Expressing “Logical” Constraints and Conveying Them to Solvers

Robert Fourer, Jun Ma
Industrial Engineering & Management Sciences
McCormick School of Engineering and Applied Science
Northwestern University
{4er,maj}@iems.northwestern.edu

Kipp Martin
Graduate School of Business
University of Chicago
kmartin@gsb.uchicago.edu

GICOLAG Workshop
University of Vienna, December 4-8, 2006
Contents

**Assorted examples**
- Machine scheduling
- Small integer programs
- CPLEX “indicator” constraints

**“Logical” operators and constraints**
- Usefulness in formulations
- Expression in algebraic modeling languages
- Conveyance to solvers

**Standard forms for problem instances**
- XML-based forms for traditional solvers
- Extensions to “logical” operators and constraints
Machine Scheduling: $1 / s_{ij} / [S_{Tj} = 0]$

1 machine, available continuously

$n$ jobs with known, constant data, available at time 0

- $f = \text{number of job families}$
- $n_i = \text{number of jobs in family } i, \text{where } n_1 + \ldots + n_f = n$
- $s_{jj} = \text{setup time incurred changing from family } i \text{ to family } j$
Background

1 / s / \(L_{max}\) is NP-hard
  - Bruno and Downey (1978) in *SIAM J. on Computing*

1 / / \(ST_j\) is NP-hard
  - Du and Leung (1990) in *Math. of OR*

**EDDWIF Property for 1 / s_{ij} / ST_j:**
  - There exists an optimal schedule such that the jobs within each family are sequences in nondecreasing order of their due dates.
  - Woodruff and Spearman (1992) in *POM*
Constraint Programming Model

Data description (in ILOG OPL)

```oql
enum Jobs ... ;
int+ procTime[Jobs] = ... ;
int+ dueTime[Jobs] = ... ;
Jobs succ[Jobs] = ... ;

enum Families = ... ;
Families jobFamily[Jobs] = ... ;

int setupTime[Families,Families] = ... ;
```
\section*{CP Model (cont’d)}

\subsection*{Model description}

\begin{verbatim}
UnaryResource Machine(setupTime);
Activity Do[j in Jobs](procTime[j])
    transitionType jobFamily[j];
solve {
    forall(j in Jobs) Do[j] requires Machine;
    forall(j in Jobs) Do[j].end <= dueTime[j];
    forall(j in Jobs: succ[j] <> j)
        Do[j].end <= Do[succ[j]].end;
};
\end{verbatim}
**CP Model (cont’d)**

*Search directive*

```plaintext
search {
    LDSearch(1) {
        while not isRanked(Machine) do
            select(j in Jobs: not isRanked(Machine, Do[j])
            ordered by increasing
            <dmin(Do[j].start), dueTime[j]>)
            tryRankFirst(Machine, Do[j]);
    }
};
```
Test Setup

**Fixed factors:**
- Number of families 4
- Number of jobs 60

**Varying factors:**
- Slack in due dates 0 5
- Target number of setups 21 7
- Mean of random setup times 50 5
# Test Results

<table>
<thead>
<tr>
<th>Due-date slack</th>
<th>Target setups</th>
<th>Setup times</th>
<th>Without EDDWIF</th>
<th>With EDDWIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>21</td>
<td>50</td>
<td>5.44</td>
<td>1.92</td>
</tr>
<tr>
<td>0</td>
<td>21</td>
<td>5</td>
<td>.77</td>
<td>.71</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>50</td>
<td>&gt;</td>
<td>3.24</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>5</td>
<td>1.15</td>
<td>.76</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>50</td>
<td>.66</td>
<td>.77</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>5</td>
<td>.72</td>
<td>1.32</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>50</td>
<td>146.21</td>
<td>3.35</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>5</td>
<td>&gt;</td>
<td>1.49</td>
</tr>
</tbody>
</table>
Small Integer Programs

\textit{knapsack}
- 12 integer variables, 7 constraints

\textit{prodpln}
- (1) 36 integer variables, 32 constraints
- (2) 36 integer variables, 35 constr (3 cuts added)

\textit{diet}
- (1) small: 9 integer variables, 7 constraints
- (2) large: 63 integer variables, 16 constraints
  * both with (a) integer and (b) real objective

