Beyond Prolog: Constraint Logic Programming

This lecture will cover:
- introduce the idea of constraints and constraint satisfaction;
- introduce programming with constraints over several domains;
- look at the use of CLP(R);
- review applications of CLPs.

Constraint satisfaction

Constraint programming
About formulating and solving problems that are defined in terms of constraints among a set of variables.

Constraint satisfaction
Solving the problem involves finding combinations of values that satisfy the constraints.

Constraint logic programming combines constraints with logic programming.

Constraint satisfaction – an example

Tasks
1. Task a – 2 hours – finishes before b and c
2. Task b – 3 hours – finishes before d
3. Task c – 5 hours
4. Task d – 4 hours

Find the minimal time to complete all the tasks.

Constraint satisfaction – an example - 1

Variables: Ta, Tb, Tc, Td, Tfinish
Domain: Non-negative real numbers

Constraints:
1. Ta + 2 ≤ Tb
2. Ta + 2 ≤ Tc
3. Tb + 3 ≤ Td
4. Tc + 5 ≤ Tfinish
5. Td + 4 ≤ Tfinish
How are constraints satisfied?

We can draw the relationships between each variable:

```
Ta +2 <= Tb
Tb +3 <= Td
Td +4 <= Tfinish
```

We move round the graph, adjusting the values of the variables.

If Ta is [0,1,2,3,4,5,6,7,8,9,10] then Tb must be adjusted to be consistent with Ta:
As Ta +2 <= Tb, Tb is [2,3,4,5,6,7,8,9,10]

The process of adjusting the domain of variables by traversing arcs is called arc consistency.

Changing the domain of one value may propagate changes in other the domain of other variables.

But it is not this is not the whole search.

Each variable has a domain, but it doesn’t mean that any and all combinations of values from the domains are valid solutions.

We need more search – e.g. pick a value from a domain with the narrowest range and see if that value can be propagated round the network.

Success is either one value for each variable – or all domains are non-empty but >=1 variable has multiple members in its domain.
How are constraints satisfied?

<table>
<thead>
<tr>
<th>Arc</th>
<th>Ta</th>
<th>Tb</th>
<th>Tc</th>
<th>Td</th>
<th>Tfinish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tb,Ta</td>
<td>0..10</td>
<td>0..10</td>
<td>0..10</td>
<td>0..10</td>
<td>0..10</td>
</tr>
<tr>
<td>Tb,Tb</td>
<td>2..10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Td,Tb</td>
<td></td>
<td>5..10</td>
<td>9..10</td>
<td>5..6</td>
<td></td>
</tr>
<tr>
<td>Td,Ty</td>
<td></td>
<td>2..10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ta,Tb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0..1</td>
</tr>
<tr>
<td>Ta,Ty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2..5</td>
</tr>
<tr>
<td>Tc,Ta</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What does the programmer have to do?

Specify the constraints in the program.

The search happens automatically when the program is executed.

If a rule changes the domain of one variable, the arc consistency algorithm will change the domains of any other variables – without the programmer doing anything.

An alternative to unification

Prolog’s search is good because it allows alternatives to be explored.

Prolog uses a very strict notion of “constraint” – i.e. matching through unification

This is often perfect for the problem in hand, but sometimes not adequate – as with the Sudoku solver.

Constraint Logic Programming

Replaces unification with constraint satisfaction.

Essentially, there is a global store of constraints.

Each rule that contains constraints produces its own list of current constraints.

Global and current constraints are merged – and domains of variables are made consistent or the program fails (if inconsistent).

Kinds of CLP

There are several types of CLP system – written CLP(X) where X denotes the type of the domain.

CLP(R) – domain = real numbers; constraints are arithmetic equalities, inequalities and disequalities over real numbers;

CLP(B) – boolean domain;

CLP(FD) – user-defined finite domains; constraints are essentially membership-related; and

CLP(Q) - rational numbers; CLP(Z) - integers
The idea of constraints

Imagine if we could type equations which the programming language would solve:

Programming with constraints - 1

We can use CLP(R) to work with constraints over real numbers.

A common example is converting Fahrenheit to Celsius and vice versa (in Prolog):

```
convert_clpr(Celsius, Fahrenheit) :-
    { Celsius = (Fahrenheit - 32) * 5 / 9 }.
```

Programming with constraints - 2

In Prolog:
```
convert_pl(Celsius, Fahrenheit) :-
    Celsius is (Fahrenheit - 32) * 5 / 9.
```

Programming with constraints - 3

How do the two differ in execution?

```
In CLP(R):
convert_clpr(Celsius, Fahrenheit) :-
    { Celsius = (Fahrenheit - 32) * 5 / 9 }.
```

Programming with constraints - 4

So CLP(R) allows us to work with expressions as constraints, rather than in terms of “executing arithmetic”.

We can express constraints that can be resolved when more information becomes available – or never resolved beyond a range of values.

```
In Prolog:
convert_pl(Celsius, Fahrenheit) :-
    Celsius is (Fahrenheit - 32) * 5 / 9.
```

Can similar ideas be applied to the Sudoku puzzle?

That’s the next lecture and an insight in CLP(FD).

Programming with constraints - 5

Can similar ideas be applied to the Sudoku puzzle?
What can CLP be used for?

CLP(Q, R) is used for:
- scheduling (e.g. tasks)
- simulating (e.g. electronic circuits)
- verification
- error diagnosis

What can CLP be used for?

CLP(FD) is used for:
- optimization
- planning
- layout configuration

Examples include:
optimizing containers in harbours;
planning car production plants;
layout of wireless network components
university timetabling.

CHR?

CHR
- stands for Constraint Handling Rules
- is a general language for writing constraint solvers

You have seen it or will see it:
- as the inference engine in the NL interface to the Tudor family database;
- as the central store controller in an implementation of a parallel logic programming language.