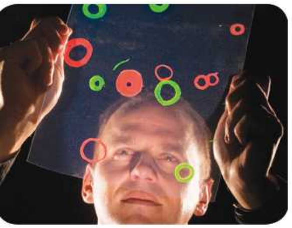
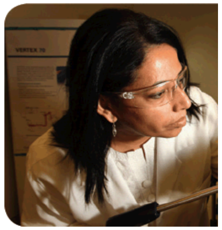
SAND2015-0009C

Leveraging Model Transformations in Algebraic Modeling Systems


John D. Siirola
William E. Hart
Jean-Paul Watson

Discrete Math & Optimization (1464)
Sandia National Laboratories

Exceptional service in the national interest

INFORMS Computing Society Conference
11 January 2015

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

What do these have in common?

$$a = b + c$$
$$b \le M \cdot y$$
$$y \le M(1 - y)$$
$$x - 3 = c - b$$
$$b \ge 0$$
$$c \ge 0$$
$$y \in \{0,1\}$$

$$a = \sqrt{(x-3)^2 + \epsilon}$$

$$a \ge x - 3$$
$$a \ge 3 - x$$

$$a = b + c$$
$$x - 3 = c - b$$
$$b \ge 0 \perp c \ge 0$$

$$a = \frac{2(x-3)}{1 + e^{-\frac{x-3}{h}}} - x + 3$$

What do these have in common?

$$a = b + c$$
$$b \leq M \cdot y$$
$$y \leq M(1 - y)$$
$$x - 3 = c - b$$
$$b \geq 0$$
$$c \geq 0$$
$$y \in \{0,1\}$$

$$a = \sqrt{(x-3)^2 + \epsilon}$$

$$a = abs(x - 3)$$

$$a \geq x - 3$$
$$a \geq 3 - x$$

$$a = b + c$$
$$x - 3 = c - b$$
$$b \geq 0 \perp c \geq 0$$

$$a = \frac{2(x-3)}{1 + e^{-\frac{x-3}{h}}} - x + 3$$

If we *mean* “$a = abs(x - 3)$”,
why don’t we *write* that in our models???

What's a Model?

- A general representation of a class of problems
  - Data (instance) independent

Model + Data → Instance

- Represents our understanding of the class of problems
  - Explicitly annotates and conveys the class structure
    - Hierarchical? Separable? Graph based?
  - Sets, vectors, matrices
- Incorporates assumptions and simplifications
- Is both tractable and valid
  - (although these are often contradictory goals)

Transformations help here

Why are we interested in transformations?

- Separate model expression from how we intend to solve it
  - Defer decisions that improve tractability until solution time
  - Explore alternative reformulations or representations
  - Support *solver-specific* model customizations (e.g., `abs()`)
  - Support iterative methods that use different solvers requiring different representations (e.g., initializing NLP from MIP)
- Support "higher level" or non-algebraic modeling constructs
  - Express models that are "closer" to reality, e.g.:
    - Piecewise expressions
    - Disjunctive models (switching decisions & logic models)
    - Differential-algebraic models (dynamic models)
    - Bilevel models (game theory models)
- Reduce "mechanical" errors due to manual reformulation

A rose by any other name…

- Is this a *reformulation* or a *transformation?*

$$\begin{gathered} a = b + c \\ b \leq M \cdot y \\ y \leq M(1 - y) \\ x - 3 = c - b \\ b \geq 0 \\ c \geq 0 \\ y \in \{0,1\} \end{gathered}$$

$\leftrightarrow$

$$a = abs(x - 3)$$

- Literature is not clear here; potential distinctions:
  - Does it preserve the same feasible space (or relaxed space)?
  - Is the operation reversible?
  - …

A rose by any other name…

- Is this a *reformulation* or a *transformation?*

$$a = b + c$$
$$b \leq M \cdot y$$
$$y \leq M(1 - y)$$
$$x - 3 = c - b$$
$$b \geq 0$$
$$c \geq 0$$
$$y \in \{0,1\}$$

