\}
data \text{ C\{worker,job\}}
x[i] \text{ is which job assign(worker i)}
minimize \text{ sum\{worker i\} C[i,x[i]]}
alldiff\{x[\ast]\}

Objective function is formulated
\[
\max \sum_{i} c_{ij}y_{ij}, \ x_i = \sum_{j} y_{ij}, \text{ all } i
\]
y[i,j] \text{ is whether assign(worker i, job j)}

Alldiff is formulated
\[
\sum_{j} y'_{ij} = 1, \text{ all } i, \ \sum_{i} y'_{ij} = 1, \text{ all } j, \ x_i = \sum_{j} jy'_{ij}, \text{ all } i
\]
y'[i,j] \text{ is whether assign(worker i, job j)}

Solver identifies y and y' to create classical AP.
Latin squares

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Numbers in every row and column are distinct.
We will use **three** formulations to improve propagation.
Latin squares

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Numbers in every row and column are distinct.
We will use **three** formulations to improve propagation.

\[
\text{row, col, num in \{1..n\}}
\]

\[
x[i,j] \text{ is which num assign(row i, col j)}
\]

\[
y[i,k] \text{ is which col assign(row i, num k)}
\]

\[
z[j,k] \text{ is which row assign(col j, num k)}
\]

\[
\text{alldiff}(x_{i1}, \ldots x_{in}), \text{ all } i
\]

\[
\text{alldiff}(x_{1j}, \ldots x_{nj}), \text{ all } j
\]

\[
\text{alldiff}(y_{i1}, \ldots y_{in}), \text{ all } i
\]

\[
\text{alldiff}(y_{1k} \ldots y_{nk}), \text{ all } k
\]

\[
\text{alldiff}(z_{j1}, \ldots x_{jn}), \text{ all } j
\]

\[
\text{alldiff}(z_{1k}, \ldots x_{nk}), \text{ all } k
\]
Latin squares

Numbers in every row and column are distinct.
We will use **three** formulations to improve propagation.

row, col, num in {1..n}
x[i,j] is which num assign(row i, col j)
y[i,k] is which col assign(row i, num k)
z[j,k] is which row assign(col j, num k)

\[
\begin{align*}
\text{alldiff} (x_{i1}, \ldots x_{in}), \text{ all } i \\
\text{alldiff} (x_{1j}, \ldots x_{nj}), \text{ all } j \\
\text{alldiff} (y_{i1}, \ldots y_{in}), \text{ all } i \\
\text{alldiff} (y_{1k} \ldots y_{nk}), \text{ all } k \\
\text{alldiff} (z_{j1}, \ldots x_{jn}), \text{ all } j \\
\text{alldiff} (z_{1k} \ldots x_{nk}), \text{ all } k
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The predicate \texttt{assign} denotes the 3-place relation \(k, x_{ij}\) \(j, y_{ik}\) \(i, z_{jk}\) which \texttt{num}\ \texttt{col}\ \texttt{row}

\begin{tabular}{|c|c|c|}
\hline
1 & 2 & 3 \\
\hline
\texttt{k, } & \texttt{j, } & \texttt{i, } \\
\texttt{x}_{ij} & \texttt{y}_{ik} & \texttt{z}_{jk} \\
\hline
\texttt{which} & \texttt{which} & \texttt{which} \\
\texttt{num} & \texttt{col} & \texttt{row} \\
\hline
\end{tabular}

\begin{itemize}
\item \texttt{row, col, num in \{1..n\}}
\item \texttt{x[i,j] is which num assign(row i, col j)}
\item \texttt{y[i,k] is which col assign(row i, num k)}
\item \texttt{z[j,k] is which row assign(col j, num k)}
\item \texttt{\{row i\} alldiff\{x[i,*]\}; \{col j\} alldiff\{x[*,j]\}}
\item \texttt{\{row i\} alldiff\{y[i,*]\}; \{num k\} alldiff\{y[*,j]\}}
\item \texttt{\{col j\} alldiff\{z[j,*]\}; \{num k\} alldiff\{z[*,k]\}}
\end{itemize}
Latin squares

The predicate \texttt{assign} denotes the 3-place relation

\[
\begin{array}{c|c|c}
1 & 2 & 3 \\
\hline
k, x_{ij} & j, y_{ik} & i, z_{jk} \\
\text{which} & \text{which} & \text{which} \\
\text{num} & \text{col} & \text{row} \\
\end{array}
\]

