**Constraint Programming**

Magnus Ågren
agren@csd.uu.se

Thanks to Mats Carlsson at SICS!

---

**Examples of Constraints**

- The coding starts when the specification is finished.
- The meetings cannot overlap in time.
- After two night shifts one must have one day of rest.
- I must be at the airport 2 hours before departure.
- I must study for the LP exam if and only if I must pass the LP exam.

---

**Constraint Technology**

- Constraints
  - A constraint is a logical relation between a number of unknown variables.
- Constraint Programming (CP)
  - Framework for solving certain problem classes based on constraints
- CP over Finite Domains (CP(FD))
  - Each variable can take a value from a finite discrete domain. Usually a set of integers.

---

**Constraint Satisfaction Problems**

The problem classes that we are interested in are captured by the concept of Constraint Satisfaction Problems (CSPs).

A CSP is a 3-tuple \((V, D, C)\) where:

- \(V = \{V_1, \ldots, V_n\}\) is a set of variables.
- \(D = \{D_1, \ldots, D_n\}\) is a set of finite domains, where \(D_i\) is the domain of variable \(V_i\).
- \(C = \{C_1, \ldots, C_m\}\) is a set of constraints, each being defined on a subset of the variables in \(V\), specifying the valid combinations for those.

Solving CSPs is in general an NP-complete task!
Problem Classes

- Hardware design
- Compilation
- Financial problems
- Layout problems
- Cutting problems
- Stand allocation
- Air traffic control
- Frequency allocation
- Configuration
- Production scheduling
- Satellite
- Maintenance
- Product blending
- Time tabling
- Air-crew rostering
- Logistics
- Personnel requirement
- Sport scheduling

Solving a CSP

Solving a CSP using Constraint Programming is usually done in the following way:

1. From your problem description, identify the variables in the problem.
2. Identify the set of values each variable can take.
3. Identify how to best express the conditions of the problem using the available constraints.
4. Find out a good labelling strategy, i.e., how to enumerate the variables in an efficient way.
5. Possibly iterate and optimise.

A Simple Example (in SICStus Prolog)

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

top([X,Y,Z]) :-
    domain([X,Y,Z],1,4),
    X+Y #< Z,
    labeling([], [X,Y,Z]).
```

A Simple Example (in SICStus Prolog)

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

top([X,Y,Z]) :-
    domain([X,Y,Z],1,4),  % Domain declaration
    X+Y #< Z,
    labeling([], [X,Y,Z]).
    X:1..4
    Y:1..4
    Z:1..4
```
A Simple Example (in SICStus Prolog)

:- use_module(library(clpfd)).
top([X,Y,Z]) :-
    domain([X,Y,Z],1,4),
    X + Y #< Z, \textit{Stating a constraint}
    labeling([],[X,Y,Z]).

A Simple Example (in SICStus Prolog)

:- use_module(library(clpfd)).
top([X,Y,Z]) :-
    domain([X,Y,Z],1,4),
    X + Y #< Z, \textit{Stating a constraint}
    labeling([],[X,Y,Z]).
Domain Declaration

domain([X,Y,Z],1,4)
or
X in 1..4, Y in 1..4, Z in 1..4

• Normally before any constraints are posted
• You need to consider:
  - What are the variables of my problem?
  - What values should the variables take?

Stating a Constraint

X + Y #< Z

• Similar to stating a predicate in Prolog.
• If a solution is returned then all posted constraints hold.
• You need to consider:
  - How to express a given condition in a good way?
  - This is not always obvious!

Enumerate the Variables

labeling([], [X,Y,Z]).

• Default strategy (in SICStus) is to first choose the leftmost unassigned variable and assign to it its minimum value.
• A popular and often efficient strategy is to pick the variable with the smallest domain first since it is most probable to fail.
• Application specific enumeration strategies are often needed.