$\leftrightarrow$

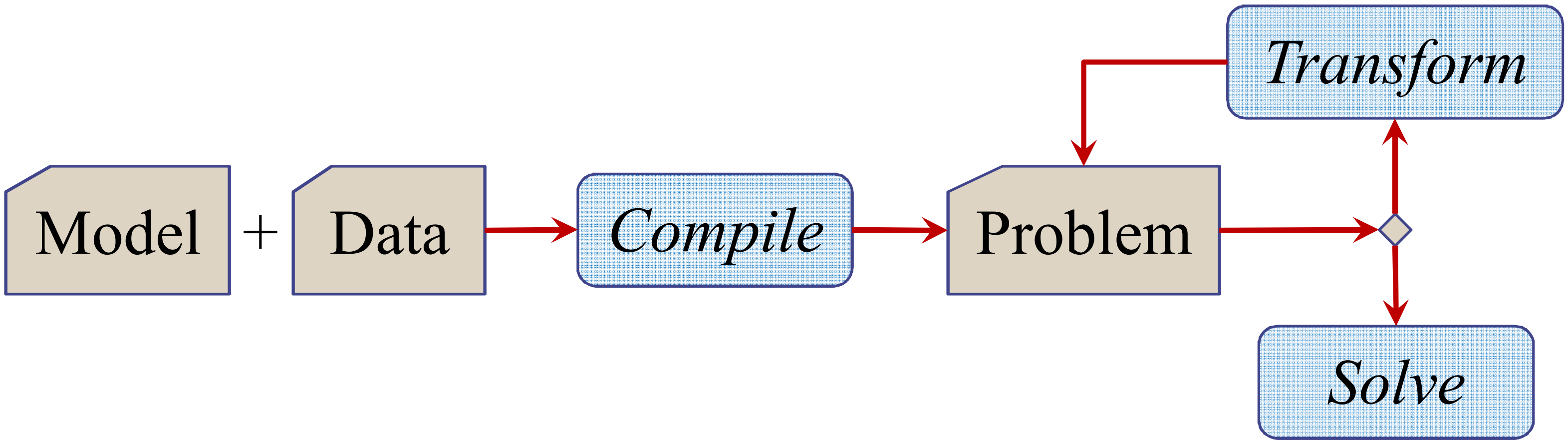
$$a = abs(x - 3)$$

- Literature is not clear here; potential distinctions:
  - Does it preserve the same feasible space (or relaxed space)?
  - Is the operation reversible?
  - …

A *Transformation* is any *Reformulation* that can be automatically applied

A new solution workflow

- Model Transformations: *Projecting problems to problems*
  - Project from one problem space to another
  - Standardize common reformulations or approximations
  - Convert “unoptimizable” modeling constructs into equivalent optimizable forms



Transformations are not entirely new

- LINGO's automatic linearization:

```
MODEL:
  MIN = @ABS( X-3 );
  X <= 2;
END
```

