Logical Modeling Frameworks for the Optimization of Discrete-Continuous Systems

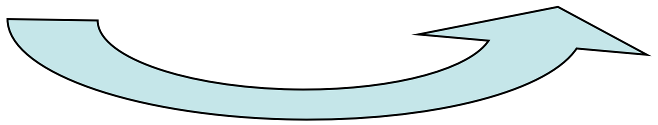
Ashish Agarwal
Department of Chemical Engineering, Carnegie Mellon University

Advisors:
Ignacio E. Grossmann
Department of Chemical Engineering, Carnegie Mellon University
Robert Harper
Computer Science Department, Carnegie Mellon University

PhD Defense
May 5, 2006

Modeling

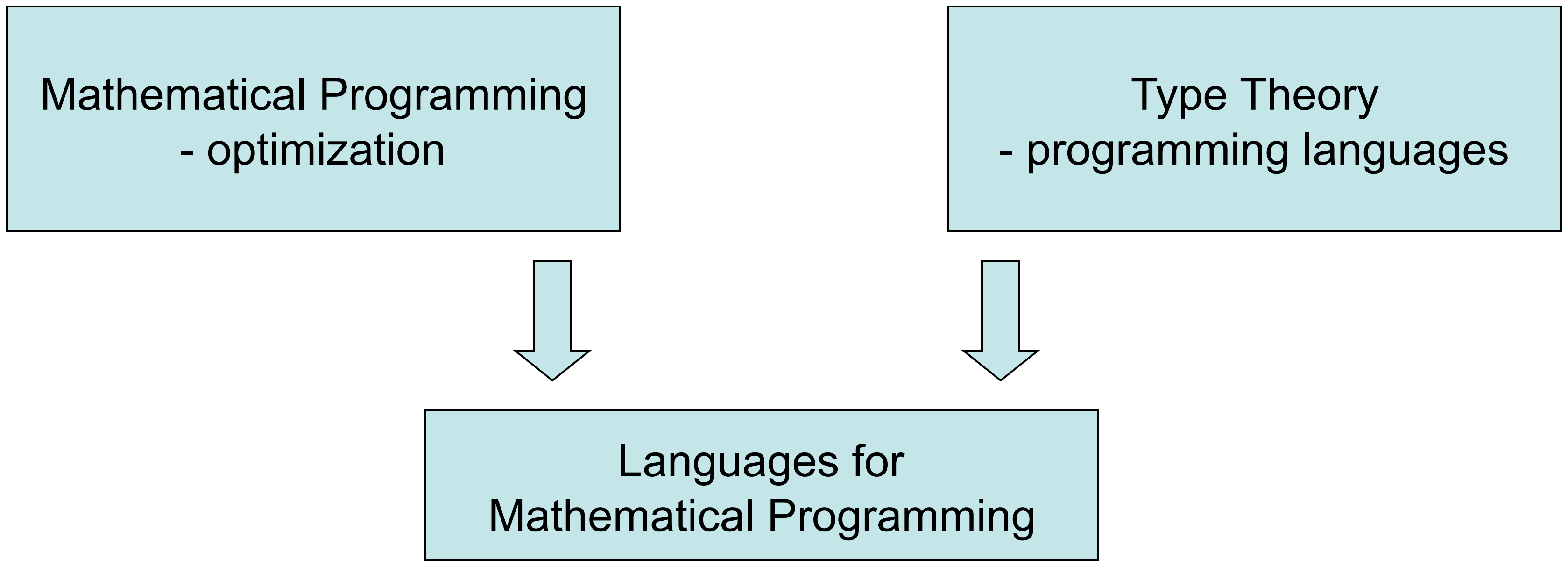
- Conceptual Description
  - statements in a natural language
  - pictures, diagrams
- Formal Model
  - statements in a mathematical language



- Modeling is the process of translating the conceptual description of a physical system into mathematical statements
- Affected by mathematical language we choose to express the model in

Goals

- Use type theory to develop more elegant modeling languages
- Define model transformations
  - to connect to existing algorithms



Outline

- Introductory example
- Review previous definitions of MP

- Define MP (using type theory)
- Convert MP to MIP
- Language for indexing
- Indexed MP
- Hybrid systems modeling (quick overview)

- Conclusions

Introductory Example

- Conceptual Description
  A reactor can run one of three reactions. If rxnX is run, then the temperature must stay below 400. If rxnZ is run, the temperature must be below 600, and the maximum temperature the reactor can handle is 1000.

- Mathematical Program (MP)
  - real, integer, bool
  - conjunction and disjunction of (in)equations, Boolean constraints

$$x, y, z \in \texttt{bool}$$
$$\Theta \in \texttt{real}$$
$$x \veebar y \veebar z$$
$$0 \leq \Theta \leq 1000$$
$$\begin{bmatrix} x \\ \Theta \leq 400 \end{bmatrix} \vee [y] \vee \begin{bmatrix} z \\ \Theta \leq 600 \end{bmatrix}$$

- Mixed-Integer Program (MIP)
  - real, integer
  - conjunction of (in)equations

$$\begin{aligned} x, y, z &\in \{0, 1\} \\ \Theta_x, \Theta_y, \Theta_z &\in \texttt{real} \\ \Theta &\in \texttt{real} \\ x + y + z &= 1 \\ \Theta &= \Theta_x + \Theta_y + \Theta_z \\ 0 \leq \Theta_x &\leq 400x \\ 0 \leq \Theta_y &\leq 1000y \\ 0 \leq \Theta_z &\leq 600z \end{aligned}$$

Outline

- Introductory example
- Review previous definitions of MP

- Define MP (using type theory)
- Convert MP to MIP
- Language for indexing
- Indexed MP
- Hybrid systems modeling (quick overview)

- Conclusions

Matrix-Based Definition of Linear MIP ☺

$$\min \left\{ c^T x : Ax \leq b, x^1 \in \mathbb{Z}^n, x^2 \in \mathbb{R}^p \right\}$$

- This defines a set of mathematical programs
  - each particular choice of c, x, A, and b refers to an element of this set
- Advantages
  - compact
  - clearly captures numerical aspects of the problem
  - all previous results made possible by this definition

