Constraint (Logic) Programming

Magnus Ågren
Computing Science, Information Technology, Uppsala University
agren@it.uu.se
October 14, 2005

What is a Constraint?

Constraints surround us in our everyday life and to reason about them is crucial for our society to work. Some constraints known to all of us are:

- Two lectures cannot overlap in time.
- I must have 20 kr to take the bus in Uppsala.
- The coding starts when the specification is finished.
- I must be at the airport 2 hours before departure.
- I must study for the exam if and only if I shall pass the exam.

Combinatorial Problems

A combinatorial problem can usually be thought of as trying to find a certain combination of assignments to variables such that they satisfy a set of given properties.

Such properties can be specified by a set of constraints on the variables.
**Combinatorial Optimisation Problems**

Sometimes an assignment that (in addition to satisfying a set of properties) maximises/minimises a given expression is needed.

Such problems are called *combinatorial optimisation problems*.

---

**The Map Colouring Problem**

**Given are:**
- A map over a number of countries.
- A set of colours.
- A positive integer $q$.

**Problem:** Colour the countries of the map in such a way that

- (c1) no two adjacent countries have the same colour.
- (c2) at most $q$ colours are used.

---

**A concrete example. . .**

- Map over the Nordic countries.
- The set of colours \{blue, green, purple, red, yellow\}
- $q = 3$

...and a solution.
The Wine Tasting Session Problem

Given are:
- A set $W$ of wines.
- A set $T$ of wine-tasters.

Problem: Construct an assignment of wines to tasters such that:
- (c1) each wine-taster tastes exactly $q$ different wines.
- (c2) each wine is tasted by exactly $r$ different wine-tasters.
- (c3) each pair of different wines is tasted by exactly $m$ different tasters.

A concrete example... $W = \{\text{Cardinal, GatoNegro, Preferido, Bellissimo, Periquita, Killawarra, Santiano}\}$ $T = \{\text{Robert, Al, John, Nicole, Liv, Jack, Jodie}\}$ $q = 3, r = 3, m = 1$

...and a solution.

<table>
<thead>
<tr>
<th>Robert</th>
<th>Al</th>
<th>John</th>
<th>Nicole</th>
<th>Liv</th>
<th>Jack</th>
<th>Jodie</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cardinal</td>
<td>GatoNegro</td>
<td>Preferido</td>
<td>Preferido</td>
<td>Bellissimo</td>
<td>Bellissimo</td>
<td>Bellissimo</td>
</tr>
<tr>
<td>Periquita</td>
<td>Killawarra</td>
<td>Killawarra</td>
<td>GatoNegro</td>
<td>Killawarra</td>
<td>GatoNegro</td>
<td>Preferido</td>
</tr>
<tr>
<td>Santiano</td>
<td>Santiano</td>
<td>Periquita</td>
<td>Cardinal</td>
<td>Cardinal</td>
<td>Periquita</td>
<td>Santiano</td>
</tr>
</tbody>
</table>

Examples of Combinatorial (Optimisation) Problems

- Hardware design
- Compilation
- Financial problems
- Layout problems
- Cutting problems
- Air traffic control
- Frequency allocation
- Configuration
- Production scheduling
- Satellite
- Maintenance
- Product blending

Constraint Programming

Constraint Programming is a framework for solving combinatorial (optimisation) problems based on constraints.

The framework provides off-the-shelf software components (constraints) to be composed together to specify each specific problem.

A given problem can be specified using constraints in many different ways. It is up to the user to do this in a correct and efficient way.
A User and His Problem

Very Hard Problem

A Library of Constraints
\text{all}\text{\_}\text{different}([x_1, x_2, \ldots, x_n]) \&
\text{all}\text{\_}\text{different}([y_1, y_2, \ldots, y_n]) \&
\text{AllDisjoint}(S \subseteq T) \&
\text{AllDisjoint}((S, T, U)) \&
x_1 < y_1 \&
x_n < y_n \&
x \neq y \&

Think, think, think, ... 

A Composition of Constraints (a model of the problem)
\text{all}\text{\_}\text{different}([x_1, x_2, \ldots, x_n]) \&
\text{all}\text{\_}\text{different}([y_1, y_2, \ldots, y_n]) \&
\text{AllDisjoint}(S \subseteq T) \&
\text{AllDisjoint}((S, T, U)) \&
x_1 < y_1 \&
x_n < y_n \&
x \neq y \&

Elements to Consider

Domains (sets of values), e.g., $\mathbb{Z}$, $\mathbb{R}$, $\{1, 2, 3, 4, 5\}$, $[1, 10)$, $\{\text{"foo", "bar", "loop"}\}$. Most common type is (finite) subsets of $\mathbb{Z}$.

