Constraint Processing Theory

Supplementary document – Read this document and join the class on May 30

Constraints are used to model the behavior of systems of objects in the real world by capturing an idealized view of the interaction among the objects. Here we will study some fundamentals for further handling of constraints.

Constraints and valuations

The statement X=Y+2 is an example of a *constraint* (arithmetic constraint; there are many types of constraints). X and Y are *variables*, that is they are place-holders for values, presumably in this case numbers. It stipulates a relation which must hold between any values with which we choose to replace X and Y, namely, the first must be two greater than the second. The statement X=Y+2 is not simply true or false. Its status depends on the values we substitute for X and Y. However, when we have a statement involving variables, one question we can meaningfully ask is whether *any* substitution will yield truth. In the case of X=Y+2 the answer is clearly *yes* (take X=3 and Y=1, for example), but in other cases the answer will be *no*, witness X=X+2. Occasionally the answer may even be *don't know*!

☛ To judge whether given constraint can be satisfied or not is called *constraint satisfaction problem(CSP)*. A program or a system which can cope with CSP is called a *constraint solver*, from which we can get the output *true*, *false* or *unknown* corresponding to yes, no or "don't know" in the above explanation, respectively.

Throughout our "Constraint Processing Theory" classes we shall use strings which start with an upper case letter or underscore "_" to name variables. For example: X, Y, Yoshie, List, _somename are all variables.

The legitimate forms of constraint and their meaning is specified by a *constraint domain*. Constraints are written in a language made up of constants like 0 and 1, functions like + and /, and constraint relations like equality (=) and less than (<).

(1) The constraint domain specifies the "syntax" of the constraints. That is, it specifies the rules for creating constraints in the domain. It details the allowed constants, functions and constraint relations as well as how many arguments each function and constraint relation is required to have and how the arguments are placed. For instance, both + and < require two arguments and are written between their arguments.

(2) The constrain domain also determines the values that variables can take. For example, the constraint X*X<1/2 (* denotes multiplication) means quite different things depending on whether

X can take integer values, real values or complex values.

(3) Finally, the constraint domain determines the meaning of all of these symbols. It determines what will be the result of applying a function to its arguments and whether a constraint relation holds for given arguments. For example, for arithmetic the application of the + function in the expression 3+4 evaluates to 7, and the constraint 1<2 holds while it is not the case that 2<1 holds.

Given a constraint domain $\mathcal{D}$ the simplest form of constraint we can define is a *primitive constraint*. A primitive constraint consists of a constraint relation symbol from $\mathcal{D}$ together with the appropriate number of arguments. These are constructed from the constants and functions of $\mathcal{D}$ and variables. Most constraint relations in Constraint Processing Theory classes are binary, that is they require two arguments (*binary constraints*).

For example, in the constraint domain of real numbers, which we call $\mathcal{Real}$, variables take real number values, the set of function symbols is +, -, * and /, while the constraint relation symbols are =, <, $\leqq$, >, $\geqq$, all of which take two arguments. X>3 and X+Y*Y=5 are examples of primitive constraints.

More complicated constraints can be built from primitive constraints by using the conjunctive connective $\wedge$ which stands for "and". For example, a constraint X+Y=1 $\wedge$ X-Y=1 is composed of two primitive constraints joined by conjunction.

Definition 1

A *constraint* C is of the form $C=c_1 \wedge c_2 \wedge \ldots \wedge c_n$, where $n \geqq 0$ and $c_1, c_2, \ldots, c_n$ are primitive constraints. The symbol $\wedge$ denotes *and*, so a constraint C holds whenever all of the primitive constraints $c_1, c_2, \ldots, c_n$ hold.

Given a constraint, we can determine values of variables for which the constraint holds. By replacing variables by their values, the constraint can be simplified to a variable-free constraint which may then be seen to be either true or false. For example, consider the constraint

X+Y=1 $\wedge$ X-Y=1.

The assignment of values to variables X and Y

$\{X \mapsto 1, Y \mapsto 0\}$

makes the constraint take the form

1+0=1 $\wedge$ 1-0=1

which after some simplification can be seen to be true. On the other hand, the assignment

$\{X \mapsto 0, Y \mapsto 1\}$

leaves the constraint in the form

0+1=1 $\wedge$ 0-1=1

which is false.

Definition 2

A *valuation* θ for a set V of variables is an assignment of values from the constraint domain to the variables in V. Suppose $V=\{X_1, \ldots, X_n\}$ then θ may be written $\{X_1 \mapsto d_1 \ldots, X_n \mapsto d_n\}$ indicating that each X_i is assigned the value d_i.

An expression e over variables V is given a value θ (e) under the valuation θ over variables V. $\theta(e)$ is obtained by replacing each variable by its corresponding value and calculating the value of the resulting variable-free expression. Let *vars*(e) denote the set of variables occurring in an expression e. Similarly, let *vars*(C) denote the set of variables occurring in a constraint C. If θ is a valuation for $V \supset vars(C)$, then it is a *solution* of C if $\theta(C)$ holds in the constraint domain.

☛ Because you are not familiar with this kind of description, understanding each terminology and idea might be rather difficult for you. But you'll get accustomed to them soon. Take it easy!

Definition 3

A constraint C is *satisfiable* if it has a solution. Otherwise it is *unsatisfiable*.

Example

Suppose we have a variable set $V=\{X, Y, Z\}$, but the given constraint C takes the form $C=(X+Y=1) \wedge (X-Y=1)$, which means $vars(C)=\{X,Y\}$ and $V \supset vars(C)$. We consider a valuation over V : $\{X \mapsto 1, Y \mapsto 0, Z \mapsto 1\}$ which is assumed to be allowed in the constraint domain, then it makes C a variable-free constraint and true. That is, the constraint C has a solution and satisfiable.

Definition 4

Two constraints C_1 and C_2 are *equivalent*, written $C_1 \leftrightarrow C_2$, if they have the same set of solutions.

The reason we consider constraints to be sequences of primitive constraints is because the order of the primitive constraints will prove to be important for some of the constraint manipulation algorithms we shall define later in Constraint Processing Theory classes. Many algorithms, however, produce the same result, independent of the order and the number of occurrences of each primitive constraint. That is, in these algorithms constraints are treated as sets of primitive constraints. We shall find it useful to have a function which takes a constraint and returns the set of primitive constraints in the constraint:

Definition 5

The function *primitives* takes a constraint $C = c_1 \wedge c_2 \wedge \ldots \wedge c_n$ and returns the set of primitive constraints $\{ c_1, c_2, \ldots, c_n \}$. That is to say,

$primitives(C) = \{ c_1, c_2, \ldots, c_n \}$.

Clearly, if $primitives(C_1)=primitives(C_2)$ then $C_1 \leftrightarrow C_2$. However, please note that not all equivalent constraints contain the same set of primitive constraints. For instance, $X>0 \wedge Y=X+2$ is equivalent to $X=Y-2 \wedge Y>2$.