Matrix-Based Definition of Linear MIP ☹

$$\min \left\{ c^T x : Ax \leq b, x^1 \in \mathbb{Z}^n, x^2 \in \mathbb{R}^p \right\}$$

- Disadvantages
  1. cannot actually model systems in this form
  2. problem structure cannot be expressed
  3. not extensible
  4. constraints are not objects of the formal theory
  5. does not lead to software implementation
  6. does not specify information desired from a solution
  7. many more …
     - no way to state variable bounds
     - are Z and R different types
     - computation on the reals is not possible

} addressed

} not addressed

1. Models cannot be posed in matrix form (in practice)

- The matrix form does not provide a language expressive in practice
- We would like to write

  $$x_i = x_{i-1} + y_j \quad \forall i \in I, \forall j \in J_i$$

  which involves
  - universal quantifiers
  - index sets
  - functions returning index sets
  - function variables
- Must define a mathematical system supporting these features

Critique of Current Definition of MP

2. Knowledge of problem structure not retained

$$x_i = x_{i-1} + y_j \quad \forall i \in I, \forall j \in J_i$$

- Eliminating indices discards valuable knowledge
  - indexed constraint: there are N constraints of an identical form
  - expanded constraint: there are N unrelated constraints
- How many constraints are there?
  - current answer: $N = \sum_{i \in I} |J_i|$
  - our answer: it can be $N = 1$

Critique of Current Definition of MP

3. Not extensible

- Matrices do not accommodate nonlinear constraints
- There is no way to give a matrix based definition of
  - Boolean expressions
  - finite domain terms
  - …
- Thus, matrix-based definition is theoretically limiting.

Critique of Current Definition of MP

4. Constraints are not mathematical objects

- If they were, it would be possible to define a function taking a constraint as an argument, e.g.

$$g(Ax \leq b)$$

- Actually, we define

$$g(A,b)$$

Interpretation of A and b as relating to a constraint is external to the theory in which g is defined.

- Program transformations require operating on constraints
  - convex hull
  - big-M
- Must define a constraint space

Outline

- Introductory example
- Review previous definitions of MP

- Define MP (using type theory)
- Convert MP to MIP
- Language for indexing
- Indexed MP
- Hybrid systems modeling (quick overview)

- Conclusions

Type Theory

- Closely related to set theory and mathematical logic

- Has served as a foundation of mathematics during the last century
  - Whitehead and Russell, *Principia Mathematica*, 1910
  - Martin-Löf, P., *Intuitionistic Type Theory*, 1984
  - Andrews, P. B., *An introduction to mathematical logic and type theory: to truth through proof*, 2002

- Is the mathematical basis for designing programming languages
  - Harper, R., *Programming Languages: Theory and Practice*, Draft 2005
  - Harper, R., *Type Systems for Programming Languages*, Draft 2000
  - Pierce, B. C., *Types and Programming Languages*, 2002

Syntax

- types $\tau ::= \texttt{real} \mid \texttt{bool}$

  $\tau \in \{\texttt{real}, \texttt{bool}\}$

- expressions $e ::= x \mid r \mid \texttt{true} \mid \texttt{false}$

  $\mid -e \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2$

  $\mid \texttt{not}\, e \mid e_1\, \texttt{or}\, e_2 \mid e_1\, \texttt{and}\, e_2$

$$e \in \{x, 0.0, 0.1, \ldots, \texttt{true}, \texttt{false},$$
$$-x, -0.0, \ldots, -\texttt{true}, \ldots,$$
$$2.0 + \texttt{true}, \ldots\}$$

  – invalid expressions included

Type of Expression $\Gamma \vdash e : \tau$

- Need context $\Gamma ::= \emptyset \mid \Gamma, x : \tau$
- Let $\Gamma \vdash e : \tau$ mean e is of type τ in context Γ
- Definition of judgement is given by *inference rules*

$$\begin{array}{ll} \dfrac{\Gamma \vdash e_1 : \texttt{real} \qquad \Gamma \vdash e_2 : \texttt{real}}{\Gamma \vdash e_1 + e_2 : \texttt{real}} & \begin{array}{l} \longrightarrow \text{Preconditions} \\ \\ \longrightarrow \text{Conclusion} \end{array} \end{array}$$

Type of Expression (cont.)

$$\frac{}{\Gamma \vdash x : \tau} \text{ where } (x : \tau) \in \Gamma$$

$$\frac{}{\Gamma \vdash r : \texttt{real}}$$

$$\frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}}$$

$$\frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{real} \qquad \Gamma \vdash e_2 : \texttt{real}}{\Gamma \vdash e_1 + e_2 : \texttt{real}}$$

$$\frac{\{\Gamma \vdash e_j : \texttt{real}\}_{j=1}^{2}}{\Gamma \vdash e_1 - e_2 : \texttt{real}}$$

$$\frac{\{\Gamma \vdash e_j : \texttt{real}\}_{j=1}^{2}}{\Gamma \vdash e_1 * e_2 : \texttt{real}}$$

$$\frac{\Gamma \vdash e : \texttt{bool}}{\Gamma \vdash \texttt{not}\, e : \texttt{bool}}$$

$$\frac{\{\Gamma \vdash e_j : \texttt{bool}\}_{j=1}^{2}}{\Gamma \vdash e_1 \,\texttt{or}\, e_2 : \texttt{bool}}$$

$$\frac{\{\Gamma \vdash e_j : \texttt{bool}\}_{j=1}^{2}}{\Gamma \vdash e_1 \,\texttt{and}\, e_2 : \texttt{bool}}$$

- Expressions have been categorized into types
- Any expression not categorized is ill-formed

Logical Formulation of MP

Syntax (cont.)