Variables that range over domains, e.g., $v \in \{1, 2, 3, 4, 5\}$ denotes that $v$ can take any discrete value from 1 to 5.

Constraints that define a set of valid combinations of values for a set of variables. E.g., the set of valid combinations for $x < y$, where $x, y \in \{1, 2, 3\}$, is $\{(1, 2), (1, 3), (2, 3)\}$.

Search methods that specify how to instantiate the variables.

The trick is to combine these elements for a particular problem!

Example 1: Send More Money

Assign distinct values to the variables $s, e, n, d, m, o, r, y$ such that the equation

\[
\begin{align*}
\text{send} + \text{more} &= \text{money}
\end{align*}
\]

holds.
Example 1 (continued)

Domains: \{0, \ldots, 9\}

Variables: \(s, e, n, d, m, o, r, y\) \(\in\) \{0, \ldots, 9\}

Constraints:
\[
\begin{align*}
s & \neq e, s \neq n, s \neq d, \ldots, y \neq m, y \neq o, y \neq r, \\
1000 \times s + 100 \times e + 10 \times n + d & + \\
1000 \times m + 100 \times o + 10 \times r + e & = \\
10000 \times m + 1000 \times o + 100 \times n + 10 \times e + y, \\
s & \neq 0, \\
m & \neq 0
\end{align*}
\]

The above is then a constraint-based model of the problem.

Solving the Problem: First Attempt

We need to find a solution, i.e., an assignment to the variables \(s, e, n, d, m, o, r, y\) from the domain \{0, \ldots, 9\} such that all the constraints are satisfied.

A simple and naïve way of doing this is generate and test: Try all combinations of assignments in some systematic way until a satisfying one is found.

This is very inefficient! There are \(10^8\) such combinations which makes the search tree huge.

And this is just a toy problem!

Solving the Problem: Second Attempt

In a constraint programming framework, the constraints are active entities that try to remove values from the variables and thus exclude certain combinations automatically.

This is equivalent to removing branches in a search tree, shrinking the search space drastically.

For instance, the constraints \(s \neq 0\) and \(m \neq 0\) may prevent \(s\) and \(m\) from taking the value 0 before any search has started.
**Constraint Propagation**

Constraint propagation means reasoning about the properties of a constraint and how that affects its variables.

A value of a variable that does not fulfil a given property may be removed.

As an example, are there any values in \( x, y \in \{1, \ldots, 3\} \) that may be removed given the constraint \( x < y \)?

**Constraint Store**

The constraint store contains information about variables, their domains, and constraints:

- \( s \in \{0, \ldots, 9\} \)
- \( m \neq 0 \)
- \( s \neq c \)
- \( y \neq m \)
- \( r \in \{0, \ldots, 9\} \)
- \( e \in \{0, \ldots, 9\} \)
- \( x \neq d \)
- \( y \neq r \)
- \( y \in \{0, \ldots, 9\} \)
- \( s \neq n \)
- \( y \neq o \)
- \( 1000 \times x + 100 \times e + 10 \times n + d + \ldots = \ldots \)
**Constraint Store**

Being initially empty, the basic operation is to add constraints to it. This is called *telling* or *posting*.

\[
\begin{align*}
 s &= 3 \\
 e &= 0
\end{align*}
\]

**Consistency**

- Constraint programming relies heavily on consistency techniques for solving problems efficiently.
- This is a way of reasoning about allowed and disallowed values of the variables of a constraint.
- There are many different kinds of consistency.
- I will only give it an intuitive meaning here.

