Lecture 17 - Programs and Summary

This lecture presents a direction in which logic programming/Prolog has developed. Constraint Logic Programming offers an alternative way of searching for solutions in search spaces where there are many interacting factors.

The idea of constraints and constraint satisfaction
Constraint programming is a large field within mathematics. It is about formulating and solving problems that are defined in terms of constraints among a set of variables. *Constraint satisfaction* is about solving the problem by finding combinations of values that satisfy the constraints. *Constraint logic programming* combines constraint satisfaction with logic programming.

A constraint satisfaction problem has three parts:

1. a set of variables;
2. a domain from which variables take values, for instance a range of integers;
3. constraints that the variables have to satisfy.

The task is to find an assignment of values to the variables such that all constraints are satisfied. There may be more than one assignment and, if so, it can be part of the task to find the optimal assignment.

Consider this simple constraint satisfaction problem:

There are four tasks necessary to finish a job:

1. Task a - takes 2 hours - must finish before b and c
2. Task b - takes 3 hours - must finish before d
3. Task c - takes 5 hours
4. Task d - takes 4 hours

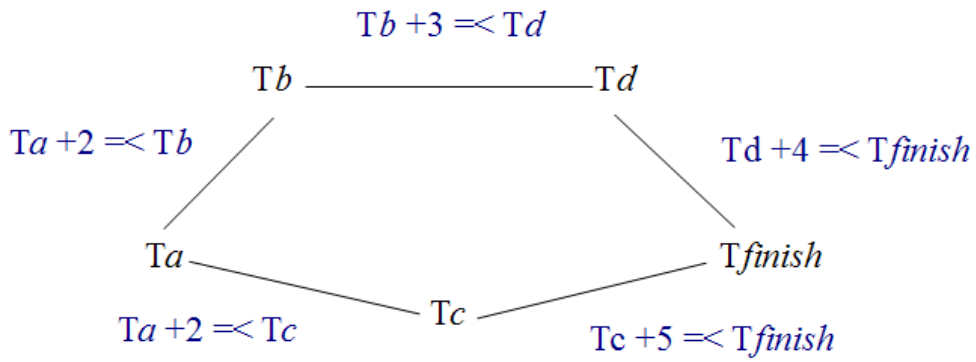
This can be described as:

Variables:	T*a*, T*b*, T*c*, T*d*, T*finish*
Domain:	Non-negative real numbers
Constraints:	T*a* +2 ≤ T*b*
	T*a* +2 ≤ T*c*
	T*b* +3 ≤ T*d*
	T*c* +5 ≤ T*finish*
	T*d* +4 ≤ T*finish*

and the minimum time taken could be:

T*a* = 0
T*b* = 2
2 ≤ T*c* ≤ 4
T*d* = *5*
T*finish* = *9*

Arc consistency
We need to have an algorithm for ensuring that the domain of each variable is consistent with the constraints. This can be diagrammed as a series of bidirectional arcs between nodes which represent individual variables.



The principle is that a constraint can have the effect of changing the domain of a variable. Assuming each variable starts with the domain :

[0,1,2,3,4,5,6,7,8,9,10]

we move round the graph, adjusting the values of the variables. For example, if T*a* is [0,1,2,3,4,5,6,7,8,9,10] and we have the constraint T*a* +2 ≤ T*b*, then T*b* must be adjusted to be consistent with T*a:* [2,3,4,5,6,7,8,9,10]. In turn, the domain of T*a* must be adjusted as well: [0,1,2,3,4,5,6,7,8].

As T*a* is now [0,1,2,3,4,5,6,7,8] then T*c* must be adjusted to be consistent with *Ta* because of the presence of the constraint T*a* +2 ≤ T*c* becoming [2,3,4,5,6,7,8,9,10]. This change propagates through the network so, with the constraint T*c* +5 ≤ T*finish*, T*finish* is [7,8,9,10].