- types …
- expressions …
- propositions (constraints)

$$
\begin{aligned}
c ::= &\ \mathtt{T} \mid \mathtt{F} \\
&\mid \mathtt{isTrue}\ e \mid e_1 = e_2 \mid e_1 \leq e_2 \\
&\mid c_1 \vee c_2 \mid c_1 \wedge c_2 \\
&\mid \exists x : \tau \,.\, c
\end{aligned}
$$

  – Boolean conjunction/disjunction are different than propositional conjunction/ disjunction

- programs

$$p ::= \delta_{x_1:\tau_1,\ldots,x_m:\tau_m}\ \{e \mid c\}$$

$$\text{where } \delta ::= \mathtt{min} \mid \mathtt{max}$$

Type System

- Type of Expression $\Gamma \vdash e : \tau$
- Well-Formed Proposition $\Gamma \vdash c \ \mathrm{PROP}$
- Well-Formed Program $p \ \mathrm{MP}$
  - *p* MP defines a set of mathematical programs

Software Example

Input Program

$$\left.\mathtt{min}_{x:\mathtt{real},y:\mathtt{real}} \left\{ x \mid \begin{bmatrix} \mathtt{isTrue}\ y \wedge \\ x \leq 3.0 \end{bmatrix} \vee \begin{bmatrix} \mathtt{isTrue}\ (\mathtt{not}\ y) \wedge \\ x \geq 4.0 \end{bmatrix} \right\} \qquad \right\} p$$

Input Program

```
var x:real
var y:real

min x subject_to
(isTrue y, x <= 3.0) disj (isTrue (not y), x >= 4.0)
```

$$\left.\vphantom{\begin{matrix}a\\a\\a\\a\\a\end{matrix}}\right\} p$$

Software Example (cont.)

Input Program

```
var x:real
var y:real

min x subject_to
(isTrue y, x <= 3.0) disj (isTrue (not y), x >= 4.0)
```

$$\left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} p$$

Output Messages

```
ERROR: type analysis failed
  context: x:real, y:real
  expr at 5.9: y
  type: bool

MSG: ill-formed prop, previous messages should explain why
  context: x:real, y:real
  prop at 5.2-5.9: isTrue y
```

$\Gamma \vdash e : \tau$

$\Gamma \vdash c \text{ PROP}$

$\vdots$

$p \text{ MP}$

Logical Formulation of MP

Semantics

- Thus far, we have defined the syntax of MP
- But we do not know what the syntax means
- Need to
  1. Define how to evaluate an expression
     e.g. must state that 2.0 + 3.1 is equal to 5.1
  2. State what it means for a proposition to be true
     e.g. must state that a $c_1 \vee c_2$ is true if c_1 is true or if c_2 is true
  3. Explain what the solution to a mathematical program is
- Our definition of the semantics elucidates the information desired from an algorithm
  - we want not just the solution but a proof explaining why the given solution is in fact the solution
  - it appears that current practice follows a mixture of the classical and constructive traditions of proof

Outline

- Introductory example
- Review previous definitions of MP

- Define MP (using type theory)
- Convert MP to MIP
- Language for indexing
- Indexed MP
- Hybrid systems modeling (quick overview)

- Conclusions

Compiling MP to MIP

Mixed-Integer Programming Sub-Language

- Types $\tau ::= \texttt{real} \mid \cancel{\texttt{bool}}$

- Expressions $e ::= x \mid r \mid \cancel{\texttt{true}} \mid \cancel{\texttt{false}}$
  $\mid -e \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2$
  $\cancel{\mid \texttt{not}\, e \mid e_1\, \texttt{or}\, e_2 \mid e_1\, \texttt{and}\, e_2}$

- Propositions $c ::= \texttt{T} \mid \texttt{F}$
  $\mid \cancel{\texttt{isTrue}\, e} \mid e_1 = e_2 \mid e_1 \leq e_2$
  $\mid \cancel{c_1 \vee c_2} \mid c_1 \wedge c_2$
  $\mid \exists x : \rho\, .\, c$

- Programs $p ::= \delta_{x_1:\rho_1,\ldots,x_m:\rho_m}\{e \mid c\}$

Overview of Compiler

- Type Compiler $\tau \xmapsto{\text{TYPE}} \tau^{\text{MIP}}$
- Expression Compiler $e \xmapsto{\text{DLF}} e^{\text{MIP}}$ $e \xmapsto{\text{CONJ}} c^{\text{MIP}}$
- Proposition Compiler $\Upsilon \vdash c \xmapsto{\text{PROP}} c^{\text{MIP}}$ $\Upsilon \vdash c \xmapsto{\text{DISJ}} c^{\text{MIP}}$
- Program Compiler $p \xmapsto{\text{PROG}} p^{\text{MIP}}$

Balas (1974)

Raman and Grossmann (1994)

- Precondition $p\ \textsc{DisjVarsBounded}$

Disjunctive Constraint Compiler $\Upsilon \vdash c \overset{\text{DISJ}}{\longmapsto} c^{\text{MIP}}$

- $\Upsilon \vdash c \multimap c'$ adds bounding constraints to c
- $e \circledast e_1 \hookrightarrow e_2$ multiplies e to all constant terms to in e_1
- $e \circledast c_1 \hookrightarrow c_2$ is analogous judgement for propositions

Then, the definition of $\Upsilon \vdash c \overset{\text{DISJ}}{\longmapsto} c^{\text{MIP}}$ is

$$\frac{\begin{array}{c}\left\{\Upsilon \vdash c_j \overset{\text{PROP}}{\longmapsto} c'_j\right\}_{j\in\{A,B\}} \qquad \Upsilon \overset{\text{CTXT}}{\longmapsto} \Upsilon' \\ \left\{\Upsilon' \vdash c'_j \multimap c''_j\right\}_{j\in\{A,B\}} \qquad \left\{y^j \circledast \{\vec{x}^j/\vec{x}\}\, c''_j \hookrightarrow c'''_j\right\}_{j\in\{A,B\}}\end{array}}{\Upsilon \vdash (c_A \vee c_B) \overset{\text{DISJ}}{\longmapsto} \left(\begin{array}{l} \exists \vec{x}^A : \vec{\rho}\,.\, \exists \vec{x}^B : \vec{\rho}\,.\, \exists y^A : [0,1]\,.\, \exists y^B : [0,1]\,. \\ \qquad \left(\vec{x} = \vec{x}^A + \vec{x}^B\right) \wedge \left(y^A + y^B = 1\right) \wedge \left(c'''_A \wedge c'''_B\right)\end{array}\right)}$$

Example

Input File (MP)

```
var x:real
var w:real

min x + w subject_to
(x <= w) disj (x >= w + 4.0)
```