**Example 2: Consistency**

Consider the variables \( x, y \) and \( z \) each with the domain \( \{0, 1, \ldots, 10\} \) and the single constraint:

\[ x + y = z \]

For any variable \( \text{var} \in \{x, y, z\} \) and any value \( \text{val} \in \{0, 1, \ldots, 10\} \) there is a solution to the constraint that includes \( \text{var} = \text{val} \).

\[
\begin{align*}
 x, y, z &\in \{0, 1, \ldots, 10\}, \quad x + y = z
\end{align*}
\]

- For any value of \( x \), set \( y \) to 0 and \( z \) to \( x \).
- For any value of \( y \), set \( x \) to 0 and \( z \) to \( y \).
- For any value of \( z \), set \( x \) to 0 and \( y \) to \( z \).
Example 3: More Consistency

Consider again the variables $x$ and $y$ each with the domain $\{0, 1, \ldots, 10\}$. Now with the single constraint:

$$x < y$$

For any variable $\text{var} \in \{x, y\}$ and any value $\text{val} \in \{0, 1, \ldots 10\}$ there is a solution to the constraint that includes $\text{var} = \text{val}$.

Is this still true?

Consistent vs. Inconsistent Constraints

- Given $x, y, z \in \{0, 1, \ldots, 10\}$:
  - the constraint $x + y = z$ is consistent.
  - the constraint $x < y$ is inconsistent.

- Question: How can we make an inconsistent constraint consistent?

- Answer: Remove the values that make it inconsistent!

Example 3 (continued)

No! Take a look at the following:

- If $x = 10$ there is no value for $y$ such that $x < y$.
- If $y = 0$ there is no value for $x$ such that $x < y$.

Removing the Inconsistent Values

The inconsistent values are: $x = 10$ and $y = 0$.

By removing them, we obtain the following domains:

- $x \in \{0, \ldots, 9\}$
- $y \in \{1, \ldots, 10\}$

And with respect to these domains the constraint $x < y$ is consistent!
Now What About the Other Constraint?

\[ x \in \{0, \ldots, 9\}, y \in \{1, \ldots, 10\}, z \in \{0, \ldots, 10\}, x + y = z \]

Is this still consistent given the new domains?

No! If \( z = 0 \) there are no values for \( x \) and \( y \) such that \( x + y = z \).

By again removing any inconsistent value we end up with the domains \( x \in \{0, \ldots, 9\} \), \( y \in \{1, \ldots, 10\} \) and \( z \in \{1, \ldots, 10\} \).

Propagators

- A constraint is represented by one or more propagators.
- An activated propagator narrows the domains of the variables until its constraint is consistent.
- Propagators are activated by the constraint store.
- Activation of a propagator may (re)activate other propagators.

Example 4: Propagators and the Store

Initial state of the store:

Initial state of the store:

Propagating \( x + y = z \) results in no change to the store:

Example 4 (continued)

Propagating \( x + y = z \) results in no change to the store:
**Example 4 (continued)**

Propagating $x < y$ changes the store:

$$\begin{align*}
\text{Constraint Store} \\
x \in \{0, \ldots, 9\} & \quad x + y = z \\
y \in \{1, \ldots, 10\} & \quad x < y \\
z \in \{1, \ldots, 10\} & \quad x < y
\end{align*}$$

Which in turn activates $x + y = z$ again:

$$\begin{align*}
\text{Constraint Store} \\
x \in \{0, \ldots, 9\} & \quad x + y = z \\
y \in \{1, \ldots, 10\} & \quad x < y \\
z \in \{1, \ldots, 10\} & \quad x < y
\end{align*}$$

**Final state of the store:**

$$\begin{align*}
\text{Constraint Store} \\
x \in \{0, \ldots, 9\} & \quad x + y = z \\
y \in \{1, \ldots, 10\} & \quad x < y \\
z \in \{1, \ldots, 10\} & \quad x < y
\end{align*}$$

**Entailed Constraints**

If a constraint will stay consistent no matter what is done to its variables’ domains we say that it is **entailed**.

For example, given $x \in \{1, \ldots, 3\}$ and $y \in \{4, \ldots, 6\}$, the constraint $x < y$ is entailed since any assignment to $x$ and $y$ satisfies it and this is also true for any subsets of the domains of $x$ and $y$. 
Failed Constraints

If a constraint will stay inconsistent no matter what is done to its variables’ domains we say that it is failed.

For example, given \( x \in \{4, \ldots, 6\} \) and \( y \in \{1, \ldots, 4\} \), the constraint
\[
x < y
\]
is failed since no assignment to \( x \) and \( y \) satisfies it.

More About the Constraint Store

The constraint store activates the propagators of all the constraints until no more domain narrowing is possible.

When this is the case we say that a fixed point has been reached.

An entailed constraint is removed from the store.

A failed constraint signals failure to solve the problem.