\textit{team}
- 93 binary variables, 36 constraints
Small IPs: Results

- knapsack
- prodpln1
- prodpln2
- diet1a
- diet1b
- diet2a
- diet2b
- team

<table>
<thead>
<tr>
<th></th>
<th>CPLEX</th>
<th>ILOG Planner</th>
</tr>
</thead>
<tbody>
<tr>
<td>knapsack</td>
<td>0.001</td>
<td>0.100</td>
</tr>
<tr>
<td>prodpln1</td>
<td>0.001</td>
<td>0.100</td>
</tr>
<tr>
<td>prodpln2</td>
<td>0.001</td>
<td>0.100</td>
</tr>
<tr>
<td>diet1a</td>
<td>0.001</td>
<td>0.100</td>
</tr>
<tr>
<td>diet1b</td>
<td>0.001</td>
<td>0.100</td>
</tr>
<tr>
<td>diet2a</td>
<td>0.001</td>
<td>0.100</td>
</tr>
<tr>
<td>diet2b</td>
<td>0.001</td>
<td>0.100</td>
</tr>
<tr>
<td>team</td>
<td>0.001</td>
<td>0.100</td>
</tr>
</tbody>
</table>
“Indicator” Constraints (CPLEX 10.1)

**General form**

- (binary variable = 0) implies constraint
- (binary variable = 1) implies constraint

**AMPL representation**

- Uses ==> for “implies”
- Also recognizes an else clause
- Similarly defines <= and <=>
  * if-then-else expressions & statements as before
**CPLEX 10.1 (cont’d)**

*User cuts*
- Satisfied by any integer-feasible solution, but . . .
- Cut off some fractional solutions

*Lazy constraints*
- Required by any feasible solution, but . . .
- Most will be satisfied even if left out

*AMPL “suffix” settings*
- `.lazy = 1` indicates a lazy constraint
- `.lazy = 2` indicates a user cut
- AMPL generates all, but CPLEX only includes some
Example 1

Multicommodity flow with fixed costs

```
set ORIG;   # origins
set DEST;   # destinations
set PROD;   # products

param supply {ORIG,PROD} >= 0;  # amounts available at origins
param demand {DEST,PROD} >= 0;  # amounts required at destinations
param limit {ORIG,DEST} >= 0;

param vcost {ORIG,DEST,PROD} >= 0; # variable shipment cost on routes
param fcost {ORIG,DEST} > 0;       # fixed cost on routes

var Trans {ORIG,DEST,PROD} >= 0;   # actual units to be shipped
var Use {ORIG, DEST} binary;       # = 1 iff link is used

minimize total_cost:
    sum {i in ORIG, j in DEST, p in PROD} vcost[i,j,p] * Trans[i,j,p]
    + sum {i in ORIG, j in DEST} fcost[i,j] * Use[i,j];
```
Example 1 (cont’d)

Conventional constraints

subject to Supply \{i \in \text{ORIG}, p \in \text{PROD}\}:
    \sum \{j \in \text{DEST}\} \ Trans[i,j,p] = \text{supply}[i,p];

subject to Demand \{j \in \text{DEST}, p \in \text{PROD}\}:
    \sum \{i \in \text{ORIG}\} \ Trans[i,j,p] = \text{demand}[j,p];

subject to Multi \{i \in \text{ORIG}, j \in \text{DEST}\}:
    \sum \{p \in \text{PROD}\} \ Trans[i,j,p] \leq \text{limit}[i,j] \ast \text{Use}[i,j];

subject to Supply \{i \in \text{ORIG}, p \in \text{PROD}\}:
    \sum \{j \in \text{DEST}\} \ Trans[i,j,p] = \text{supply}[i,p];

subject to Demand \{j \in \text{DEST}, p \in \text{PROD}\}:
    \sum \{i \in \text{ORIG}\} \ Trans[i,j,p] = \text{demand}[j,p];

subject to Multi \{i \in \text{ORIG}, j \in \text{DEST}\}:
    \sum \{p \in \text{PROD}\} \ Trans[i,j,p] \leq \text{limit}[i,j];

subject to UseDefinition \{i \in \text{ORIG}, j \in \text{DEST}, p \in \text{PROD}\}:
    \ Trans[i,j,p] \leq \text{min}(\text{supply}[i,p], \text{demand}[j,p]) \ast \text{Use}[i,j];
Example 1 (cont’d)

