Typesetting Conventions for Mathematical Formulations of CP Optimizer Models

Philippe Laborie
IBM, 9 rue de Verdun 94250 Gentilly, France
laborie@fr.ibm.com

1 Introduction

This document proposes some conventions for typesetting mathematical formulations of combinatorial optimization problems for CP Optimizer. It extends the common practice used for typesetting MIP models to the mathematical concepts and constraints introduced in CP Optimizer (interval, sequence variables, functions). Following these conventions ensures a non-ambiguous description of the mathematical formulation of the problem with a one-to-one correspondence with the implementation of the model whatever API of CP Optimizer is used (C++, Python, Java, OPL, CPO file format).

The formal semantics of the concepts of CP Optimizer models is defined in the CP Optimizer Reference Manual [?]. A summary of these concepts is available in [?], Section 3.

2 General constructs

2.1 Scopes

A scope is a set of tuples used for indexing. For describing a scope, we use the classical set notation. For instance if N and M are two integers:

- $i \in [1..N]$ is the set of all integers $(i) \in \{1, 2, ..., N\}$
- $i \in [1..N], j \in [1..M]$ is the cartesian product $(i, j) \in [1..N] \times [1..M]$
- $i, j \in [1..N] \mid i \neq j$ is the subset of the cartesian product $[1..N] \times [1..N]$ such that $i \neq j$

When the order of the elements in the scope is important (like for instance in the vectors or matrices below), the tuples are supposed to be generated by iterating on the indexes from left to right. For instance $i, j \in [1..3] \mid i \neq j$ will generate the ordered set of tuples $(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)$.

2.2 Vectors

Vectors are denoted $[v_i]_{SCOPE(i)}$. For instance:

$$[x_{ij}]_{i,j \in [1..N] | i \neq j}$$

2.3 Matrices

Matrices are denoted $[v_{ij}]_{SCOPE(i);SCOPE(j)}$. Note the use of semicolon ";" instead of the comma "," between the scopes. This permits to differentiate:

- $[x_{ij}]_{i \in [1..2],\ j \in [1..2]}$ which is vector $\begin{bmatrix} x_{11} & x_{12} & x_{21} & x_{22} \end{bmatrix}$
- $[x_{ij}]_{i \in [1..2];\ j \in [1..2]}$ which is matrix $\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$

2.4 Decision variables

CP Optimizer provides different types of decision variables (integer, interval, sequence, state function). The general syntax for defining a decision variable is:

$$\texttt{TYPE} \quad \texttt{NAME} \quad \texttt{DOMAIN} \qquad \forall\, \texttt{SCOPE}$$

As a rule of thumb, we denote decision variables with lower cases whereas constants of the problem (including known stepwise or piecewise linear functions) are denoted with upper cases.

Integer variables

The keyword for the type of an integer variable is "integer". The domain is a set of integers. So for instance we can have:

$$\begin{array}{lr} \text{integer } x \in [1..N] & \\ \text{integer } y_i \in \{2k+1\}_{k \in [0..M)} & \forall i \in [1..N] \\ \text{integer } z_{ij} \in \{0,1\} & \forall i \in [1..N], j \in [1..N] \mid i \neq j \end{array}$$

Interval variables

The keyword for the type of an interval variable is "interval". The domain is a subset of domain specifiers in the following order, separated by a comma:

- Minimal start (R) and maximal end (D) range: "$\subset [R..D]$" (default: $\subset [0..+\infty]$)
- Specification whether the interval is optional: "optional" (default: not optional that is, interval is present)

- Value or range for the interval size: "size $=$ S" or "size $\in$ $[S1..S2]$" (default: size $\in [0..+\infty]$)
- Intensity function (stepwise function F): "intensity $=$ F" (default: no intensity)

For instance:

$$\begin{array}{ll}
\text{interval } x & \\
\text{interval } y_i \subset [-H..H] & \forall i \in [1..N] \\
\text{interval } z_{ij} \subset [0..H],\ \text{optional},\ \text{size} \in [A_i..B_i],\ \text{intensity} = F_j & \forall i \in [1..N], j \in [1..M]
\end{array}$$