Output

```
ERROR: variable in disjunct must be
   bounded
  variable at 5.7: w
  is of unbounded type at 2.7-2.10:
   real

ERROR: variable in disjunct must be
   bounded
  variable at 5.2: x
  is of unbounded type at 1.7-1.10:
   real
```

Compiling MP to MIP

Example

Input File (MP)

```
var x:<10.0, 100.0>
var w:<2.0, 50.0>

min x + w subject_to
(x <= w) disj (x >= w + 4.0)
```

Output

```
var x:<10.0, 100.0>
var w:<2.0, 50.0>

min x + w subject_to

exists y1:[0, 1]
exists y2:[0, 1]
exists x1:<10.0, 100.0>
exists x2:<10.0, 100.0>
exists w1:<2.0, 50.0>
exists w2:<2.0, 50.0>
  w = w1 + w2,
  x = x1 + x2,
  y1 + y2 = 1,

  10.0 * y1 <= x1,
  x1 <= 100.0 * y1,
  2.0 * y1 <= w1,
  w1 <= 50.0 * y1,
  x1 <= w1,

  10.0 * y2 <= x2,
  x2 <= 100.0 * y2,
  2.0 * y2 <= w2,
  w2 <= 50.0 * y2,
  x2 >= w2 + 4.0 * y2
```

Outline

- Introductory example
- Review previous definitions of MP

- Define MP (using type theory)
- Convert MP to MIP
- Language for indexing
- Indexed MP
- Hybrid systems modeling (quick overview)

- Conclusions

Syntax: Expressions

$$x_i = x_{i-1} + y_j \quad \forall i \in I, \forall j \in J_i$$

- Expressions $$\begin{aligned} \varepsilon ::= {} & x \mid l \mid k \\ & \mid (\varepsilon_1, \dots, \varepsilon_m) \mid \varepsilon.k \\ & \mid -\varepsilon \mid \varepsilon_1 + \varepsilon_2 \mid \varepsilon_1 - \varepsilon_2 \mid \varepsilon_1 * \varepsilon_2 \\ & \mid \texttt{case}\ \varepsilon\ \texttt{of}\ \{l_j \Rightarrow \varepsilon_j\}_{j=1}^{m} \mid \texttt{case}\ \varepsilon\ \texttt{of}\ \{k_j \Rightarrow \varepsilon_j\}_{j=1}^{m} \end{aligned}$$

- Example: a one dimensional table

$$\texttt{case}\ i\ \texttt{of}\ \text{‘A’} \Rightarrow 4 \mid \text{‘B’} \Rightarrow 3 \mid \text{‘C’} \Rightarrow j + 2$$

  – if *i* takes the value ‘A’, then the whole expression is equal to 4

Syntax: Types

$$\begin{aligned}\sigma ::= {} & \{l_1, \dots, l_m\} \mid [\varepsilon_L, \varepsilon_U] \mid x_1 : \sigma_1 \times \cdots \times x_m : \sigma_m \\ & \mid \texttt{case}\ \varepsilon\ \texttt{of}\ \{l_j \Rightarrow \sigma_j\}_{j=1}^{m} \mid \texttt{case}\ \varepsilon\ \texttt{of}\ \{k_j \Rightarrow \sigma_j\}_{j=1}^{m} \\ & \mid \lambda x \mathbin{.} \sigma \mid \sigma\, \varepsilon \\ & \mid \sigma :: \kappa\end{aligned}$$

- Integer interval $[\varepsilon_L, \varepsilon_U]$ is a *dependent* type
  - unlike real and bool, its elements depend on the values of variables
- $x_1{:}\sigma_1 \times \ldots \times x_m{:}\sigma_m$ is a dependent product, a generalization of the Cartesian product
  - ex.

$$[1,2] \times [1,3] = \{(1,1), (1,2), (1,3), (2,1), (2,2), (2,3)\}$$

$$(x : [1,2]) \times [1,x] = \{(1,1), (2,1), (2,2)\}$$

Syntax: Types (cont.)

- Example. Let *f* be the function

$$\lambda x\,\texttt{.}\,\texttt{case}\ x\ \texttt{of}\ 1 \Rightarrow \{\text{‘A’, ‘B’}\} \mid 2 \Rightarrow [1, 10]$$

Then

$$f(1) = \{\text{‘A’, ‘B’}\}$$
$$f(2) = [1, 10]$$

Syntax: Kinds

$$\kappa ::= \texttt{IndexSet} \mid x : \sigma \Rightarrow \kappa$$

- $\texttt{IndexSet}$ is set of all index sets
- $x : \sigma \Rightarrow \kappa$ is set of all functions mapping σ to κ

Type System

- Main judgements

$$\Delta \vdash \kappa \ \text{KIND} \qquad \Delta \vdash \sigma :: \kappa \qquad \Delta \vdash \varepsilon : \sigma$$

- Example

$$\emptyset \vdash \underbrace{x : [1,10] \times [x,10]}_{\text{this is an index set}} :: \texttt{IndexSet} \qquad ✓$$

$$\emptyset \vdash \underbrace{x : [1,10] \times [x,5]}_{\text{this is not an index set}} :: \texttt{IndexSet} \qquad ✗$$

Type System (cont.)

