Introduction

Constraint programming is one of the most exciting developments in programming languages of the last decade. Based on a strong theoretical foundation, it is attracting widespread commercial interest and is now becoming the method of choice for modelling many types of optimization problems, in particular, those involving heterogeneous constraints and combinatorial search. Not surprisingly, therefore, constraint programming has recently been identified by the ACM (Association for Computing Machinery) as one of the strategic directions in computing research.

The reason for this interest in constraint programming is simple. Early programming languages, such as FORTRAN-66, closely reflected the underlying physical architecture of the computer. Since then, the major direction of programming language design has been to give the programmer freedom to define objects and procedures which correspond to entities and operations in the application domain. Object oriented languages, in particular, provide good mechanisms for declaring program components which capture the behaviour of entities in a particular problem domain. However, traditional programming languages, including object oriented languages, provide little support for specifying *relationships* or *constraints* among programmer-defined entities. It is the role of the programmer to explicitly maintain these relationships, and to find objects which satisfy them.

For instance, consider Ohm's Law,

$$V = I \times R,$$

which describes the relationship between voltage V, current I and resistance R in a resistor. In a traditional programming language, the programmer cannot use this relationship directly, instead it must be encoded into statements which compute a value for one of the variables given values for the other two variables. Thus, I can be computed from V and R using the assignment

```
I := V / R.
```

But if the value of R must be determined from the other two, a different assignment statement is needed:

```
R := V / I.
```

Requiring the programmer to explicitly maintain relationships between objects in the program is reasonable for applications in which the relationships are simple and static. However, for many applications, the crux of the problem is to model the relationships and find objects that satisfy them. For this reason, since the late 1960's, there has been interest in programming languages which allow the programmer simply to state relationships between objects. It is the role of the underlying implementation to ensure that these relationships or "constraints" are maintained. Such languages are called *constraint programming* languages.

The first pioneering constraint programming languages were only partly successful. They involved augmenting a traditional language with constraints solved by simple ad hoc techniques. These languages primarily depended on "local" propagation for constraint solving. Local propagation works by using a constraint to assign a value to an unknown variable given known values for the other variables in the constraint. For instance, Ohm's Law can be used to compute a value for R, I or V, given the other two variables have known values. The problem with local propagation is that it is a relatively weak constraint solving method. For example, it cannot be used to solve simultaneous equations such as $X = Y - Z$ and $X = 2 \times Y + Z$. Owing partly to a reliance on local propagation these early languages had two major weaknesses: the constraint solving facilities were not powerful enough and the languages were not sufficiently expressive. This meant that the languages were not general-purpose, but rather primarily application oriented.

In the last decade there has been renewed interest in constraint programming languages. More recent constraint programming languages have overcome the difficulties of the early languages. They provide constraints which are thoroughly integrated into the programming language, allowing the programmer to state problems at a very high level, while their underlying execution mechanism uses powerful incremental constraint solvers. The new generation of constraint programming languages are true programming languages which are suited for a wide variety of applications.

As a simple example of an application using a modern constraint programming language, imagine that you are buying a house and want to investigate various options for repaying your loan. At each repayment interval, the amount of interest that is accrued is $P \times I$ where P is the amount borrowed (or principal) and I is the interest rate. The accrued interest is added to the current principle to obtain a new principal NP. If any repayment R is made this is subtracted. This behaviour is encapsulated in the equational constraint

$$NP = P + P \times I - R.$$

A multiple period loan (mortgage) that lasts for T time periods can be described by repeatedly applying this calculation, once for every time period, until there are no periods remaining. The final amount owing is called the balance B. This behaviour is encapsulated in the following constraint program:

```
mortgage(P, T, I, R, B) :-
    T = 0,
    B = P.
mortgage(P, T, I, R, B) :-
    T ≥ 1,
    NT = T - 1,
    NP = P + P * I - R,
    mortgage(NP, NT, I, R, B).
```

The user-defined constraint `mortgage` defines the relationship between the initial principal P, time of loan T, interest rate I, repayment amount R and final balance B. The first rule (the first 3 lines) handles the case when no time periods remain. In this case the balance is simply the current principal. The second rule (the last 5 lines) handles the case in which the number of periods remaining is greater than or equal to one. In this case a new time (NT) of one less is calculated. Then the change to the principal is calculated. The remainder of the loan is determined by considering it to be a new loan with the new amount owing for one less time period.

An equivalent program may seem easy to write in a traditional language such as C. The advantage of this program is, because all the statements are made up of constraints, it is extremely versatile. Program execution is initiated by giving a goal which the system then rewrites to constraints. For example, consider borrowing \$1000 for 10 years at an interest rate of 10% and repaying \$150 per year. This is expressed by the goal

```
mortgage(1000, 10, 10/100, 150, B).
```

Given this goal, the answer returned by the program is $B = 203.129$, indicating that the balance remaining will be \$203.13.

The same program can be used in many other ways. For example, given the repayment is \$150 and at the end of the loan the remaining balance should be 0, we may wish to ask "how much can be borrowed in a 10 year loan at 10% with repayments of \$150 ?" This question is expressed by the goal

```
mortgage(P, 10, 10/100, 150, 0).
```

The answer given is $P = 921.685$. A more complex query is to ask for the relationship between the initial principal, repayment and balance in a 10 year loan at 10%. This is done with the goal:

```
mortgage(P, 10, 10/100, R, B).
```

The answer is the constraint, $P = 0.3855 * B + 6.1446 * R$, relating the variables P, R and B.