The process of arc consistency can be applied until there are no more changes to be made in the domains of variables, but that is not the end of the search for a solution. Arc consistency doesn't ensure that a value picked from the domain of one variable and a value picked from the domain of another variable are part of the same solution; it only ensures that none of the constraint relationships are broken. To find a solution, it is necessary to pick a value from one variable's domain and then apply the constraints to pick a variable from all other domains. (This can be done using a "backtracking" search such as Prolog's.) There are three possible outcomes: no solution (no set of values satisfies all constraints); one solution (one set of values satisfies all constraints); more than one solution (where one or more of the variables can have more than one valuation). Sometimes the programmer will want to choose between multiples solutions by trying to apply some evaluation function to give the optimal solution, e.g. the minimum time duration.

Example of domain revision

Arc	**T*a***	**T*b***	**T*c***	**T*d***	**T*finish***
	0..10	0..10	0..10	0..10	0..10
T*b*,T*a*		2..10			
T*d*,T*b*				5..10	
T*f*,T*d*					
T*d*,T*f*					
T*b*,T*d*					
T*a*,T*b*					
T*c*,T*a*					
T*c*,T*f*					

It is important to realise that the programmer has to do nothing other than to specify the relevant constraints. All work of revising domains is done automatically when one variable's domain is changed.

Constraint Logic Programming

As its name suggests, this is a part of logic programming that uses constraints instead of unification. Unification can be seen as a very limited from of constraints satisfaction – one where identity is required rather than, for instance, arithmetic equality and inequality. A CLP system works by replacing unification with constraint satisfaction. Essentially, there is a global store of constraints; each rule that contains constraints produces its own list of current constraints and these and the global constraints are merged. If the domains of variables are made consistent the process continues; otherwise the program fails (i.e. is arc inconsistent).

There are several types of CLP system (written as CLP*(X)* where *X* denotes the type of the domain):

- CLP(R) domain is 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;
- CLP(Q) rational numbers;
- CLP(Z) integers.

CLP(R)
We can enter constraints directly at the prompt:

```
| ?- { 3 * X - 2 * Y = 6, 2 * Y = X }.
X = 3.0,
Y = 1.5 ? ;
no
```

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_pl(Celsius, Fahrenheit) :-
     Celsius is (Fahrenheit - 32) * 5 / 9.
```

and in CLP(R)

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

Prolog is very inflexible in the way its code can be used:

```
| ?- convert_pl(25, Fahr).
! Instantiation error in argument 2 of is/2
! goal:  _85 is(_76-32)*5/9
| ?- convert_pl(Cels, 80).
Cels = 26.666666666666668 ? ;
no
| ?- convert_pl(Cels, Fahr).
! Instantiation error in argument 2 of is/2
! goal:  _85 is(_76-32)*5/9
```

but CLP(R) is very much more flexible:

```
| ?- convert_clpr(25, Fahr).
Fahr = 77.0 ? ;
no
| ?- convert_clpr(Cels, 80).
Cels = 26.666666666666664 ? ;
no
| ?- convert_clpr(Cels, Fahr).
{Fahr=32.0+1.7999999999999998*Cels} ? ;
no
```

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. Can similar ideas be applied to the Sudoku puzzle? That is the subject of the next lecture.

Applications of CLPs

CLP(Q, R) is used for:

- scheduling (e.g. tasks)
- simulating (e.g. electronic circuits)
- verification
- error diagnosis

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.

Extending your learning

Use the temperature conversion program as the basis for writing programs to convert between other units of measurement, from instance between miles and kilometres.

Preparation for the next lecture

Revise your knowledge of the generate-and-test search technique used in the Sudoku puzzle solver (Lecture 10)..

s

Example 1: converting between Celsius and Fahrenheit

```
:- ensure_loaded(library(clpr)).


/* ********************************************** */
/*                                                */
/*   convert_clpr/2                               */
/*      Arg 1: temperature in degrees Celsius     */
/*      Arg 2: temperature in degrees Fahrenheit  */
/*   Summary: Converts between Celsius and        */
/*            Fahrenheit.                         */
/*   Author: P J Hancox                           */
/*   Date: 24 October 2012                        */
/*                                                */
/* ********************************************** */

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







/* ********************************************** */
/*                                                */
/*   convert_clpr/2                               */
/*      Arg 1: temperature in degrees Celsius     */
/*      Arg 2: temperature in degrees Fahrenheit  */
/*   Summary: Converts Fahrenheit into Celsius.   */
/*   Author: P J Hancox                           */
/*   Date: 24 October 2012                        */
/*                                                */
/* ********************************************** */

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