We can read off the channeling constraints:

\[
k = x_{z_{jk}y_{ik}}, \quad j = y_{z_{jk}x_{ij}}, \quad i = z_{y_{ik}x_{ikj}}, \quad \text{all } i, j, k
\]

which can be propagated.
Latin squares

\{\text{row \ i}\} \ \text{alldiff}\{x[i,*]\}; \ \{\text{col \ j}\} \ \text{alldiff}\{x[*,j]\}
\{\text{row \ i}\} \ \text{alldiff}\{y[i,*]\}; \ \{\text{num \ k}\} \ \text{alldiff}\{y[*,j]\}
\{\text{col \ j}\} \ \text{alldiff}\{z[j,*]\}; \ \{\text{num \ k}\} \ \text{alldiff}\{z[*,k]\}

The 3 formulations generate 3 identical MIP models:

\[x_{ij} = \sum_k k \delta^{x}_{ijk}; \ \sum_k \delta^{x}_{ijk} = 1, \ \text{all \ } i, j; \ \sum_j \delta^{x}_{ijk} = 1, \ \text{all \ } i, k; \ \sum_i \delta^{x}_{ijk} = 1, \ \text{all \ } j, k\]

\[y_{ik} = \sum_j j \delta^{y}_{ijk}; \ \sum_j \delta^{y}_{ijk} = 1, \ \text{all \ } i, k; \ \sum_k \delta^{y}_{ijk} = 1, \ \text{all \ } i, j; \ \sum_i \delta^{y}_{ijk} = 1, \ \text{all \ } j, k\]

\[z_{jk} = \sum_i i \delta^{z}_{ijk}; \ \sum_i \delta^{z}_{ijk} = 1, \ \text{all \ } j, k; \ \sum_k \delta^{z}_{ijk} = 1, \ \text{all \ } i, j; \ \sum_j \delta^{z}_{ijk} = 1, \ \text{all \ } i, k\]
The 3 formulations generate 3 identical MIP models:

\[
\begin{align*}
    x_{ij} &= \sum_k k \delta_{ijk}^x, \quad \sum_k \delta_{ijk}^x = 1, \text{ all } i, j; \quad \sum_j \delta_{ijk}^x = 1, \text{ all } i, k; \quad \sum_i \delta_{ijk}^x = 1, \text{ all } j, k \\
    y_{ik} &= \sum_j j \delta_{ijk}^y, \quad \sum_j \delta_{ijk}^y = 1, \text{ all } i, k; \quad \sum_k \delta_{ijk}^y = 1, \text{ all } i, j; \quad \sum_i \delta_{ijk}^y = 1, \text{ all } j, k \\
    z_{jk} &= \sum_i i \delta_{ijk}^z, \quad \sum_i \delta_{ijk}^z = 1, \text{ all } j, k; \quad \sum_k \delta_{ijk}^z = 1, \text{ all } i, j; \quad \sum_j \delta_{ijk}^z = 1, \text{ all } i, k
\end{align*}
\]

The solver declares \( \delta_{ijk}^x, \delta_{ijk}^y, \delta_{ijk}^z \) whether assign(row i, col j, num k)

So it treats them as the same variable and generates only 1 MIP model.
Relating which variables

In general, an $n$-place predicate that denotes the relation

<table>
<thead>
<tr>
<th></th>
<th>...</th>
<th></th>
<th>...</th>
<th></th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_1, x_{i(1)}$</td>
<td>...</td>
<td>$i_k, x_{i(k)}$</td>
<td>$i_{k+1}$</td>
<td>...</td>
<td>$i_n$</td>
</tr>
<tr>
<td>\textit{which}</td>
<td>...</td>
<td>\textit{which}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{entity}_1</td>
<td>...</td>
<td>\text{entity}_k</td>
<td>\text{entity}_{k+1}</td>
<td>...</td>
<td>\text{entity}_n</td>
</tr>
</tbody>
</table>

where $i(j) = i_1 \cdots i_{j-1}i_{j+1} \cdots i_n$

generates the channeling constraints

\[ i_j = x_j^{j} \times x_{i(1)}^{j} \cdots x_{i(j-1)}^{j} \times x_{i(j+1)}^{j} \cdots x_{i(k)}^{j} \times i_{k+1} \cdots i_n, \text{ all } i_1, \ldots, i_n, j = 1, \ldots, k \]
Piecewise linear

Piecewise linear function \( z = f(x) \)

Breakpoints in \( A \), ordinates in \( C \)