In all of these cases the *same* program has been used to solve the given problem. This contrasts with a conventional programming language in which a different program would be required to answer each of these queries. Indeed, writing a program to answer the last query would be quite difficult. Constraint programming has allowed us to specify the problem very naturally, and at a high level, in terms of arithmetic constraints. It is the job of the underlying language implementation to solve these constraints, rather than the job of the programmer.

This example program illustrates how constraint programming can be naturally used to model complex applications. For this reason constraint programming has been used in many diverse areas. The engineering disciplines are one area in which constraint programming has been applied with notable success. Many engineering problems are extremely well suited to constraint programming since they often involve hierarchical composition of complex systems, mathematical or Boolean models, and—especially in the case of diagnosis and design—deep rule-based reasoning. Another major area has been options trading and financial planning, where applications usually take the form of expert systems involving mathematical models. More exotic applications have included restriction site mapping in genetics and generation of test data for communications protocols.

Arguably, however, the most important application of the new constraint programming languages is for tackling difficult multi-faceted combinatorial problems, such as those encountered in job scheduling, timetabling and routing. These kinds of problems are very difficult to express and solve in conventional programming languages. This is because they require searching through a possibly exponentially sized solution space in order to find a good or optimal solution to the problem. To obtain reasonable efficiency, the constraints must be used to prune the search space.

There are two traditional approaches to solving difficult combinatorial problems: either hand-craft an algorithm which will solve the exact problem in a traditional programming language, or else model the problem using an off-the-shelf solver for a particular class of arithmetic constraints. The drawback of a hand-crafted solver is that it is expensive and it is not easy to modify when the problem changes. The drawback of an off-the-shelf solver is that it is unlikely to be sufficiently expressive for constraints in the application domain to be expressed naturally and flexible enough to allow domain-specific heuristics to be used in the solver. Modern constraint programming languages overcome these problems by providing a programming language built on top of a sophisticated constraint solver. This means that a programmer can code domain-specific methods in the programming language, while generic solving techniques are provided by the language implementation.

As a simple example, consider the crypto-arithmetic problem:

$$\begin{array}{cccccc}
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}$$

where each letter represents a different digit and the arithmetic equation holds. This is modelled by the following constraint program:

```
smm(S,E,N,D,M,O,R,Y) :-
    [S,E,N,D,M,O,R,Y] :: [0..9],
    constrain([S,E,N,D,M,O,R,Y]),
    labeling([S,E,N,D,M,O,R,Y]).

constrain([S,E,N,D,M,O,R,Y]) :-
    S ≠ 0, M ≠ 0,
    alldifferent_neq([S,E,N,D,M,O,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.
```

In the program, each of the variables S, E, N, D, M, O, R and Y is declared to range over the values [0..9] while the other constraints in the problem are defined by the user-defined constraint `constrain`. The underlying solver for such integer constraints is very fast, employing sophisticated propagation techniques. The price of this speed is that the solver is incomplete, and so may return an answer *unknown* indicating that it does not know if there is a solution or not. In this example, the programmer has called an auxiliary function, `labeling`, which extends the underlying solver by using search to try different values in the range [0..9] for each variable. This means that the program is guaranteed to find a solution when it exists. In this case, `labeling` is a library function provided by the system, but a strength of the CLP approach is that the programmer can define their own problem-specific solver extension. Given the goal `smm(S,E,N,D,M,O,R,Y)`, the program will quickly determine that $S = 9$, $E = 5$, $N = 6$, $D = 7$, $M = 1$, $O = 0$, $R = 8$ and $Y = 2$.

The first modern constraint programming languages were extensions of logic programming languages. These languages were called ***constraint logic programming languages*** and extended logic programming languages such as Prolog by replacing unification with a test for constraint satisfaction using a dedicated solver. They arose from research in Europe and Australia in the late 1980's. Both programs given above are examples of constraint logic programs. Different solvers and different application domains led to different languages which share a common evaluation mechanism.

Inspired by the success of constraint logic programming, several other classes of constraint programming languages have recently been suggested. These include: concurrent constraint languages which use constraint entailment to extend constraint logic programming languages by providing asynchronous communication between "agents," constraint query languages for databases which extend relational databases by allowing tuples that contain "constrained variables," constraint functional programming languages, constraint imperative programming languages, and object oriented constraint solving toolkits.

Constraint logic programming languages are, however, the archetypal constraint programming languages. They are, in a sense, the minimal and purest constraint

Benson and others, is the first constraint imperative programming language [49]. There are now several object oriented constraint solving toolkits. One of the first was an object oriented Lisp based toolkit called PECOS [105] which was the precursor to the commercially available C++ based ILOG SOLVER [106]. Further references for these different paradigms can be found at the end of Chapters 11 and 12.

Constraint programming is now a fertile research area, with two main international conferences: *Principles and Practice of Constraint Programming (CP)*, proceedings of which are published by Springer-Verlag in the Lecture Notes in Computer Science Series; and *Practical Applications of Constraint Technology (PACT)*, proceedings of which are published by The Practical Application Company. There are also numerous international workshops devoted to specific applications and classes of languages, together with sessions at the major international conferences on programming languages. One journal, *Constraints*, published by Kluwer Academic is exclusively devoted to constraints and constraint programming. Many other journals, in particular *The Journal of Logic Programming* and *Artificial Intelligence*, also publish papers on constraint programming. Information about constraint programming is available on the internet starting from the Constraints Archive at `http://www.cirl.uoregon.edu/constraints/` and through the news group `comp.constraints`.