User cuts

subject to Multi \{i \in ORIG, j \in DEST\}:
   \sum \{p \in PROD\} Trans[i,j,p] \leq limit[i,j] \ast Use[i,j];

subject to UseDefinition \{i \in ORIG, j \in DEST, p \in PROD\}:
   Trans[i,j,p] \leq \min(supply[i,p], demand[j,p]) \ast Use[i,j];
Example 1 (cont’d)

Indicator constraint formulations

subject to DefineUsedA \{i \text{ in ORIG}, j \text{ in DEST}\}:

\[ \text{Use}[i,j] = 0 \implies \sum \{p \text{ in PROD}\} \text{Trans}[i,j,p] = 0; \]

subject to DefineUsedB \{i \text{ in ORIG}, j \text{ in DEST}, p \text{ in PROD}\}:

\[ \text{Use}[i,j] = 0 \implies \text{Trans}[i,j,p] = 0; \]

subject to DefineUsedC \{i \text{ in ORIG}, j \text{ in DEST}\}:

\[ \text{Use}[i,j] = 0 \implies \sum \{p \text{ in PROD}\} \text{Trans}[i,j,p] = 0 \]

\[ \text{else sum} \{p \text{ in PROD}\} \text{Trans}[i,j,p] \leq \text{limit}[i,j]; \]
Example 1 (cont’d )

Results for 3 origins, 7 destinations, 3 products

<table>
<thead>
<tr>
<th></th>
<th>iters</th>
<th>nodes</th>
<th>cuts</th>
<th>used</th>
</tr>
</thead>
<tbody>
<tr>
<td>no cuts</td>
<td>374</td>
<td>79</td>
<td></td>
<td></td>
</tr>
<tr>
<td>all cuts</td>
<td>317</td>
<td>39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>user cuts</td>
<td>295</td>
<td>42</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>indic A</td>
<td>355</td>
<td>77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>indic B</td>
<td>406</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>indic C</td>
<td>277</td>
<td>57</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 2

Assignment to groups with “no one isolated”

\[
\begin{align*}
\text{var } & \text{Lone } \{(i_1,i_2) \in \text{ISO}, \; j \in \text{REST}\} \; \text{binary}; \\
\text{param } & \text{give } \{\text{ISO}\} \; \text{default } 2; \\
\text{param } & \text{giveTitle } \{\text{TITLE}\} \; \text{default } 2; \\
\text{param } & \text{giveLoc } \{\text{LOC}\} \; \text{default } 2; \\
\text{param } & \text{upperbnd } \{(i_1,i_2) \in \text{ISO}, \; j \in \text{REST}\} = \\
& \quad \min \left(\text{ceil}\left(\frac{\text{number2}[i_1,i_2]}{\text{card}\{\text{PEOPLE}\}} \times \text{hiDine}[j]\right) + \text{give}[i_1,i_2], \right. \\
& \left. \quad \text{hiTargetTitle}[i_1,j] + \text{giveTitle}[i_1], \right. \\
& \left. \quad \text{hiTargetLoc}[i_2,j] + \text{giveLoc}[i_2], \; \text{number2}[i_1,i_2]\right); \\
\text{subj to } & \text{Isolation1 } \{(i_1,i_2) \in \text{ISO}, \; j \in \text{REST}\}: \\
& \text{Assign2}[i_1,i_2,j] \leq \text{upperbnd}[i_1,i_2,j] \times \text{Lone}[i_1,i_2,j]; \\
\text{subj to } & \text{Isolation2a } \{(i_1,i_2) \in \text{ISO}, \; j \in \text{REST}\}: \\
& \text{Assign2}[i_1,i_2,j] \geq \text{Lone}[i_1,i_2,j]; \\
\text{subj to } & \text{Isolation2b } \{(i_1,i_2) \in \text{ISO}, \; j \in \text{REST}\}:
& \text{Assign2}[i_1,i_2,j] + \\
& \quad \sum_{ii_1 \in \text{ADJACENT}[i_1]} \{(ii_1,i_2) \in \text{TYPE2}\} \text{Assign2}[ii_1,i_2,j] \\
& \quad \geq 2 \times \text{Lone}[i_1,i_2,j];
\end{align*}
\]
Example 2

Same using indicator constraints

```plaintext
var Lone {(i1,i2) in ISO, j in REST} binary;
subj to Isolation1 {(i1,i2) in ISO, j in REST}:
    Lone[i1,i2,j] = 0 ==> Assign2[i1,i2,j] = 0;
subj to Isolation2b {(i1,i2) in ISO, j in REST}:
    Lone[i1,i2,j] = 1 ==> Assign2[i1,i2,j] +
        sum {ii1 in ADJACENT[i1]: (ii1,i2) in TYPE2} Assign2[ii1,i2,j] >= 2;
```
Example 3