Sequence variables

The keyword for the type of a sequence variable is "sequence". The domain is a vector X of interval variables (see section Vector above). Optionally, the domain can specify a vector of integer types T (T must have the same dimension as X). For instance:

$$\begin{array}{ll}
\text{sequence } s_i \text{ on } [x_{ij}]_{j \in [1..M]} & \forall i \in [1..N] \\
\text{sequence } s_i \text{ on } [x_{ij}]_{j \in [1..M]},\ \text{types } [T_{ij}]_{j \in [1..M]} & \forall i \in [1..N]
\end{array}$$

State functions

The keyword for the type of a state function is "stateFunction". The domain of a state function can specify a matrix of integers as transition distance between the states. For instance:

$$\begin{array}{ll}
\text{stateFunction } f_k & \forall k \in [1..M] \\
\text{stateFunction } g_k \text{ with } [D_{ij}]_{i \in [0..S); j \in [0..S)} & \forall k \in [1..M]
\end{array}$$

2.5 Constraints

Constraints are defined as follows:

$$\texttt{CONSTRAINT} \quad \forall\ \texttt{SCOPE}$$

The signature of the different constraints available in CP Optimizer is summarized in Appendix A, Table 3.

For instance:

$$\begin{array}{ll}
v_i \leq v_j & \forall\ i, j \in [1..N] \mid i < j \\
\text{endBeforeStart}(x_i, x_j) & \forall\ i, j \in [1..N] \mid i < j
\end{array}$$

2.6 Expressions

The signature of the different constraints available in CP Optimizer is summarized in Appendix A, Table 2. Expressions can be defined directly in the constraints they are used in (case 1) or as separate definitions (case 2). The second case is particularly useful when a given expression is used in several constraints.

Two examples of case 1:

$$\sum_{i \in [1..N]} R_i x_i \leq D$$
$$\sum_{i \in [1..N]} \text{pulse}(y_i, Q_i) \leq C$$

Equivalent examples using case 2:

$$\begin{aligned} u &= \sum_{i \in [1..n]} R_i x_i \\ f &= \sum_{i \in [1..n]} \text{pulse}(y_i, Q_i) \\ u &\leq D \\ f &\leq C \end{aligned}$$

Blackbox functions are first declared as follow (by default, the dimension D, that is the size of the returned vector, is 1):

$$\begin{aligned} &\text{blackbox } \ FUNCTION \\ &\text{blackbox } \ FUNCTION \ \text{ dim } \ D \end{aligned}$$

For instanc, if $f(x)[0]$ is the average of x, $f(x)[1]$ is the standard deviation of vector of decision variables x:

$$\begin{array}{ll} \text{blackbox } \ f \ \text{ dim } \ 2 & \\ \text{integer } \ x[i] & \forall i \in [1..N] \\ stats = f(x) & \\ \text{max} \quad stats[0] + stats[1] & \end{array}$$

3 Examples

3.1 Job-shop scheduling problem

Here is a CP Optimizer formulation of the classical job-shop scheduling problem with N jobs and M machines. The j^{th} operation of the i^{th} job, represented by interval variable x_{ij}, requires machine M_{ij} and has a processing time of P_{ij}.

$$\begin{aligned}
&\min \quad \max_{i\in[1..N]} \text{endOf}(x_{iM}) && && (1)\\
&\text{noOverlap}([x_{ij}]_{i,j\in[1..N]\times[1..M]:MC_{ij}=k}) && \forall k \in [1..M] && (2)\\
&\text{endBeforeStart}(x_{ij-1}, x_{ij}) && \forall i \in [1..N], j \in [2..M] && (3)\\
&\text{interval } \ x_{ij}, \ \text{size} = PT_{ij} && \forall i \in [1..N], j \in [1..M] && (4)
\end{aligned}$$