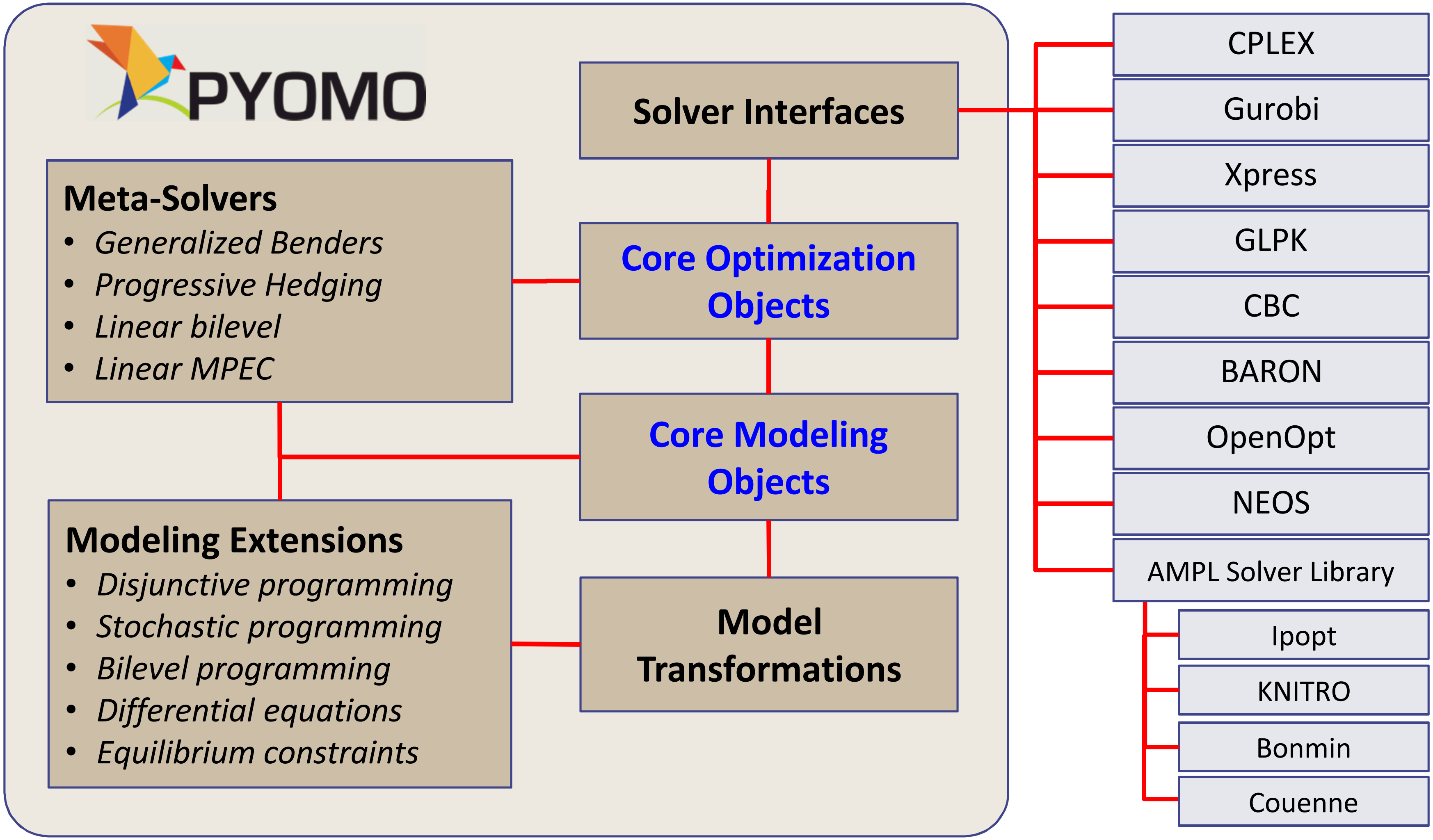
  - Generates the "usual" Big-M integer linear model:

```
MAX _C3
SUBJECT TO
  X <= 2
  - _C1 - _C2 + _C3 = 0
  _C1 – 100000 _C4 <= 0
  _C2 + 100000 _C4 <= 100000
  X - _C1 + _C2 = 3
END
INTE _C4
```

Cunningham and Schrage, "The LINGO Algebraic Modeling Language." In *Modeling Languages in Mathematical Optimization*, Josef Kallrath ed. Springer, 2004.

Pyomo: *Python Optimization Modeling Objects*

Sandia National Laboratories



A Quick Tour of Pyomo

Idea: a Pythonic framework for formulating optimization models

- Provides a natural syntax to describe mathematical models
- Leverages an extensible optimization object model
- Formulates large models with a concise syntax
- Separates modeling and data declarations
- Enables data import and export in commonly used formats

Highlights:

- Python provides a clean, readable syntax
- Python scripts provide a flexible context for exploring the structure of Pyomo models

```
from pyomo.environ import *

model = ConcreteModel()

model.x1 = Var()
model.x2 = Var(bounds=(-1,1))
model.x3 = Var(bounds=(1,2))

model.obj = Objective(
    expr= m.x1**2 + (m.x2*m.x3)**4 +
          m.x2*sin(m.x1+m.x3) + m.x2,
    sense= minimize)
```

Why transformations in Pyomo?