- Type system is semantically defined
  - constructs declared to exist precisely when they have meaning
- This is easier
- Leads to more programs being considered well-formed
- Possible only because index types are finite

Example

```
let
  set JOBS = {'a','b','c'}

  typei RUNS_ON = fni j . case j of
                              'a' => {'s1','s2'}
                            | 'b' => {'s1','s3','s4'}
                            | 'c' => {'s3','s4'}
in
  j:JOBS * RUNS_ON[j]
end
```

σ

- Explicitly, above set is

```
        {('a','s1'), ('a','s2'),
         ('b','s1'), ('b','s3'), ('b','s4'),
         ('c','s3'), ('c','s4')}
```

Outline

- Introductory example
- Review previous definitions of MP

- Define MP (using type theory)
- Convert MP to MIP
- Language for indexing
- Indexed MP
- Hybrid systems modeling (quick overview)

- Conclusions

Syntax

- Types $\tau ::= \texttt{real} \mid \texttt{bool} \mid i : \sigma \to \tau$

- Expressions

$$
\begin{aligned}
e ::= {} & x \mid r \mid \texttt{true} \mid \texttt{false} \\
& \mid -e \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \\
& \mid \texttt{not}\, e \mid e_1 \,\texttt{or}\, e_2 \mid e_1 \,\texttt{and}\, e_2 \\
& \mid \sum_{i:\sigma} e \\
& \mid \texttt{case}_{i.\tau}\ \varepsilon\ \texttt{of}\ \{l_j \Rightarrow e_j\}_{j=1}^{m} \mid \texttt{case}_{i.\tau}\ \varepsilon\ \texttt{of}\ \{k_j \Rightarrow e_j\}_{j=1}^{m} \\
& \mid \lambda i : \sigma \,.\, e \mid e\,\varepsilon \\
& \mid e : \tau
\end{aligned}
$$

Example

- The expression

$$\lambda i : \{\text{‘A’},\ \text{‘B’}\}\,\texttt{.}\,\texttt{case}\ i\ \texttt{of}\ \begin{array}{l} \quad \text{‘A’} \Rightarrow 2.0 \\ \mid \text{‘B’} \Rightarrow 1.0 + 3.4 \end{array}$$

  is of type

$$\{\text{‘A’}, \text{‘B’}\} \rightarrow \texttt{real}$$

Syntax (cont.)

- Types …
- Expressions …
- Propositions

$$\begin{aligned} c ::= {} & \texttt{T} \mid \texttt{F} \\ & \mid \texttt{isTrue}\; e \mid e_1 = e_2 \mid e_1 \leq e_2 \\ & \mid c_1 \vee c_2 \mid c_1 \wedge c_2 \\ & \mid \exists x : \tau \,.\, c \\ & \mid \bigvee_{i:\sigma} c \mid \bigwedge_{i:\sigma} c \\ & \mid \texttt{case}_{i.\zeta}\; \varepsilon \;\texttt{of}\; \{l_j \Rightarrow c_j\}_{j=1}^{m} \mid \texttt{case}_{i.\zeta}\; \varepsilon \;\texttt{of}\; \{k_j \Rightarrow c_j\}_{j=1}^{m} \\ & \mid \lambda i : \sigma \,.\, c \mid c\, \varepsilon \\ & \mid c : \zeta \end{aligned}$$

- Propositional Types

$$\zeta ::= \texttt{Prop} \mid i : \sigma \to \zeta$$

Examples

- The proposition $x_1 + x_2 = 4.0 * x_3$

  is of propositional type $\texttt{Prop}$

- The proposition

$$\lambda i : \{\text{‘A’}, \text{‘B’}\}\,\texttt{.}\,\texttt{case}\ i\ \texttt{of}$$
$$\text{‘A’} \Rightarrow (x_1 = 3)$$
$$\mid \text{‘B’} \Rightarrow (x_1 = x_2 + x_3)$$

  is of propositional type $\{\text{‘A’}, \text{‘B’}\} \rightarrow \texttt{Prop}$

Type System

Well-Formed Type	$\vdash_\Delta \tau \ \textsc{type}$
Type of Expression	$\Gamma \vdash_\Delta e : \tau$
Well-Formed Propositional Type	$\vdash_\Delta \zeta \ \textsc{prop\_type}$
Type of Proposition	$\Gamma \vdash_\Delta c : \zeta$
Well-Formed Program	$p \ \textsc{mp}$

Overview of Compiler

- Type Compiler $\tau \xmapsto{\text{TYPE}} \tau^{\text{MIP}}$
- Refined Type Compiler $\rho \xmapsto{\text{RTYPE}} \rho^{\text{MIP}}$
- Expression Compiler $e \xmapsto{\text{DLF}} e^{\text{MIP}}$ $e \xmapsto{\text{CONJ}} c^{\text{MIP}}$
- Proposition Compiler $\Upsilon \vdash c \xmapsto{\text{PROP}} c^{\text{MIP}}$ $\Upsilon \vdash c \xmapsto{\text{DISJ}} c^{\text{MIP}}$
- Program Compiler $p \xmapsto{\text{PROG}} p^{\text{MIP}}$

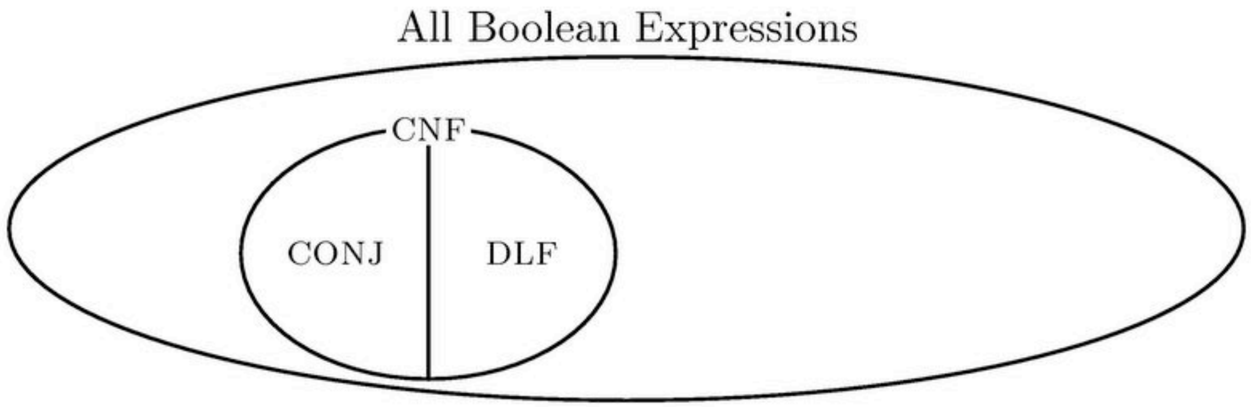
- Our compilers retain indexing structure