3.2 Extended flexible job-shop scheduling problem

Here is a CP Optimizer formulation of the scheduling problem described in [?]. Let $V = \{1, 2, ..., o\}$ be the set of all operations. For each operation $i(i = 1, ..., o)$, let $F_i \subseteq (1, 2, ..., m)$, where $F_i \neq \emptyset$, be the subset of machines that can process operation i and let $P_{ik}(i = 1, ..., o, k \in F_i)$ be the corresponding processing times. Furthermore, let A be a set of pairs (i, j) with $i, j \in \{1, ..., o\}$ such that, if (i, j) belongs to A, this means that operation i precedes operation j, i.e. operation j cannot start to be processed until operation i ends to be processed. The problem consists in assigning each operation i to a machine $k \in F_i$ and to determine a starting processing time s_i such that precedences are satisfied. A machine can not process more than an operation at a time and preemption is not allowed. The objective is to minimize the makespan, i.e. the completion time of the last operation.

$$\begin{aligned}
&\min \quad \max_{i\in V} \text{endOf}(x_i) && && (1)\\
&\text{noOverlap}([y_{ik}]_{i\in V|k\in F_i}) && \forall k \in [1..M] && (2)\\
&\text{alternative}(x_i, [y_{ik}]_{k\in F_i}) && \forall i \in V && (3)\\
&\text{endBeforeStart}(x_i, x_j) && \forall (i, j) \in A && (4)\\
&\text{interval } \ x_i && \forall i \in V &&\\
&\text{interval } \ y_{ik}, \ \text{optional}, \ \text{size} = P_{ik} && \forall i \in V, k \in F_i &&
\end{aligned}$$

3.3 Resource-constrained project scheduling problem

Here is a CP Optimizer formulation of the classical RCPSP with N tasks and M resources. Task i has a processing time of PT_i and requires Q_{ik} units of resource k. The capacity of resource j is C_j. The precedence constraints are described as a set of pairs of tasks P.

$$\begin{aligned}
&\min \quad \max_{i\in[1..N]} \text{endOf}(x_i) && && (1)\\
&\sum_{i\in[1..N]} \text{pulse}(x_i, Q_{ik}) \leq C_k && \forall k \in [1..M] && (2)\\
&\text{endBeforeStart}(x_i, x_j) && \forall (i, j) \in P && (3)\\
&\text{interval } \ x_i, \ \text{size} = PT_i && \forall i \in [1..N] && (4)
\end{aligned}$$

3.4 Multi-Mode Resource-constrained project scheduling problem

$$\begin{aligned}
&\min \quad \max_{i\in[1..N]} \text{endOf}(x_i) && && (1)\\
&\text{alternative}(x_i, [y_{ij}]_{j\in M[i]}) && \forall i\in[1..N] && (2)\\
&\sum_{i\in[1..N],j\in M[i]} \text{pulse}(y_{ij}, QR_{ijk}) \le CR_k && \forall k\in[1..R] && (3)\\
&\sum_{i\in[1..N],j\in M[i]} \text{presenceOf}(y_{ij})\cdot QS_{ijk} \le CS_k && \forall k\in[1..S] && (4)\\
&\text{endBeforeStart}(x_i, x_j) && \forall (i,j)\in P && (5)\\
&\text{interval } \ x_i && \forall i\in[1..N] && (6)\\
&\text{interval } \ y_{ij}, \ \text{optional}, \ \text{size} = PT_{ij} && \forall i\in[1..N], j\in M[i] && (7)
\end{aligned}$$

3.5 Multi-Mode Resource-constrained project scheduling problem with discounted cash flows

$$\begin{aligned}
&\max \quad \sum_{i\in[1..N]} CF[i]\cdot e^{-\alpha\cdot\text{endOf}(x_i)} && && (1)\\
&\text{alternative}(x_i, [y_{ij}]_{j\in M[i]}) && \forall i\in[1..N] && (2)\\
&\sum_{i\in[1..N],j\in M[i]} \text{pulse}(y_{ij}, QR_{ijk}) \le CR_k && \forall k\in[1..R] && (3)\\
&\sum_{i\in[1..N],j\in M[i]} \text{presenceOf}(y_{ij})\cdot QS_{ijk} \le CS_k && \forall k\in[1..S] && (4)\\
&\text{endBeforeStart}(x_i, x_j) && \forall (i,j)\in P && (5)\\
&\text{interval } \ x_i \subset [0,H) && \forall i\in[1..N] && (6)\\
&\text{interval } \ y_{ij}, \ \text{optional}, \ \text{size} = PT_{ij} && \forall i\in[1..N], j\in M[i] && (7)
\end{aligned}$$