- Pyomo is an *object model*
  - Extensions declare new object classes (*components*)
    - Supports annotating model components
  - Transformations can detect presence of relevant components
  - Core code (e.g., problem writers) can validate supported components
  - Whole model (including expressions) is transparent and manipulable
- Pyomo natively supports hierarchical models
  - "Block": collection of modeling components (e.g., Sets, Prams, Vars)
    - Namespacing: component names must only be unique within a block
    - Blocks can contain blocks: hierarchical structure
  - Many modeling extenions derive from Block
  - Transformations can be "sandboxed" in transformation-specific Blocks

An example: Disjunctive programming

- Disjunctions: selectively enforce sets of constraints
  - Sequencing decisions: x ends before y or y ends before x
  - Switching decisions: a process unit is built or not
  - Alternative selection: selecting from a set of pricing policies

- Implementation: leverage Pyomo "blocks"
  - `Disjunct`:
    - Block of Pyomo components
      - (Var, Param, Constraint, etc.)
    - Boolean (binary) indicator variable determines if block is enforced
  - `Disjunction`:
    - Enforces logical XOR across a set of Disjunct indicator variables
  - (Logic constraints on indicator variables)

$$\bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ h_{ik}(x) \le o \\ c_k = \gamma_{ik} \end{bmatrix}$$

$$\Omega(Y) = true$$

Example: Task sequencing

- Prevent tasks colliding on a single piece of equipment
  - Derived from Raman & Grossmann (1994)
  - Given:
    - Tasks I processed on a sequence of machines (with no waiting)
    - Task i starts processing at time t_i with duration τ_{im} on machine m
    - $J(i)$ is the set of machines used by task i
    - C_{ik} is the set of machines used by both tasks i and j

$$\begin{bmatrix} Y_{ik} \\ t_i + \sum_{\substack{m \in J(i) \\ m \le j}} \tau_{im} \le t_k + \sum_{\substack{m \in J(k) \\ m < j}} \tau_{km} \end{bmatrix} \vee \begin{bmatrix} Y_{ki} \\ t_k + \sum_{\substack{m \in J(k) \\ m \le j}} \tau_{km} \le t_i + \sum_{\substack{m \in J(i) \\ m < j}} \tau_{im} \end{bmatrix}$$

$$\forall j \in C_{ik}, \forall i, k \in I, i < k$$

Example: Task sequencing in Pyomo

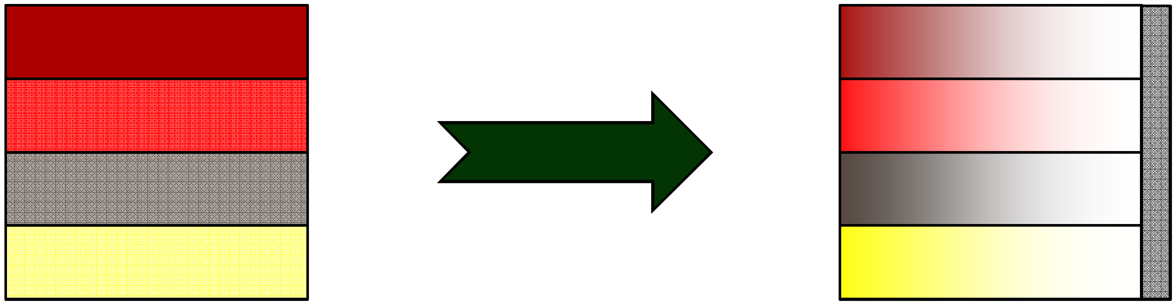
```
def _NoCollision(model, disjunct, i, k, j, ik):
    lhs = model.t[i] + sum(model.tau[i,m] for m in model.STAGES if m<j)
    rhs = model.t[k] + sum(model.tau[k,m] for m in model.STAGES if m<j)
    if ik:
        disjunct.c = Constraint( expr= lhs + model.tau[i,j] <= rhs )
    else:
        disjunct.c = Constraint( expr= rhs + model.tau[k,j] <= lhs )
model.NoCollision = Disjunct( model.L, [0,1], rule=_NoCollision )

def _setSequence(model, i, k, j):
    return [ model.NoCollision[i,k,j,ik] for ik in [0,1] ]
model.setSequence = Disjunction(model.L, rule=_setSequence)
```