Workforce planning

\[
\begin{align*}
\text{var NoShut} \ {m} \ \text{in MONTHS} & \text{ binary;} \\
\text{var LayoffCost} \ {m} \ \text{in MONTHS} & \geq 0; \\
\text{subj to NoShutDefn1} \ {m} \ \text{in MONTHS}: \\
& \quad \text{LayoffCost}[m] \\
& \quad \quad \leq \text{snrLayOffWages} \times 31 \times \text{maxNbrSnrEmpl} \times (1 - \text{NoShut}[m]); \\
\text{subj to NoShutDefn2a} \ {m} \ \text{in MONTHS}: \\
& \quad \text{LayoffCost}[m] - \text{snrLayOffWages} \times \text{ShutdownDays}[m] \times \text{maxNbrSnrEmpl} \\
& \quad \quad \leq \text{maxNbrSnrEmpl} \times 2 \times \text{dayAvail}[m] \times \text{snrLayOffWages} \times \text{NoShut}[m]; \\
\text{subj to NoShutDefn2b} \ {m} \ \text{in MONTHS}: \\
& \quad \text{LayoffCost}[m] - \text{snrLayOffWages} \times \text{ShutdownDays}[m] \times \text{maxNbrSnrEmpl} \\
& \quad \quad \geq -\text{maxNbrSnrEmpl} \times 2 \times \text{dayAvail}[m] \times \text{snrLayOffWages} \times \text{NoShut}[m];
\end{align*}
\]

Fourer, Ma, Martin, GICOLAG Workshop, University of Vienna, December 4-8, 2006
Example 3

*Same using indicator constraints*

```plaintext
var NoShut {m in MONTHS} binary;
var LayoffCost {m in MONTHS} >=0;
subj to NoShutDefn1 {m in MONTHS}:
    NoShut[m] = 1 ==> LayoffCost[m] = 0;
subj to NoShutDefn2 {m in MONTHS}:
    NoShut[m] = 0 ==> LayoffCost[m] =
    snrLayoffWages * ShutdownDays[m] * maxNumberSnrEmpl;
```
“Logical” Expressions in Optimization

Formulations
- More natural for modelers than integer programs
- Independent of solvers
- Compatible with existing modeling languages

Solution methods
- Theoretically optimal
- Based on tree search (like branch & bound)
- Sensitive to details of search strategy
Example: Job Sequencing with Setups

Given

- A set of jobs, with production times, due times and earliness penalties
- One machine that processes one job at a time
- Setup costs and times between jobs
- Precedence relations between certain jobs

Choose

- A sequence for the jobs

Minimizing

- Setup costs plus earliness penalties

Example: Variables and Costs

*Either way*

- $\text{ComplTime}[j]$ is the completion time of job $j$
- Earliness penalty is the sum over jobs $j$ of $\text{duePen}[j] \times (\text{dueTime}[j] - \text{ComplTime}[j])$

*Integer programming formulation*

- $\text{Seq}[i,j] = 1$ iff $i$ immediately precedes $j$
- Setup cost is the sum over job pairs $(i,j)$ of $\text{setupCost}[i,j] \times \text{Seq}[i,j]$

*More natural formulation*

- $\text{JobForSlot}[k]$ is the job in the $k$th slot in sequence
- Setup cost is the sum over slots $k$ of $\text{setupCost}[$JobForSlot$[k],\text{JobForSlot}[k+1]]$
Example: Production Constraints

**Integer programming formulation**

For each job \( i \), \( \text{ComplTime}[i] \leq \text{dueTime}[i] \)

For each job pair \((i,j)\),

\[
\text{ComplTime}[i] + \text{setupTime}[i,j] + \text{procTime}[j] \leq \\
\text{ComplTime}[j] + \text{BIG} \times (1 - \text{Seq}[i,j])
\]

**More natural formulation**

For each slot \( k \),

\[
\text{ComplTime}[\text{JobForSlot}[k]] = \min ( \\
\text{dueTime}[\text{JobForSlot}[k]], \\
\text{ComplTime}[\text{JobForSlot}[k+1]] \\
- \text{procTime}[\text{JobForSlot}[k+1]] \\
- \text{setupTime}[\text{JobForSlot}[k],\text{JobForSlot}[k+1]] )
\]
Example: Sequencing Constraints