3.6 Resource allocation and scheduling problem

Here is a CP Optimizer formulation of the resource allocation and scheduling problem proposed in [**?**]. The problem is defined by a set of jobs J and a set of facilities I. Each job $j \in J$ must be assigned to a facility $i \in I$ and scheduled to start after its release date R_j, end before its due date D_j, and execute for P_{ij} consecutive time units. Each job j has a facility assignment cost F_{ij} and a resource requirement Q_{ij} when allocated to facility i. Each facility $i \in I$ has a capacity C_i and the constraint that the resource capacity must not be exceeded at any time. The problem is to minimize the total facility assignment cost.

$$\min \sum_{i \in I, j \in J} F_{ij} \,\text{presenceOf}(y_{ij}) \tag{1}$$

$$\text{alternative}(x_j, [y_{ij}]_{i \in I}) \qquad \forall j \in J \tag{2}$$

$$\sum_{j \in J} \text{pulse}(y_{ij}, Q_{ij}) \leq C_i \qquad \forall i \in I \tag{3}$$

$$\text{interval } x_j \subset [R_j, D_j] \qquad \forall j \in J$$

$$\text{interval } y_{ij}, \text{ optional}, \text{ size} = P_{ij} \quad \forall i \in I, j \in J$$

3.7 Simplified photo-lithography machine

Here is a CP Optimizer formulation to model a batching machine in the context of a photo-lithography scheduling problem. A set of N operations is to be scheduled on the machine. Each operation x_i consists in the treatment of a set of W_i wafers on the machine. Each operation i specifies a minimal (A_i) and a maximal (B_i) duration (Equation 3). There are different families of operations, the family of an operation x_i is denoted F_i. The machine can perform several operations at the same time (notion of batch) provided that (1) the duration of the operations is the same as that of the batch, (2) the operations are from the same family and (3) the total capacity C of the machine in terms of number of wafers is not exceeded. Batches of operations are synchronized: that is, all operations in the same batch start (resp. end) at the same time. Furthermore, some family-dependent setup time given by a matrix M is needed to configure the machine from a given batch family to the next batch family. The limited capacity is modeled as a cumul function (Constraint 2). A state function (Equation 4) describes the evolution over time of the operation family currently executing on the machine. Batching constraints are defined using `alwaysEqual` constraints on a state function with start and end alignment (Constraint 1).

$$\text{alwaysEqual}(s, x_i, F[i], 1, 1) \quad \forall i \in [1..N] \tag{1}$$

$$\sum_{i \in [1..N]} \text{pulse}(x_i, W_i) \leq C \tag{2}$$

$$\text{interval } x_i, \text{ size} \in [A_i, B_i] \quad \forall i \in [1..N] \tag{3}$$

$$\text{stateFunction } s \text{ with } M \tag{4}$$

3.8 Satellite observation scheduling

Here is the CP Optimizer formulation of the GEO-CAPE Observation Scheduling Problem described in [**?**]. A piecewise linear function V represents the gain for a given delay between consecutive observations. And a stepwise function N_i represents the non-observation time slots of the i^{th} spot to be observed. SB, ST, TU are some constants of the problem.