Consistency is Not Enough

• We usually need to introduce more information in the constraint store to solve a problem.
• We do this by searching for assignments of values to variables.
• Let us take a look at backtrack search in the presence of a constraint store.

Search

A classical search procedure:

While there is at least one variable with a non-singleton domain:

Pick a variable \( x \) whose domain \( D \) has at least 2 elements.
Pick a value \( d \in D \).
Post an additional constraint, called a decision, e.g., \( x = d \), or \( x \neq d \), or \( x > d \), or \( x \leq d \).
Backtrack Search for $x < y$ with Constraint Store

Some values are removed even before search starts:

Adding the constraint $x = 1$ to the store:

The next choice is $x = 2$ since $x = 1$ was already removed:
Backtrack Search for \( x < y \) with Constraint Store

The next choice is \( x = 3 \):

Upon backtrack the changes to the store are undone:

Now the next choice for \( x \) is 2:

The propagator of \( x < y \) is activated and \( y = 3 \) is the only choice:
Backtrack Search for $x < y$ with Constraint Store

We backtrack again:

And the last value for $x$ was already removed by the propagator of $x < y$.

In SICStus Prolog

The query:

```
?- X in 1..3, Y in 1..3, X #< Y.
```

gives the answer:

```
X in 1..2,
Y in 2..3 ?
```

Adding the constraint $X = 1$:

```
?- X in 1..3, Y in 1..3, X #< Y, X = 1.
```

gives the answer:

```
X = 1,
Y in 2..3 ?
```
In SICStus Prolog

Adding the constraint \( Y = 2 \):

\[
| ?- X \text{ in } 1..3, Y \text{ in } 1..3, X \#< Y, X = 1, Y = 2. 
\]

gives the expected answer:

\[
X = 1, \\
Y = 2 ?
\]

Adding the constraint \( X = 2 \):

\[
| ?- X \text{ in } 1..3, Y \text{ in } 1..3, X \#< Y, X = 2. 
\]

gives the answer:

\[
X = 2, \\
Y = 3 ?
\]

Search until a solution with the labeling/2 predicate:

\[
| ?- X \text{ in } 1..3, Y \text{ in } 1..3, X \#< Y, \\
labeling([], [X,Y]).
\]

gives us directly:

\[
X = 1, \quad \text{and} \quad X = 1, \quad \text{and} \quad X = 2, \\
Y = 2 ?; \quad \text{and} \quad Y = 3 ?; \quad \text{and} \quad Y = 3 ?
\]

More Powerful (Global) Constraints

- Constraints remove branches of the search tree.
- Removing branches as high up in the search tree as possible is often preferred.
- Constraint programming relies heavily on the notion of global constraints.
Global Constraints

- A basic constraint operates on a fixed number of variables.
  Examples: \( x < y \) or \( x \neq y \) operates on 2 variables.

- A global constraint is stated on any number of variables.
  Examples: all different \( [x_1, x_2, x_3, x_4] \) operates on \( n = 4 \) variables which must take different values.

\( \text{at}\_\text{most}(n, e, [x_1, x_2, x_3, x_4]) \) requires that there are at most \( n \) occurrences of \( e \) in the list of 4 variables.

The all different \( (x_1, x_2, x_3, x_4) \) Constraint

Semantically the constraint is the same as the \( \frac{n(n-1)}{2} = 6 \) basic constraints:

\[
\begin{align*}
x_1 \neq x_2, & \quad x_1 \neq x_3, \quad x_1 \neq x_4, \quad x_2 \neq x_3, \quad x_2 \neq x_4, \quad x_3 \neq x_4
\end{align*}
\]

Operationally it is much more effective in removing inconsistent values from the variables due to its more global scope.

Example: Consider the domains \( x_1 \in \{2, 3\}, x_2 \in \{2, 3\}, x_3 \in \{1, 3\}, x_4 \in \{1, 2, 3, 4\} \)

Send More Money in SICStus Prolog

Remember our model:

Variables: \( s, e, n, d, m, o, r, y \in \{0, \ldots, 9\} \)

Constraints:

\[
\begin{align*}
s \neq e, \quad & s \neq n, \quad s \neq d, \ldots, \quad y \neq m, \quad y \neq o, \quad y \neq r, \\
& 1000 \times s + 100 \times e + 10 \times n + d + \\
& 10000 \times m + 100 \times o + 10 \times r + e \\
& = 10000 \times m + 100 \times o + 10 \times n + 10 \times e + y,
\end{align*}
\]