$$\left[\begin{array}{c} Y_{ik} \\ t_i + \sum_{\substack{m \in J(i) \\ m<j}} \tau_{im} + \tau_{ij} \le t_k + \sum_{\substack{m \in J(k) \\ m<j}} \tau_{km} \end{array}\right] \vee \left[\begin{array}{c} Y_{ki} \\ t_k + \sum_{\substack{m \in J(k) \\ m<j}} \tau_{km} + \tau_{kj} \le t_i + \sum_{\substack{m \in J(i) \\ m<j}} \tau_{im} \end{array}\right]$$

$$\forall j \in C_{ik}, \forall i,k \in I, i<k$$

Solving disjunctive models

- Few solvers "understand" disjunctive models
  - *Transform* model into standard math program
  - Big-M relaxation:
    - Convert logic variables to binary
    - Split equality constraints in disjuncts into pairs of inequality constraints
    - Relax all constraints in the disjuncts with "appropriate" M values



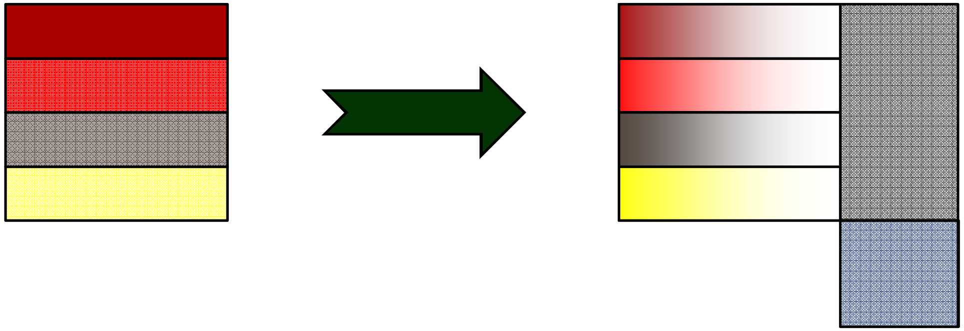
```
pyomo --preprocess=coopr.gdp.bigm jobshop.py jobshop.dat
```

Why is the transformation interesting?

- Model preserves explicit disjunctive structure
- Automated transformation reduces errors
- Automatically identifies appropriate M values (for bounded linear)

Why is the transformation interesting?

- Model preserves explicit disjunctive structure
- Automated transformation reduces errors
- Automatically identifies appropriate M values (for bounded linear)
- Big-M is not the only way to relax a disjunction!
  - Convex hull transformation (Balas, 1985; Lee and Grossmann, 2000)



```
pyomo --preprocess=coopr.gdp.chull jobshop.py jobshop.dat
```

  - Algorithmic approaches
    - e.g., Trespalacios and Grossmann (submitted 2013)
  - Prematurely choosing one relaxation makes trying others difficult

A growing library of transformations

- Bilinear relaxations
- Complementarity / Equilibrium constraints
  - Nonlinear relaxation
  - Disjunctive relaxation
  - “Standard” form relaxation
- Disjunctive programming
  - Big-M reformulation
  - Convex Hull reformulation
  - Hybrid Basic-Step based algorithm
- Dynamic systems
  - Collocation on finite elements
  - Finite difference discretization
- Bilevel optimization
  - Linear dual reformulation
  - Linear complementarity (KKT) reformulation
- Structural transformations
  - Relax integrality
  - Standard linear form
  - Dual transformation
  - Eliminate fixed variables
  - Nonnegative transform
  - Equality transform