\( x \) is how much output

index in \( \{1..n\} \)

data \( A,C\{\text{index i}\} \)

\( z \) is how much cost

\text{piecewise}(x,z,A,C) \quad \text{this metaconstraint defines } z = f(x)
Piecewise linear

Piecewise linear function $z = f(x)$
Breakpoints in $A$, ordinates in $C$

$x$ is how much output
index in $\{1..n\}$
data $A,C$ {index i}
z is how much cost
piecewise $(x,z,A,C)$

Solver generates the model

\[
x = a_1 + \sum_{i=1}^{n-1} x_i, \quad z = c_1 + \sum_{i=1}^{n-1} \frac{c_{i+1} - c_i}{a_{i+1} - a_i} x_i
\]

\[
(a_{i+1} - a_i) \delta_{i+1} \leq x_i \leq (a_{i+1} - a_i) \delta_i, \quad \delta_i \in \{0,1\}, \quad i = 1,\ldots,n-1
\]

We need to declare auxiliary variables $\delta_i, x_i$
Piecewise linear

Piecewise linear function $z = f(x)$
Breakpoints in $A$, ordinates in $C$

$x$ is howmuch output
index in $\{1..n\}$
data $A,C\{\text{index } i\}$
z is howmuch cost
$\text{piecewise}(x,z,A,C)$

piecewise constraint induces solver to declare a new index
set that associates $\text{index}$ with $A$, and use it to declare $\delta_i$, $x_i$

$\text{indexA in } \{1..n\}$
delta[i] is whether output(output(indexA i))
x[i] is howmuch output(output(indexA i))

Both declarations create predicates inherited from output
Piecewise linear

Suppose there is another piecewise function on the same break points

\[ x \text{ is howmuch output} \]
\[ \text{index in } \{1..n\} \]
\[ \text{data } A,C\{\text{index } i\} \]
\[ z \text{ is howmuch cost} \]
\[ \text{piecewise}(x,z,A,C) \]
\[ \text{data } C'\{\text{index } i\} \]
\[ z' \text{ is howmuch profit} \]
\[ \text{piecewise}(x,z',A,C') \]
\[ \text{indexA in } \{1..n\} \]
\[ \delta[i] \text{ is whether output(indexA i)} \]
\[ x[i] \text{ is howmuch output(indexA i)} \]
Piecewise linear

Suppose there is another piecewise function on the same break points

\[ x \text{ is howmuch output index in } \{1..n\} \]
\[ \text{data } A,C\{\text{index } i}\]  
\[ z \text{ is howmuch cost} \]
\[ \text{piecewise}(x,z,A,C) \]
\[ \text{data } C'\{\text{index } i}\]  
\[ z' \text{ is howmuch profit} \]
\[ \text{piecewise}(x,z',A,C') \]
\[ \text{indexA in } \{1..n\} \]
\[ \delta[i] \text{ is whether output(indexA } i) \]
\[ x[i] \text{ is howmuch output(indexA } i) \]

MIP model creates variables \(\delta_i'\) and \(x_i'\) with same types as \(\delta_i\) and \(x_i\) and so identifies them.

Because new piecewise constraint is associated with the same \(x\) and \(A\), solver again creates \textbf{indexA}.
Each job $j$ runs for a time interval $x_j$.
We wish to schedule jobs so that total resource consumption never exceeds $L$.

\[
\text{cumulative}(x, D, R, L) \quad x_j \subseteq W_j, \text{ all } j
\]
**Interval variables**

Each job \( j \) runs for a time interval \( x_j \). We wish to schedule jobs so that total resource consumption never exceeds \( L \).

- job in \( \{1..n\} \)
- time in \( \{t..T\} \)
- data \( W\{\text{job}\}, D\{\text{job}\}, R\{\text{job}\} \) window, duration, resource
- \( x[j] \) is interval running\((\text{job } j) \) in \( W[j] \)
- cumulative\((x,D,R,L)\)

Solver generates the MIP model

\[
\sum_t \delta_{jt} = 1, \text{ all } j; \quad \sum_j R_j \phi_{jt} \leq L, \text{ all } t
\]

\[
\phi_{jt} \geq \delta_{jt}, \text{ all } t,t' \text{ with } 0 \leq t - t' < D_j, \text{ all } j
\]

- \( \delta[j,t] \) is whether running.start(\( \text{job } j, \text{ time } t \) )
- \( \phi[j,t] \) is whether running(\( \text{job } j, \text{ time } t \) )
Interval variables