$$
\begin{array}{llr}
\max \quad \sum_{i\in[1..N],j\in[1..M]} \text{lengthEval}(s_{ij}, V) & & (1)\\
\text{presenceOf}(a_{ij+1}) = \text{presenceOf}(s_{ij}) & \forall i \in [1..N], j \in [1..M] & (2)\\
\text{startAtStart}(a_{ij}, s_{ij}) & \forall i \in [1..N], j \in [1..M] & (3)\\
\text{endAtStart}(s_{ij}, a_{ij+1}) & \forall i \in [1..N], j \in [1..M] & (4)\\
\text{alternative}(s_{ij}, [sv_{ij}, so_{ij}]) & \forall i \in [1..N], j \in [1..M] & (5)\\
\text{presenceOf}(a_{i2}) = \text{presenceOf}(a_{i1}) & \forall i \in [1..N] & (6)\\
\text{presenceOf}(so_{i1}) = 0 & \forall i \in [1..N] & (7)\\
\text{presenceOf}(a_{ij+1}) \leq \text{presenceOf}(a_{ij}) & \forall i \in [1..N], j \in [2..M+1] & (8)\\
\text{presenceOf}(so_{ij-1}) \leq \text{presenceOf}(sv_{ij}) & \forall i \in [1..N], j \in [2..M+1] & (9)\\
\text{forbidExtent}(a_{ij}, N_i) & \forall i \in [1..N], j \in [1..M+1] & (10)\\
\text{noOverlap}([a_{ij}]_{i\in[1..N],j\in[1..M+1]}) & & (11)\\
\text{interval}\, a_{ij},\ \text{optional},\ \text{size} = 1 & \forall i \in [1..N], j \in [1..M+1] & \\
\text{interval}\, s_{ij},\ \text{optional} & \forall i \in [1..N], j \in [1..M] & \\
\text{interval}\, sv_{ij},\ \text{optional},\ \text{size} \in [SB..ST] & \forall i \in [1..N], j \in [1..M] & \\
\text{interval}\, so_{ij},\ \text{optional},\ \text{size} \in [ST+1..TU] & \forall i \in [1..N], j \in [1..M] &
\end{array}
$$

3.9 Energy-aware multiple state machine scheduling problem

Here is a reformulation of the extension of the job-shop scheduling problem described in [**?**]. Set $O_k = \{(i,j) | i \in [1..N], j \in [1..M] \mid M_{ij} = k\}$ denotes the set of operations requiring machine $k \in [1..M]$. A sequence variable s_k is created for each machine. For each operation x_p on machine k we define an additional interval variable y_p that represents the gap between the end of operation x_p and the start of the next operation. When x_p is the last operation on the machine, interval y_p ends at the horizon H of the schedule. The objective function is a lexicographical objective. The first criterion is to minimize the energy consumption which is a function of the duration of the gaps between operations on the machines.

$$\min \quad \text{staticLex}\Big(\sum_{k\in[1..M],p\in O_k} \text{lengthEval}(y_p, F_k),\ \max_{i\in[1..N]} \text{endOf}(x_{iM})\Big)$$

$$
\begin{aligned}
&\text{noOverlap}(s_k) && \forall k \in [1..M] && (1)\\
&\text{endBeforeStart}(x_{ij}, x_{ij+1}) && \forall i \in [1..N], j \in [1..M-1] && (2)\\
&\text{endAtStart}(x_p, y_p) && \forall k \in [1..M], p \in O_k && (3)\\
&\text{endOf}(y_p) = \text{startOfNext}(s_k, x_p, H) && \forall k \in [1..M], p \in O_k && (4)\\
&\text{interval } \ x_{ij} \subset [0..H], \ \text{size} = P_{ij} && \forall i \in [1..N], j \in [1..M] &&\\
&\text{interval } \ y_{ij} \subset [0..H] && \forall i \in [1..N], j \in [1..M] &&\\
&\text{sequence } \ s_k \text{ on } [x_p]_{p \in O_k} && \forall k \in [1..M] &&
\end{aligned}
$$

The following redundant constraints can also be added to strengthen the model. Interval variable c_k is an interval that starts at the start time of machine k and ends at H.