Normal Forms in Presence of Higher Types

Let f be the function

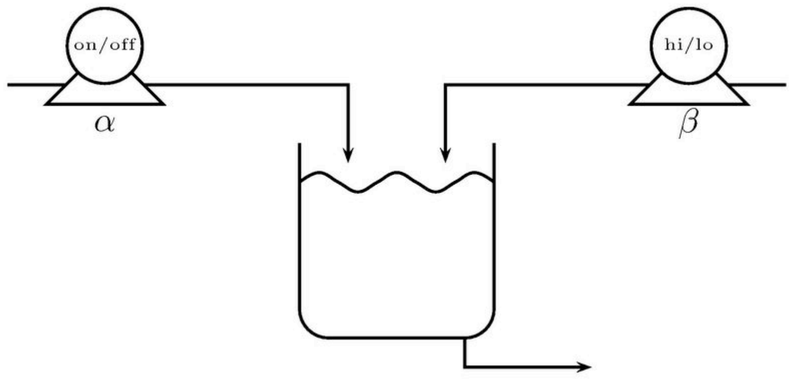
$$\lambda i : \sigma \,\texttt{.}\, \texttt{case}_{i\texttt{.}\tau}\ i\ \texttt{of}\ \text{‘A’} \Rightarrow x \mid \text{‘B’} \Rightarrow (x\ \texttt{and}\ z)\ \texttt{or}\ y$$

- Is f(‘A’) in CNF? Yes.
- Is f(‘B’) in CNF? No.
- Is $f(\varepsilon)$ in CNF?



Switched Flow Process

Conceptual Description



- Two pumps – hybrid processes
  - on/off
  - hi/lo
- Want to minimize cost
  - continuous operational costs
  - discrete switching costs

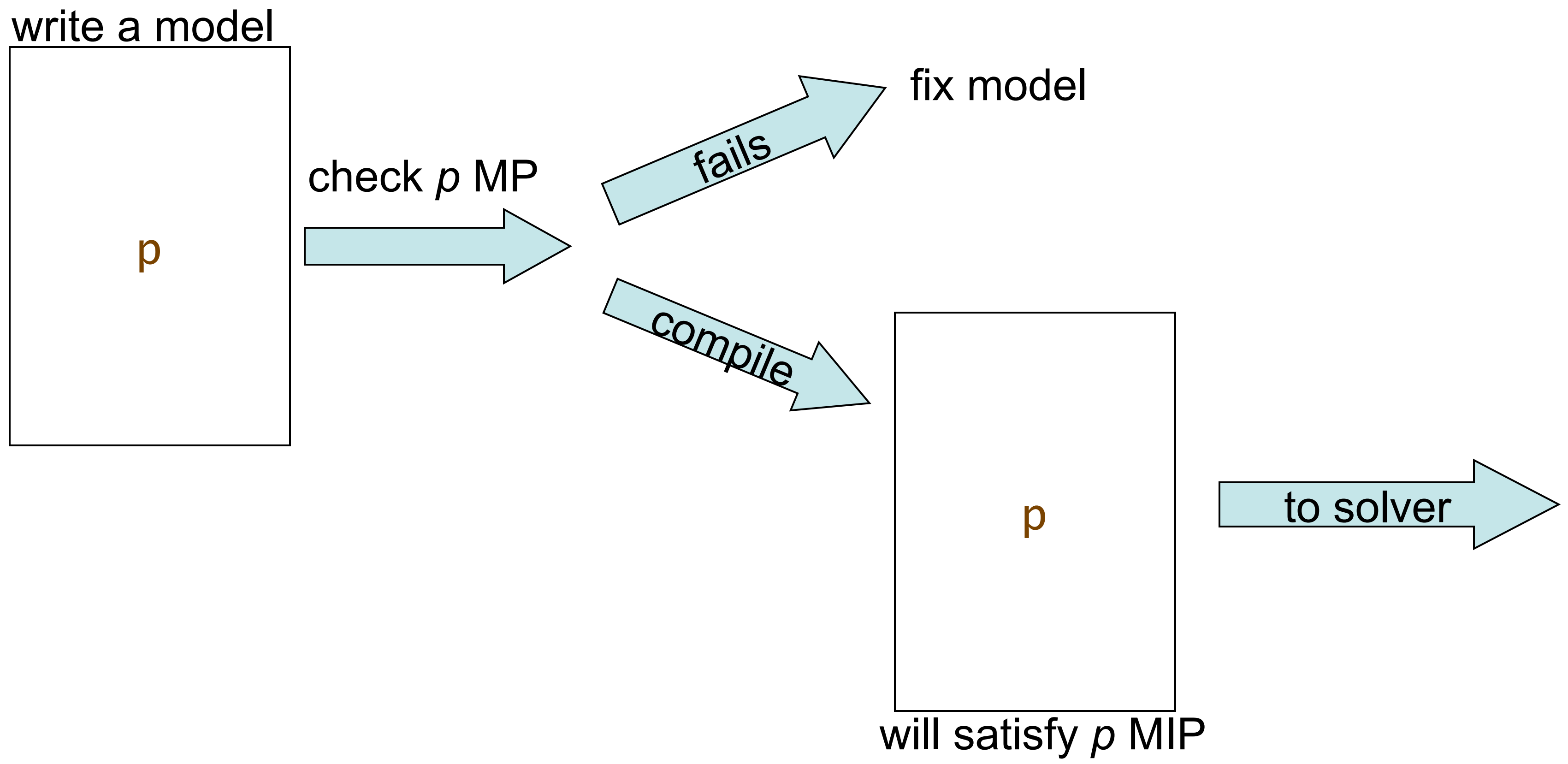
The figure depicts a tank being filled by two hybrid processes, α and β, and being emptied continuously at a rate of F=1.8. Initially, the material level in the tank is M_0=20.0. The tank's maximum capacity M^{max}=150.0 and the material level should never fall below M^{min}=10.0.

Process α represents a pump that can be either on or off. When it is on, it provides material to the tank at rate 2.0. There is also an operating cost of 10.0 per unit time for running the pump. Operational constraints on the pump forbid it from being continuously run longer than 30.0 time units; it must be switched off before this time limit is reached. There are no operating costs while it is off, but it must not be switched on again in less than 2.0 time units. When it is switched on again, if at all, a startup cost of 50.0 is incurred.