**Integer programming formulation**

For each job \( i \),

\[
\sum_{j \in \text{JOBS}} \text{Seq}[i,j] = 1
\]

For each job \( i \),

\[
\sum_{j \in \text{JOBS}} \text{Seq}[j,i] = 1
\]

**More natural formulation**

\[
\text{alldiff} \{k \in \text{SLOTS} \mid \text{JobForSlot}[k] \}
\]
Operations and Constraints

Design considerations

- Value types
- Constraint types
- New operators

Examples of tentative extensions

- Logic operators
- Counting operators
- Special-structure (“global”) operators
- Indexing by variables

Conveyance of constraints to solvers

- Range constraints
- Logical constraints
Value Types

Numerical
  ❖ Value is a number
    var Trans {ORIG, DEST} >= 0;

Logical
  ❖ Value is “true” or “false”

Object
  ❖ Value is a member of some set
    var JobForSlot {SLOTS} in JOBS;

Set
  ❖ Value is a set of numbers or objects:
    var MEMBERS {PROJECTS} within VOLUNTEERS;
Constraint Types

Range constraints

- \( \text{lowerBound} \leq \text{numExpr} \leq \text{upperBound} \)
- For one-sided constraint,
  \( \text{lowerBound} = -\infty \) or \( \text{upperBound} = +\infty \)
- For equality, \( \text{lowerBound} = \text{upperBound} \)

Logic constraints

- \( \text{logicExpr} \)
- Logical
  \((\text{Mk}[i] = 0 \text{ and } \text{Mk}[i] = 0) \text{ or } \text{Mk}[i] + \text{Mk}[i] \geq \text{ldb}\)
- Counting
  atmost \( \text{mxsrv} \ \{j \text{ in } D\} \ (\text{sum} \ \{p \text{ in } \text{PRD}\} \ \text{Tr}[i,j,p] \geq 10)\)
- Special-structure
  \( \text{alldiff} \ \{j \text{ in } \text{Jobs}\} (\text{MachineForJob}[j]) \)
New Operators

Numerical-valued on constraints

- Counting
  \[
  \text{count} \{ j \in D \} \ (\sum \{ p \in \text{PRD} \} \ Tr[i,j,p] \geq 10)
  \]

Logic-valued on constraints

- Logical
  \[
  (Mk[1] = 0 \text{ and } Mk[2] = 0) \text{ or } (Mk[1] + Mk[2] \geq 100)
  \]

- Counting
  \[
  \text{atmost mxsrv} \{ j \in D \} \ (\sum \{ p \in \text{PRD} \} \ Tr[i,j,p] \geq 10)
  \]

Special-structure ("global")

- All-different
  \[
  \text{alldiff} \{ j \in \text{Jobs} \} \ (\text{MachineForJob}[j])
  \]

- Distribution
  \[
  \text{numberof} \ 3 \ \text{in} \ \{\{ j \in 1..\text{nJobs} \} \ \text{MachineForJob}[j]\}
  \]
New Operators (cont’d)

Indexing

- Variables in subscripts of parameters or variables

```plaintext
param mCLI integer > 0;
param nLOC integer > 0;
param srvCost {1..mCLI, 1..nLOC} > 0;
param bdgCost > 0;
var Serve {1..mCLI} integer >= 1, <= nLOC;
var Open {1..nLOC} integer >= 0, <= 1;
minimize TotalCost:
    sum {i in 1..mCLI} srvCost[i,Serve[i]] +
    bdgCost * sum {j in 1..nLOC} Open[j];
subject to OpenDefn {i in 1..mCLI}:
    Open[Serve[i]] = 1;
```
New Operators (cont’d)

Indexing

- Variables constrained by subscripts

```plaintext
set ABLE within {1..mCLI, 1..nLOC};
param srvCost {ABLE} > 0;

.......  
minimize TotalCost:
  sum {i in 1..mCLI} srvCost[i,Serve[i]] + ...

...(i,Serve[i]) must be in ABLE
```

With set operands

- Set valued: union, intersection, difference
- Numerical valued: cardinality
- Logic valued: membership, containment
- Special-structure: all-disjoint
Conveying “Range” Constraints

General format

- lower-bound ≤
  linear-expr + nonlinear-expr ≤ upper-bound
- Arrays of lower-bound and upper-bound values
- Coefficient lists for linear-expr
- Expression tree for nonlinear-expr