Back to our original example: ABS(x)

- Chaining transformations

$$f = abs(x) \Rightarrow \begin{array}{c} f = x^+ + x^- \\ x = x^+ - x^- \\ x^+ \geq 0 \perp x^- \geq 0 \end{array} \Rightarrow \begin{array}{c} f = x^+ + x^- \\ x = x^+ - x^- \\ \begin{bmatrix} Y \\ x^- = 0 \end{bmatrix} \vee \begin{bmatrix} \neg Y \\ x^+ = 0 \end{bmatrix} \\ x^+ \geq 0,\ x^- \geq 0 \end{array} \Rightarrow \begin{array}{c} f = x^+ + x^- \\ x = x^+ - x^- \\ x^- \leq My \\ x^- \leq M(1-y) \\ x^+ \geq 0,\ x^- \geq 0 \end{array}$$

```
model = ConcreteModel()
# […]
TransformFactory("abs.complements").apply(model, inplace=True)
TransformFactory("mpec.disjunctive").apply(model, inplace=True)
TransformFactory("gdp.bigm").apply(model, inplace=True)
```

An open issue: back-mapping

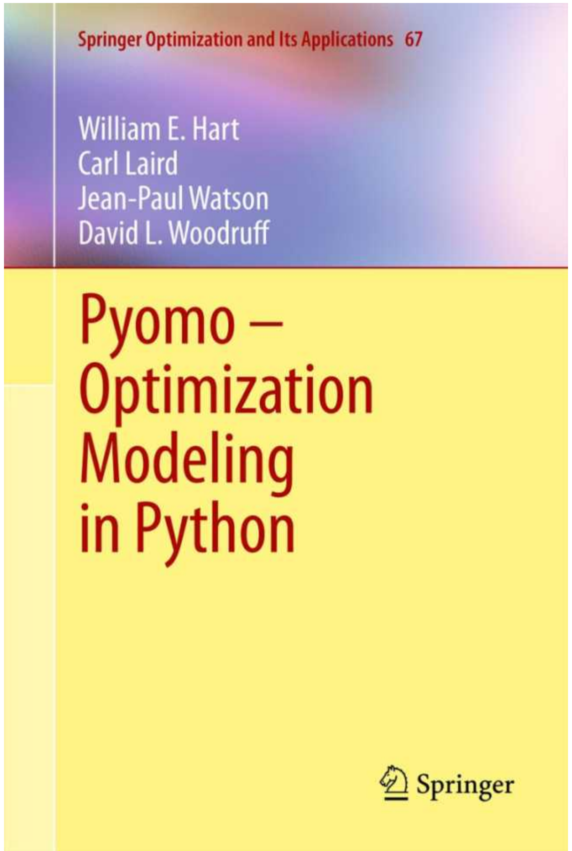
- Transformations can fundamentally alter problem instance.
  - Presenting results is best done in the original problem context
  - How to (automatically) map results from the transformed space back to the original instance?
    - Many transformations are not isomorphic

Summary

- Model transformations can significantly impact modeling
  - Separates the intent of the Modeler from the needs of the solver
  - Expands the set of (high-level) modeling constructs
    - Models can closer represent how a Modeler "thinks"
  - Defers decisions on how to map the problem class to the solver to just before solve time
  - Reduces / eliminates manual transcription errors
  - Chaining transformations is a powerful operation
    - Complex transformations are cast as a series of simpler operations
    - Availability of alternative transformation routes is preserved

For more information…

- Project homepage
  - http://www.pyomo.org
  - https://software.sandia.gov/pyomo
- Mailing lists
  - "pyomo-forum" Google Group
  - "pyomo-developers" Google Group
- "The Book"



- Mathematical Programming Computation paper:
  - Pyomo: Modeling and Solving Mathematical Programs in Python (3(3), 2011)