Process β is similar, but it represents a pump that is always on, either at a high or low setting. In the high setting, material flows to the tank at rate 4.0 and the operating cost is 15.0 per time unit. In the low setting, the material flow rate drops to 0.5, and the operating cost is 2.0. Once set to low, the pump cannot be switched to the high setting again for at least 3.0 time units, and, when it does, a startup cost of 40.0 is incurred.

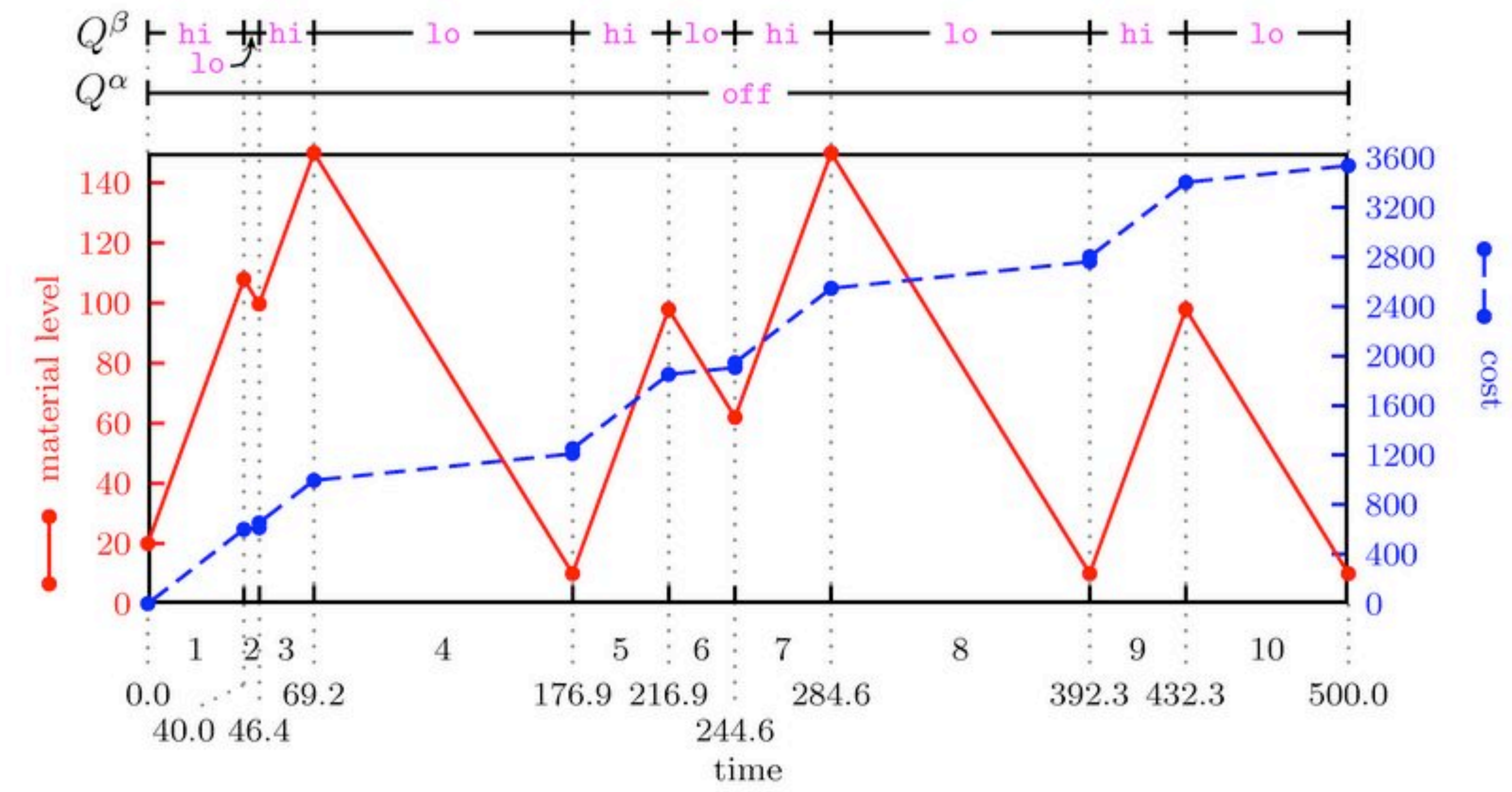
We wish to study how the material level changes over time and to understand the cost of running the system for T^{max}=500.0 time units.

Software Usage



Switched Flow Process

Optimal Solution



Outline

- Introductory example
- Review previous definitions of MP

- Define MP (using type theory)
- Convert MP to MIP
- Language for indexing
- Indexed MP
- Hybrid systems modeling (quick overview)

- Conclusions

Linear Coupled Component Automata

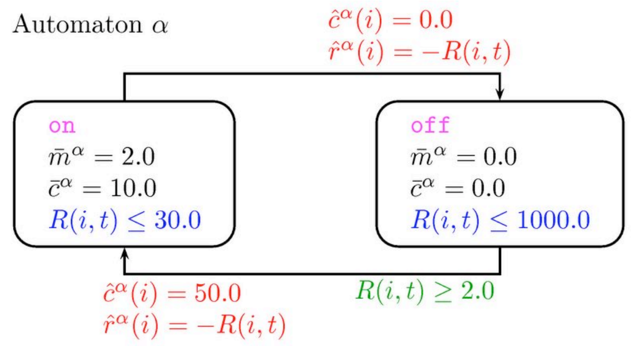
- The switched flow process is a *dynamical* discrete-continuous system, or hybrid system
- Can provide more features than in Indexed MP language
- Several researchers have provided hybrid automata frameworks
  - Cassez and Larsen (2000), Alur et al. (1995)
- We have defined the LCCA framework