Expression tree nodes

- Variables, constants
- Binary, unary operators
- Iterated summation, min, max
- Piecewise-linear terms
- If-then-else terms

... single array of variables
“Walking the Tree”: AMPL Interface to ILOG Concert

Definition of variables

IloNumVarArray Var(env, n_var);

for (j = 0; j < n_var - n_var_int; j++)
    Var[j] = IloNumVar(env, loVarBnd[j], upVarBnd[j], ILOFLOAT);

for (j = n_var - n_var_int; j < n_var; j++)
    Var[j] = IloNumVar(env, loVarBnd[j], upVarBnd[j], ILOINT);
Tree Walk (cont’d)

Top-level processing of range constraints

```c
IloRangeArray Con(env, n_con);
for (i = 0; i < n_con; i++) {
    IloExpr conExpr(env);
    if (i < nlc)
        conExpr += build_expr (con_de[i].e);
    for (cg = Cgrad[i]; cg; cg = cg->next)
        conExpr += (cg -> coef) * Var[cg -> varno];
    Con[i] = (loConBnd[i] <= conExpr <= upConBnd[i]);
}
```
Tree Walk (cont’d)

Tree-walk function for expressions

```c
IloExpr build_expr (expr *e)
{
    expr **ep;
    IloInt opnum;
    IloExpr partSum;
    opnum = (int) e->op;
    switch(opnum) {
        case PLUS_opno: ...
        case MINUS_opno: ...
        ........
    }
}
```
Tree Walk (cont’d)

Tree-walk cases for expression nodes

```c
switch(opnum) {
    case PLUS_opno:
        return build_expr (e->L.e) + build_expr (e->R.e);

    case SUMLIST_opno:
        partSum = IloExpr(env);
        for (ep = e->L.ep; ep < e->R.ep; *ep++)
            partSum += build_expr (*ep);
        return partSum;

    case LOG_opno:
        return IloLog (build_expr (e->L.e));

    .......

    case VAR_opno:
        return Var[e->a];

    case CONST_opno:
        return IloExpr (env, ((expr_n*)e)->v);
}
```
Conveying Logical Constraints

**Simple forms**
- constraint **and** constraint
- constraint **or** constraint
- **not** constraint

```
```

**Representation**
- Expression tree for entire constraint
- Constraint nodes whose children are constraint nodes
- Constraint nodes whose children are expression nodes
Tree Walk (cont’d)

Top-level processing of logical constraints

```c
IloConstraintArray LCon(env,n_lcon);
for (i = 0; i < n_lcon; i++) {
    LCon[i] = build_constr (lcon_de[i].e);
}
```
Tree Walk (cont’d)

Tree-walk function for constraints

```c
IloConstraint build_constr (expr *e)
{
    expr **ep;
    IloInt opnum;
    opnum = (int) e->op;
    switch(opnum) {
        .......
    }
}
```
Tree Walk \textit{(cont’d)}

Tree-walk cases for constraint nodes

\begin{verbatim}
switch(opnum) {
    case OR_opno:
        return build_constr (e->L.e) || build_constr (e->R.e);
    case AND_opno:
        return build_constr (e->L.e) && build_constr (e->R.e);
    case GE_opno:
        return build_expr (e->L.e) >= build_expr (e->R.e);
    case EQ_opno:
        return build_expr (e->L.e) == build_expr (e->R.e);
    .......
}
\end{verbatim}
Further Logical Constraint Cases

**Constraint types**
- Counting expressions and constraints
- Structure (global) constraints
- Variables in subscripts

**Solver inputs**
- C++ types and operators (ILOG Concert)
- Unindexed algebraic input format (BARON)
- Codelist of 4-tuples (GlobSol)
- Compact, flexible NOP format (GLOPT)
XML-Based Standard Formats

Motivation

- for any standard format
- for an XML-based format

“OSxL” standards

- OSiL: problem instances
- OSoL: solver options
- OSrL: results

Components of OSiL

- XML schema for text file format, and
- Corresponding in-memory data structures
- Libraries for reading and writing the above
Standards

XML Means “Tagged” Text Files . . .