$$
\begin{aligned}
&\text{span}(c_k, [h] \cup [x_p]_{p \in O_k}) && \forall k \in [1..M] && (5)\\
&\sum_{p \in O_k} \text{lengthOf}(x_p) + \text{lengthOf}(y_p) = \text{lengthOf}(c_k) && \forall k \in [1..M] && (6)\\
&\text{alwaysIn}(\sum_{p \in O_k} \text{pulse}(x_p, 1) + \text{pulse}(y_p, 1), c_k, 1, 1) && \forall k \in [1..M] && (7)\\
&\text{interval } \ h \subset [H..H], \ \text{size} = 0 && &&\\
&\text{interval } \ c_k && \forall k \in [1..M] &&
\end{aligned}
$$

A Appendix: Keywords

We use the following notations for the arguments of the operators:

a,*b*,*c*,*d*	Integer or numerical constants
stp	Stepwise function
pwl	Piecewise linear function
u,*v*,*w*	Integer variables or expressions
x,*y*,*z*	Interval variables
r, s	Sequence variables
cf	Cumul function
sf	State function

Upper cases denote vectors, for instance Y denote a vector of interval variables, A denotes a vector of integer constants. M denotes a matrix of integers. Optional arguments are denoted with square brackets "[]". Variants of a given keyword are denoted "[VARIANT1|VARIANT2|...]".

Table 1: Keywords for variable types

Keyword	Description
integer	Integer variable
interval	Interval variable
sequence	Sequence variable on interval variables
stateFunction	State function

Table 2: Keywords and arguments for expressions

Keyword	Short description
$+$,$-$,$*$,$/$,$\sum_i$, $\prod_i$, $\|\|$, min, max, $\div$, $\bmod$, $\log(u)$, u^v, $\lfloor u \rfloor$, $\lceil u \rceil$, ...	Classical arithmetical expressions
$A[v], U[v]$	Array expressions: v is an integer index variable
count(U, a)	Count variables with given value
countDifferent(U)	Count number of different values
standardDeviation(U, a, b)	Standard deviation
[start|end|size|length]Of$(x[, a])$	Start (etc.) value of an interval variable
[start|end|size|length]Eval$(pwl, x[, a])$	Piecewise linear function evaluated on the start (etc.) value
[start|end|size|length|type]OfNext$(s, x, a[, b])$	Start (etc.) value of next interval in a sequence
[start|end|size|length|type]OfPrev$(s, x, a[, b])$	Start (etc.) value of previous interval in a sequence
heightAt[Start|End](cf, x)	Contribution of x to a cumul function at start (or end)
overlapLength$(x, y[, a])$	Overlap length between interval variables
pulse$(x, a[, b])$	Cumul expression: pulse
step(a, b)	Cumul expression: step at constant value
stepAt[Start|End]$(x, a[, b])$	Cumul expression: step at start (or end) of interval variable

Table 3: Keywords and arguments for constraints

Keyword	Short description
$=, \neq,$ $\leq, \geq, <, >$	Classical arithmetical constraints
allDifferent(V)	Global all different constraint
pack(U, V, A, w)	Bin-packing constraint
allMinDistance(U, a)	Minimal distance between all values
inverse(U, V)	Inverse constraint
allowedAssignments(U, M)	Allowed combinations of values
forbiddenAssignments(U, M)	Forbidden combinations of values
lexicographic(U, V)	Lexicographic ordering constraint
presenceOf(x)	Presence of an interval variable
[start|end][Before|At][Start|End]$(x, y[, a])$	Precedence constraints
forbid[Start|End|Extent](x, stp)	Forbidden values
alternative$(x, Y[, u])$	Alternative
span(x, Y)	Span
noOverlap$(s[, M, bool])$	No-overlap
first(s, x)	First on a sequence
last(s, x)	Last on a sequence
prev(s, x, y)	Immediately before on a sequence
before(s, x, y)	Before on a sequence
sameSequence$(r, s[, X, Y])$	Same sequence
sameCommonSubsequence$(r, s[, X, Y])$	Same common subsequence
alwaysIn(f, x, a, b)	Always-in constraint on cumul or state function
alwaysEqual$(sf, x, a[, bool, bool])$	Always-equal constraint on state function
alwaysConstant$(sf, x[, bool, bool])$	Always-constant constraint on state function
alwaysNoState(sf, x)	Always-no-state constraint on state function