\( s \neq 0, \quad m \neq 0 \)

SendMoreMoney([S,E,N,D,M,O,R,Y]) :- domain([S,E,N,D,M,O,R,Y], 0, 9),
S #\neq E, S #\neq N, S #\neq D, S #\neq M, S #\neq O, S #\neq R, S #\neq Y,
E #\neq N, E #\neq D, E #\neq M, E #\neq O, E #\neq R, E #\neq Y,
... 0 #\neq R, 0 #\neq Y,
R #\neq Y,
1000\times S + 100\times E + 10\times N + D + \\
1000\times M + 100\times O + 10\times R + E \\
#\neq 10000\times M + 1000\times O + 100\times N + 10\times E + Y,
S #\neq 0, M #\neq 0,
labeling([], [S,E,N,D,M,O,R,Y]).
**Declaring the Variables**

```prolog
:- use_module(library(clpfd)).

smm([S,E,N,D,M,O,R,Y]) :- domain([S,E,N,D,M,O,R,Y], 0, 9),
 S #\= E, S #\= N, S #\= D, S #\= M, S #\= O, S #\= R, S #\= Y,
 E #\= N, E #\= M, E #\= O, E #\= R, E #\= Y,
 ... 0 #\= R, 0 #\= Y,
 R #\= Y,
 1000*S + 100*E + 10*N + D +
 1000*M + 100*O + 10*R + E
 #= 10000*M + 1000*O + 100*N + 10*E + Y,
 S #\= 0, M #\= 0,
 labeling([], [S,E,N,D,M,O,R,Y]).
```

**Posting the Constraints**

```prolog
:- use_module(library(clpfd)).

smm([S,E,N,D,M,O,R,Y]) :- domain([S,E,N,D,M,O,R,Y], 0, 9),
 S #\= E, S #\= N, S #\= D, S #\= M, S #\= O, S #\= R, S #\= Y,
 E #\= N, E #\= M, E #\= O, E #\= R, E #\= Y,
 ... 0 #\= R, 0 #\= Y,
 R #\= Y,
 1000*S + 100*E + 10*N + D +
 1000*M + 100*O + 10*R + E
 #= 10000*M + 1000*O + 100*N + 10*E + Y,
 S #\= 0, M #\= 0,
 labeling([], [S,E,N,D,M,O,R,Y]).
```

**Searching for a Solution**

```prolog
:- use_module(library(clpfd)).

smm([S,E,N,D,M,O,R,Y]) :- domain([S,E,N,D,M,O,R,Y], 0, 9),
 S #\= E, S #\= N, S #\= D, S #\= M, S #\= O, S #\= R, S #\= Y,
 E #\= N, E #\= M, E #\= O, E #\= R, E #\= Y,
 ... 0 #\= R, 0 #\= Y,
 R #\= Y,
 1000*S + 100*E + 10*N + D +
 1000*M + 100*O + 10*R + E
 #= 10000*M + 1000*O + 100*N + 10*E + Y,
 S #\= 0, M #\= 0,
 labeling([], [S,E,N,D,M,O,R,Y]).
```

**Running Send More Money**

```prolog
| ?- smm([S,E,N,D,M,O,R,Y]).
D = 7,
E = 5,
M = 1,
N = 6,
O = 0,
R = 8,
S = 9,
Y = 2 ? ;
no
% source_info
| ?-
### Running Send More Money

```prolog
| ?- smm([S,E,N,D,M,O,R,Y]).
D = 7,
E = 5,
M = 1,
N = 6,
O = 0,
R = 8,
S = 9,
Y = 2.
```

And indeed...