Example: html for a popular home page

```html
<html><head><meta http-equiv="content-type" content="text/html; charset=UTF-8"><title>Google</title><style>!---
body, td, a, p,.h{font-family:arial,sans-serif;}
.h{font-size: 20px;}
.q{text-decoration:none; color:#0000cc;}
//-->
</style>
</head><body bgcolor=#ffffff text=#000000 link=#0000cc vlink=#551a8b alink=#ff0000 onLoad=sf()><center><table border=0 cellspacing=0 cellpadding=0><tr><td><img src="/images/logo.gif" width=276 height=110 alt="Google"></td></tr></table><br>
```

. . . a collection of XML tags is designed for a special purpose  
. . . by use of a schema written itself in XML
Standards

Advantage of any standard

**MN drivers**

*without a standard*

**M + N drivers**

*with a standard*
Advantages of an XML Standard

Specifying it

- Unambiguous definition via a schema
- Provision for keys and data typing
- Well-defined expansion to new name spaces

Working with it

- Parsing and validation via standard utilities
- Amenability to compression and encryption
- Transformation and display via XSLT style sheets
- Compatibility with web services
Standards

What about “MPS Form”?

Weaknesses

- Standard only for LP and MIP, not for nonlinear, network, complementarity, logical, . . .
- Standard not uniform (especially for SP extension)
- Verbose ASCII form, with much repetition of names
- Limited precision for some numerical values

Used for

- Collections of (mostly anonymous) test problems
- Bug reports to solver vendors

Not used for

- Communication between modeling systems and solvers
Standards

Text from the OSiL Schema

```xml
<xs:complexType name="Variables">
  <xs:sequence>
    <xs:element name="var" type="Variable" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="number" type="xs:positiveInteger" use="required"/>
</xs:complexType>

<xs:complexType name="Variable">
  <xs:attribute name="name" type="xs:string" use="optional"/>
  <xs:attribute name="init" type="xs:string" use="optional"/>
  <xs:attribute name="type" use="optional" default="C">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="C"/>
        <xs:enumeration value="B"/>
        <xs:enumeration value="I"/>
        <xs:enumeration value="S"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="lb" type="xs:double" use="optional" default="0"/>
  <xs:attribute name="ub" type="xs:double" use="optional" default="INF"/>
</xs:complexType>
```
Standards

Diagram of the OSiL Schema
Standards

Details of OSiL’s instanceData Element
Standards

Details of OSiL’s instanceData Element
Example: A Problem Instance (in AMPL)

```ampl
ampl: expand _var;

Coefficients of x[0]:
  Con1  1 + nonlinear
  Con2  7 + nonlinear
  Obj   0 + nonlinear

Coefficients of x[1]:
  Con1  0 + nonlinear
  Con2  5 + nonlinear
  Obj   9 + nonlinear

ampl: expand _obj;

minimize Obj:
  (1 - x[0])^2 + 100*(x[1] - x[0]^2)^2 + 9*x[1];

ampl: expand _con;

subject to Con1:
  10*x[0]^2 + 11*x[1]^2 + 3*x[0]*x[1] + x[0] <= 10;

subject to Con2:
  log(x[0]*x[1]) + 7*x[0] + 5*x[1] >= 10;
```
Example in OSiL

```
<instanceHeader>
    <name>Modified Rosenbrock</name>
    <source>Computing Journal3:175-184, 1960</source>
    <description>Rosenbrock problem with constraints</description>
</instanceHeader>

<variables number="2">
    <var lb="0" name="x0" type="C"/>
    <var lb="0" name="x1" type="C"/>
</variables>

<objectives number="1">
    <obj maxOrMin="min" name="minCost" numberOfObjCoef="1">
        <coef idx="1">9</coef>
    </obj>
</objectives>

<constraints number="2">
    <con ub="10.0"/>
    <con lb="10.0"/>
</constraints>
```
Standard formats

Example in OSiL (continued)

```xml
<linearConstraintCoefficients numberOfValues="3">
  <start>
    <el>0</el>
    <el>1</el>
    <el>3</el>
  </start>
  <rowIdx>
    <el>0</el>
    <el>1</el>
    <el>1</el>
    <el>1</el>
  </rowIdx>
  <value>
    <el>1.0</el>
    <el>7.0</el>
    <el>5.0</el>
  </value>
</linearConstraintCoefficients>

<quadraticCoefficients numberOfQPTerms="3">
  <qpTerm idx="0" idxOne="0" idxTwo="0" coef="10"/>
  <qpTerm idx="0" idxOne="1" idxTwo="1" coef="11"/>
  <qpTerm idx="0" idxOne="0" idxTwo="1" coef="3"/>
</quadraticCoefficients>
```
**Standard formats**