Suppose we want finish times to be separated by at least $T_0$

- job in $\{1..n\}$
- time in $\{t..T\}$
- data $W\{\text{job}\}$, $D\{\text{job}\}$, $R\{\text{job}\}$
- $x[j]$ is interval running(job j) in $W[j]$
- $\text{cumulative}(x, D, R, L)$
- $\{\text{job j, job k}\} |x[j].end - x[k].end| \geq T_0$
- $\delta[j,t]$ is whether running.$\text{start}(\text{job j, time t})$
- $\phi[j,t]$ is whether running(\text{job j, time t})
Interval variables

Suppose we want finish times to be separated by at least $T_0$

job in $\{1..n\}$
time in $\{t..T\}$
data $W\{job\}, D\{job\}, R\{job\}$
$x[j]$ is interval running(job j) in $W[j]$
cumulative($x,D,R,L$)

\[
\{\text{job } j, \text{ job } k\} \mid |x[j].\text{end} - x[k].\text{end}| \geq T_0
\]
delta[$j,t$] is whether running.start(job j, time t)
phi[$j,t$] is whether running(job j, time t)

Solver generates

\[
\varepsilon_{jt} + \varepsilon_{kt'} \leq 1, \text{ all } t,t' \text{ with } 0 < t' - t < T_0, \text{ all } j,t \text{ with } j \neq k
\]
epsilon[$j,t$] is whether running.end(job j, time t)
Interval variables

Variables $\delta_{jt}$ and $\varepsilon_{jt}$ are related by an offset. Solver associates `running.end` in declaration of $\varepsilon_{jt}$ with `running.start` in declaration of $\delta_{jt}$ and deduces

$$e_{j,t+D_j} = \delta_{jt}, \text{ all } j,t$$

$\delta_{jt}$ is whether `running.start(job j, time t)`

$\phi_{jt}$ is whether `running(job j, time t)`

$\varepsilon_{jt}$ is whether `running.end(job j, time t)`
Interval variables

Variables $\delta_{jt}$ and $\varepsilon_{jt}$ are related by an offset.
Solver associates `running.end` in declaration of $\varepsilon_{jt}$ with `running.start` in declaration of $\delta_{jt}$ and deduces

$$e_{j,t+D_j} = \delta_{jt}, \text{ all } j,t$$

Solver also associates `running.end` in declaration of $\varepsilon_{jt}$ with `running` in declaration of $\phi_{jt}$ and deduces the redundant constraints

$$\phi_{jt} \geq \varepsilon_{jt'}, \text{ all } t,t' \text{ with } 0 \leq t' - t < D_j, \text{ all } j$$

`delta[j,t]` is whether `running.start(job j, time t)`
`phi[j,t]` is whether `running(job j, time t)`
`epsilon[j,t]` is whether `running.end(job j, time t)`
Pros and Cons of Semantic Typing

• **Pros**
  
  • Conveys problem structure to MIP solver
    – …by allowing use of metaconstaints
  
  • Incorporates state of the art in formulation, valid inequalities
  
  • Allows solver to expand repertory of techniques
    – Domain filtering, propagation
  
  • Good modeling practice
    – Self-documenting
    – Bug detection
Pros and Cons of Semantic Typing

• Cons
  • Modeler must be familiar with a large collection of metaconstraints
    – Rather than few primitive constraints
Pros and Cons of Semantic Typing

• Cons
  • Modeler must be familiar with a large collection of metaconstraints
    – Rather than few primitive constraints
  • Response
    – Modeler must be familiar with the underlying concepts anyway
    – Modeling system can offer sophisticated help, improve modeling
Pros and Cons of Semantic Typing

• Cons
  • Modeler must be familiar with a large collection of metaconstraints
    – Rather than few primitive constraints
  • Response
    – Modeler must be familiar with the underlying concepts anyway
    – Modeling system can offer sophisticated help, improve modeling
  • OR community is not accustomed to high-level modeling
    – Typed languages like Ascend never really caught on.
Pros and Cons of Semantic Typing

- **Cons**
  - Modeler must be familiar with a large collection of metaconstraints
    - Rather than few primitive constraints
  - **Response**
    - Modeler must be familiar with the underlying concepts anyway
    - Modeling system can offer sophisticated help, improve modeling

- **OR community is not accustomed to high-level modeling**
  - Typed languages like Ascend never really caught on.

- **Response**
  - Train the next generation!