```
9567 + 1085 = 10652
```

### Send More Money: Another Model?

- A given problem can usually be modelled in different ways.
- Different models may imply different performance.

### Send More Money with Carry Variables

**Variables:** 
- `s, e, n, d, m, o, r, y ∈ {0, ..., 9}`
- `c_1, c_2, c_3 ∈ {0, 1}`

**Constraints:**
- `all_different([s, e, n, d, m, o, r, y])`
- `c_3 c_2 c_1`
- `+ s e n d`
- `m o r e`
- `m o n e y`

### Carry Model in SICStus Prolog

```prolog
:- use_module(library(clpfd)).

smm([S,E,N,D,M,O,R,Y]):- domain([S,E,N,D,M,O,R,Y], 0, 9),
domain([C1,C2,C3], 0, 1),
all_different([S,E,N,D,M,O,R,Y]),
D + E #= 10*C1 + Y,
C1 + N + R #= 10*C2 + E,
C2 + E + O #= 10*C3 + N,
C3 + S + M #= 10*M + O,
S \#= 0,
M \#= 0,
labeling([], [S,E,N,D,M,O,R,Y]).
```
Evaluation of the Different Models

- Number of backtracks in labeling for the model with one equation: 1
- Number of backtracks in labeling for the model with carry variables: 3

So should we always try to not use carry variables?

Example 5: NQueens

Given is a chess board and 8 queens. Find a way to place the 8 queens on the chess board such that no two queens attack each other following the rules of chess.

DONALD + GERALD = ROBERT

Very similar puzzle: Find values to the variables D, O, N, A, L, G, E, R, B, T such that DONALD + GERALD = ROBERT.

- Number of backtracks in labeling for the model with one equation: 303
- Number of backtracks in labeling for the model with carry variables: 134

Hmm...

Example 5: Generalisation

The problem can be generalised to that of finding a way to place \( n \) queens on an \( n \times n \) chess board with the same conditions:

1. No two queens on the same row.
2. No two queens on the same column.
3. No two queens on the same NorthWest – SouthEast diagonal.
4. No two queens on the same NorthEast – SouthWest diagonal.
Example 5: A Possible Model

We let $q_i$ denote the row of the queen placed in column $i$. Then we may state the constraints as follows:

Variables: $q_1, \ldots, q_n \in \{1, \ldots, n\}$

Constraints:
1. $\forall i \neq j \in \{1, \ldots, n\} : q_i \neq q_j$
2. Already handled by the choice of variables.
3. $\forall i < j \in \{1, \ldots, n\} : q_i - i \neq q_j - j$
4. $\forall i < j \in \{1, \ldots, n\} : q_i - j \neq q_j - i$

Example 6: Magic Square

A magic square of order $n$ is an $n \times n$ matrix containing the numbers 1 to $n^2$ with each row, column and main diagonal equal the same sum.

A magic square of order 4

\[
\begin{array}{ccc}
8 & 11 & 6 \\
13 & 7 & 10 & 4 \\
12 & 14 & 3 & 5 \\
1 & 2 & 15 & 16 \\
\end{array}
\]

where each row, column and main diagonal sum up to 34.
Example 7: Round-Robin Tournament Scheduling

The problem is to schedule games between \( n \) teams over \( n - 1 \) weeks. Each week is divided into \( n/2 \) periods. We want to find a schedule of play for the \( n \) teams such that the following constraints are satisfied:

1. Every team plays exactly once a week.
2. Every team plays at most two times in the same period over all weeks.
3. Every team plays every other team exactly once.

Example 7: A Solution with 8 Teams

<table>
<thead>
<tr>
<th>Period 1</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 vs. 1</td>
<td>0 vs. 2</td>
<td>4 vs. 7</td>
<td>3 vs. 6</td>
<td>3 vs. 7</td>
<td>1 vs. 5</td>
<td>2 vs. 4</td>
</tr>
<tr>
<td>Period 2</td>
<td>2 vs. 3</td>
<td>1 vs. 7</td>
<td>0 vs. 3</td>
<td>5 vs. 7</td>
<td>1 vs. 4</td>
<td>0 vs. 6</td>
<td>5 vs. 6</td>
</tr>
<tr>
<td>Period 3</td>
<td>4 vs. 5</td>
<td>3 vs. 5</td>
<td>1 vs. 6</td>
<td>0 vs. 4</td>
<td>2 vs. 6</td>
<td>2 vs. 7</td>
<td>0 vs. 7</td>
</tr>
<tr>
<td>Period 4</td>
<td>6 vs. 7</td>
<td>4 vs. 6</td>
<td>2 vs. 5</td>
<td>1 vs. 2</td>
<td>0 vs. 5</td>
<td>3 vs. 4</td>
<td>1 vs. 3</td>
</tr>
</tbody>
</table>

More constraints could be added:

- Each team plays home and away equal number of times. (Double Round-Robin Tournaments)
- Teams \( x \) and \( y \) cannot play directly after \( x \) played \( z \).
- ...