Search Tree

• Constraint Programming uses global search.
• Each variable in the problem is assigned a value until some constraint is inconsistent. This gives rise to a search tree.
• The constraints are activated at each node in the search tree and enforce local consistency.
• An inconsistent constraint means backtracking.
• This process is called constraint propagation and is the strength of CP.
Constraint Propagation

\[ X + Y = 9 \]
\[ 2X + 4Y = 24 \]

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Constraint Propagation

\[ X + Y = 9 \]
\[ 2X + 4Y = 24 \]

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Constraint Propagation

\[ X + Y = 9 \]
\[ 2X + 4Y = 24 \]

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Constraint Propagation

\[ X + Y = 9 \]
\[ 2X + 4Y = 24 \]

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
Constraint Propagation

\[ X + Y = 9 \]
\[ 2X + 4Y = 24 \]

21

Constraint Propagation

\[ X + Y = 9 \]
\[ 2X + 4Y = 24 \]

True

22

Constraint Propagation

\[ X + Y = 9 \]
\[ 2X + 4Y = 24 \]

\[ x \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \]
\[ y \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \]

23

Operational View of a Constraint

Program code is used as a coroutine for:

1. Checking if the condition is **for sure false**:
   \[ \Rightarrow \text{backtracking} \]
2. Checking if the condition is **for sure true**:
   \[ \Rightarrow \text{entailment} \]
3. Removing values from variables that **for sure cannot be in any solution**:
   \[ \Rightarrow \text{suspend and start enumerate} \]

Usually **polynomial** but **incomplete** algorithms.
Checking If For Sure False

\textbf{domain([X,Y,Z],[1,4]), X+Y \#< Z}

(1) Evaluate \textbf{minimum} value of X+Y \\
\text{min}(X+Y) = \text{min}(X)+\text{min}(Y) = 1+1 = 2 \\
(2) Evaluate \textbf{maximum} value of Z \\
\text{max}(Z) = 4 \\
(3) \textbf{Compare}: \\
Since minimum value of X+Y is not greater than or equal to the maximum value of Z we cannot conclude that X+Y \#< Z is for sure false.

Checking If For Sure True

\textbf{domain([X,Y,Z],[1,4]), X+Y \#< Z}

(1) Evaluate \textbf{maximum} value of X+Y \\
\text{max}(X+Y) = \text{max}(X)+\text{max}(Y) = 4+4 = 8 \\
(2) Evaluate \textbf{minimum} value of Z \\
\text{min}(Z) = 1 \\
(3) \textbf{Compare}: \\
Since maximum value of X+Y is not less than minimum value of Z we cannot conclude that X+Y \#< Z is for sure true.

Removing Values

\textbf{domain([X,Y,Z],[1,4]), X+Y \#< Z}

(1) Evaluate \textbf{maximum} value of X and Y \\
\text{max}(X) \leq \text{max}(Z)-\text{min}(Y)-1 = 4-1-1 = 2 \\
\text{max}(Y) \leq \text{max}(Z)-\text{min}(X)-1 = 4-1-1 = 2 \\
(2) Evaluate \textbf{minimum} value of Z \\
\text{min}(Z) \geq \text{min}(X)+\text{min}(Y)+1 = 1+1+1 = 3 \\
\begin{align*}
X & : 1 \ldots 2 \\
Y & : 1 \ldots 2 \\
Z & : 3 \ldots 4
\end{align*}

Optimisation

In some problems you might want to find an \textbf{optimal assignment} with respect to a given cost function.

This could be \textbf{minimising} the number of flights for a travelling business woman

or

\textbf{maximising} the number of produced items in a production line.

This means that you need to search all solutions in a clever way. For example Branch-And-Bound.
Global Constraints

- Work on a set of variables
  - Global conditions
- Strong necessary conditions
  - Geometry, graph theory, network flow
- Components
  - Generic, multi-usage, strong propagation

Global Constraint: alldifferent

\[ \text{all_distinct}(Vars) \]

\textit{Vars} is a list of finite domain variables. Constraint holds iff all elements in \textit{Vars} take distinct values.