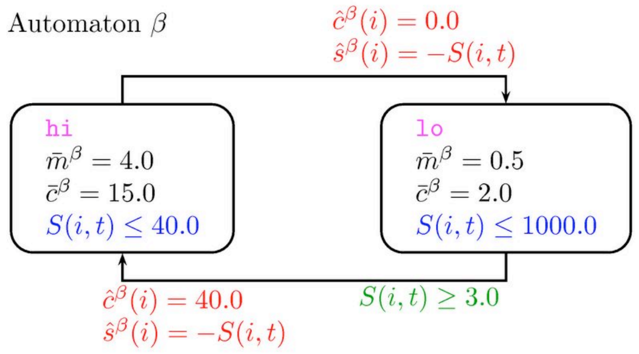
Switched Flow Process

LCCA Model

$$n = 10$$

$$t_n^e = T^{\max}$$
$$t_1^s = 0.0$$





$$\begin{aligned}
\frac{\mathrm{d}R(i,t)}{\mathrm{d}t} &= 1.0 && \forall (i,t) \in \mathbb{T} \\
R(i+1,t) &= R(i,t) + \hat{r}^{\alpha}(i) && \forall i \in \mathbb{N}\setminus\{n\} \\
R(1,t_1^s) &= 0.0
\end{aligned}$$

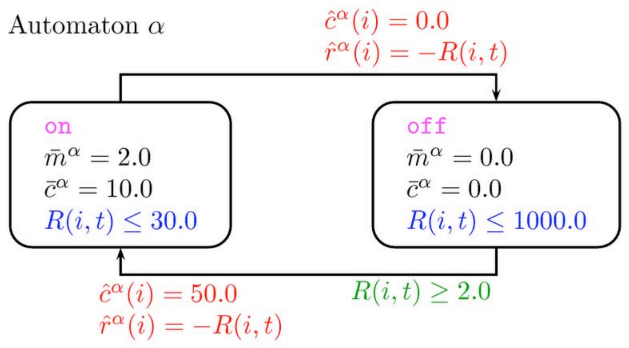
$$\begin{aligned}
\frac{\mathrm{d}S(i,t)}{\mathrm{d}t} &= 1.0 && \forall (i,t) \in \mathbb{T} \\
S(i+1,t) &= S(i,t) + \hat{s}^{\beta}(i) && \forall i \in \mathbb{N}\setminus\{n\} \\
S(1,t_1^s) &= 0.0
\end{aligned}$$

$$\begin{aligned}
\frac{\mathrm{d}M(i,t)}{\mathrm{d}t} &= \bar{m}^{\alpha}\left(Q^{\alpha}(i)\right) + \bar{m}^{\beta}\left(Q^{\beta}(i)\right) - F^{\text{out}} && \forall (i,t) \in \mathbb{T} \\
M(i+1,t) &= M(i,t) && \forall i \in \mathbb{N}\setminus\{n\} \\
M(1,t_1^s) &= M_0 \\
M^{\min} \leq M(i,t) &\leq M^{\max} && \forall (i,t) \in \mathbb{T}
\end{aligned}$$

$$\begin{aligned}
\frac{\mathrm{d}C(i,t)}{\mathrm{d}t} &= \bar{c}^{\alpha}\left(Q^{\alpha}(i)\right) + \bar{c}^{\beta}\left(Q^{\beta}(i)\right) && \forall (i,t) \in \mathbb{T} \\
C(i+1,t) &= C(i,t) + \hat{c}^{\alpha}(i) + \hat{c}^{\beta}(i) && \forall i \in \mathbb{N}\setminus\{n\} \\
C(1,t_1^s) &= 0.0
\end{aligned}$$

Switched Flow Process

Transform LCCA to MP



disjunction over modes

$$\begin{bmatrix} Q(i,t) = \texttt{on} \\ R(i,t) \leq 3.0 \end{bmatrix} \vee \begin{bmatrix} Q(i,t) = \texttt{off} \\ R(i,t) \leq 1000.0 \end{bmatrix}$$

disjunction over transitions

$$\begin{bmatrix} Q(i,t) = \texttt{on} \\ Q(i+1,t) = \texttt{off} \\ \hat{c}^\alpha(i) = 0.0 \\ \hat{r}^\alpha(i) = -R(i,t) \end{bmatrix} \vee \begin{bmatrix} Q(i,t) = \texttt{off} \\ Q(i+1,t) = \texttt{on} \\ R(i,t) \geq 2.0 \\ \hat{c}^\alpha(i) = 50.0 \\ \hat{r}^\alpha(i) = -R(i,t) \end{bmatrix}$$

Outline

- Introductory example
- Review previous definitions of MP

- Define MP (using type theory)
- Convert MP to MIP
- Language for indexing
- Indexed MP
- Hybrid systems modeling (quick overview)

- Conclusions

Comparison of MP Languages

Language Feature	GAMS	AMPL	OPL[1]	Mosel[2]	TyL
Boolean expressions				✓	✓
disjunctive constraints			✓	✓	✓
index type functions		✓	✓		✓
index set operations		✓	✓	✓	
expression functions			✓	✓	✓
proposition functions					✓
functions returning tuples					✓
dependent products		✓	✓		✓
dependent functions					✓
case expressions		✓	✓	✓	✓
index case types		✓			✓
case propositions					✓
conditional index sets	✓	✓	✓		
syntax checker	✓	✓	✓	✓	✓
type system					✓
Boolean to IP compiler					✓
disjunction to MIP compiler			✓		✓

[1] Based on OPL Studio 3.7.1. Version 4.0 has reduced features.

[2] Including XPress-Kalis.

Comparison of Mathematical Definitions of MP

	Definition Style	
	Matrix\Numerical	Type Theory\Logical
simple compact definition	✓	
numerical methods applicable	✓	✓
knowledge retention mechanisms		✓
extensible		✓
non-numerical operations		✓
practical modeling system		✓
varied algorithms easily employed		✓
leads to computer implementation		✓

Conclusions

- Formal theory of indexed mathematical programs provided
  - programs can be written in this form
  - problem structure retained for algorithms
- Defined a compiler automatically generating mixed-integer programs
  - most algorithms expect this form
  - indexing structure is retained
- Hybrid systems
  - provided novel modeling framework
  - can transform these models to MP
- Demonstrated the value of type theory to new areas

Directions for Future Research

- Connect TyL to solvers
- Enhance language with differential equations
  - define transformation to algebraic equations
- Define a logic for hybrid systems
- Invent domain specific mathematical languages
  - types
    molecule, atom, etc.
  - expressions
    MW(m) where m : molecule, and MW : molecule $\rightarrow$ real