**Example in OSiL (continued)**

```xml
<nl idx="-1">
  <plus>
    <power>
      <minus>
        <number type="real" value="1.0"/>
        <variable idx="0"/>
      </minus>
      <number type="real" value="2.0"/>
    </power>
    <times>
      <number type="real" value="100"/>
      <power>
        <minus>
          <variable coef="1.0" idx="1"/>
          <power>
            <variable idx="0"/>
            <number type="real" value="2.0"/>
          </power>
        </minus>
        <number type="real" value="2.0"/>
      </power>
    </times>
  </plus>
</nl>
```
Example in OSiL (continued)

```xml
<nl id="1">
  <ln>
    <times>
      <variable idx="0"/>
      <variable idx="1"/>
    </times>
  </ln>
</nl>
```
Compression

Specific to OSiL

- Collapse sequences of row/column numbers
- Collapse repeated element values
- Encode portions using base-64 datatype

General for XML

- Compression schemes designed for XML files

Comparisons

- XML base-64 < MPS
- XML with multiple values collapsed < 2 × MPS
- Compressed XML < Compressed MPS
Other Features in OSiL . . .

In current specification
- Real-time data
- Functions defined by the user

In process of design
- Stochastic programming / optimization under uncertainty
  * Collaboration with Gus Gassmann
- Logical / combinatorial constraints
- Semidefinite / cone programming

Associated languages
- OSoL for communicating options to solvers
- OSrL for communicating results from solvers
  . . . broader family of “optimization services” languages
  (see www.optimizationservices.org)
In-Memory Data Structures

OSInstance object class

- Parallels the OSiL schema
- complexType in schema $\leftrightarrow$ class in OSInstance
- attributes / children of an element $\leftrightarrow$ members of a class
- choices / sequences in the schema arrays $\leftrightarrow$ array members

OS expression tree

- Parallels the nonlinear part of the OSiL schema
- Designed to avoid lengthy “switch” statements

Advantages

- One standard instead of two
- Complements COIN-OR’s OSI
Libraries (APIs, Interfaces)

**Use by client**
- OSInstance `set()` methods generate instance in memory
- OSiLWriter writes instance to a file in OSiL format
- Using SOAP over HTTP, instance is sent to a solver

**Use by solver**
- OSiLReader in solver interface
  
  reads instance from OSiL format back to memory
- OSInstance `get()` methods extract instance data
  as needed by solver
- Solver works on the problem
- Results are sent back similarly, using OSrL

... OSiL can be skipped when instance is passed in memory
Logic Extensions to OSiL

Design

- Use same “nonlinear” expression tree
- Define new nodes to represent new operators

Implementation

- Extend API to give solvers access to constraint expressions
Example: Logic Operators

(Mk[i] = 0 and Mk[i] = 0) or Mk[i] + Mk[i] >= lbd

<or>
  <and>
    <eq>
      <var idx="23"/>
      <number value="0"/>
    </eq>
    <eq>
      <var idx="103"/>
      <number value="0"/>
    </eq>
  </and>
  <geq>
    <plus>
      <var idx="23"/>
      <var idx="103"/>
    </plus>
    <number value="150"/>
  </geq>
</or>
Example: “at most” operator

\[
\text{atmost } \text{mxsrv} \{j \text{ in } D\} \left( \sum \{p \text{ in PRD}\} \ Tr[i,j,p] \geq \text{lim}[j] \right)
\]

```xml
<atMost>
    <number value="2"/>
    <geq>
        <sum>
            <var idx="20"/>
            <var idx="21"/>
            <var idx="22"/>
        </sum>
        <number value="10"/>
    </geq>
    <geq>
        <sum>
            <var idx="30"/>
            <var idx="31"/>
            <var idx="32"/>
        </sum>
        <number value="27"/>
    </geq>
    <geq>
        ...
    </geq>
</atMost>
```
Examples (cont'd)

alldiff {j in Jobs} (MachineForJob[j])

```
<alldiff>
  <var idx="27"/>
  <var idx="37"/>
  <var idx="47"/>
</alldiff>
```

Open[Serve[7]] where mCLI = 40, nLOC = 15

- Serve corresponds to Var[0], ..., Var[39]
- Open corresponds to Var[40], ..., Var[54]

```
<var>
  <idx>
    <plus>
      <number value="40"/>
      <var idx="6"/>
    </plus>
  </idx>
</var>
```