\text{domain}([X,Y,Z],1,2), \text{all_distinct}([X,Y,Z]).

is semantically the same as

\text{domain}([X,Y,Z],1,2), X \neq Y, X \neq Z, Y \neq Z.

But only the former one will detect inconsistency!

Global Constraint: element

\[ \text{element}(index, \text{Table}, \text{Value}) \]

dVar dVar list dVar

Constraint holds iff the variable at position \textit{index} in \textit{Table} takes the value \textit{Value}.

\[ X \text{ in } 1..5, \ Y \text{ in } 1..6, \ element([X,4,8,1,4,2,0],Y), \]

\[ \text{labeling}([X,Y]). \]

Global Constraint: cumulative

\[ \text{cumulative}(\text{Origins}, \text{Durations}, \text{Resources}, \text{Limit}) \]

dVarList dVarList dVarList dVar

Constraint holds iff \textit{Limit} is never exceeded at any point in time.

\[ \quad \]

\[ \quad \]

\[ \quad \]
Case Study: Nqueens

- Classical puzzle
- There are n queens each acting as in the game of chess.
- Is there a way of placing the N queens on a chessboard of size n*n such that no two queens attack each other?

A model in SICStus:

Variables: Let the value of variable Qi denote the row of the queen in column i.

Domains: Each variable can then take values in the range 1..n.

Constraints: No two queens can be on the same row: all_distinct([Q1,...,Qn])

No two queens can be in the same column: already handled

No two queens can be on the same diagonal: |Qi-Qj| ≠ |i-j|

Case Study: Project Scheduling

You want to schedule a set of tasks, each task demanding a number of persons to be completed, and such that the precedence constraints below are respected. You only have 7 persons at your disposal.

Since time is money you want to find the earliest end.
Case Study: Project Scheduling

```
project :-
D1=2, D2=1, D3=4, D4=2, D5=3, D6=1, D7=0,
R1=1, R2=3, R3=3, R4=2, R5=4, R6=6,
Total is D1+D2+D3+D4+D5+D6+D7,
L = [S1,S2,S3,S4,S5,S6,S7],
domain([l,0,Total]),
S1+D1 #=< S2, S1+D1 #=< S3,
S1+D1 #=< S4, S2+D2 #=< S5,
S3+D3 #=< S6, S4+D4 #=< S5,
S5+D5 #=< S7, S6+S6 #=< S7,
cumulative([[S1,S2,S3,S4,S5,S6],
    [D1,D2,D3,D4,D5,D6],
    [R1,R2,R3,R4,R5,R6],
    7]),
labeling([minimize(S7)].L),
write(sol(S7,L)), nl.
```

Contributing Areas

- Artificial Intelligence
  - Constraint Networks, consistency.
- Logic Programming
  - Prolog, nondeterministic search.
- Discrete Mathematics
  - Graph theory, combinatorics, group theory.
- Operations Research
  - Modelling languages.
- Algorithms and Data Structures
  - Data structures, incremental algorithms.

History

- ALICE (J.L. Laurière) 1976
- CHIP (ECRC) 1987-1990
- Libraries 1990-
  - C++ Ilog, CHIP, Figaro
  - Java Koolog, JCL, MINERVA
  - Prolog SICStus, ECLIPSê, GNU
  - CLAIRE ECLAIR, CHOCO
  - OZ Mozart
  - O’Caml Figaro
  - Lisp, Python Screamer, ...

Constraint Programming in Uppsala

- At Uppsala University
  - Symmetry in Constraint Programs
    - Breaking symmetry
    - Detecting symmetry
  - Modelling
    - Solving problem at higher levels
    - Reducing the modelling effort for the users
    - Combining different solving methods
- At Swedish Institute of Computer Science (SICS)
  - Global Constraints
    - Good and relevant abstractions
    - Classification of Global Constraints
    - Algorithms & Debugging
Constraint Course This Summer

When? Week 24 - 27

Where? Polacksbacken

How? Speed 100%
   Lectures 2-3 days a week
   Fun assignments! Written exam!

Sign up before March 15!