A formal model for Constraint Programming

Marco Kuhlmann

3rd May 2007

The purpose of this document is to outline a formal model for constraint programming. It draws heavily from course slides made available by Christian Schulte.

Preliminaries We write $\mathbb{N}$ for the set of (positive) natural numbers. Given $n \in \mathbb{N}$, we write $[n]$ for the set of all positive natural numbers up to and including n.

1 Constraint satisfaction problems

We start with a formal definition of constraint satisfaction problems. Remember that informally, we introduced it as being specified by three components: a set of variables, for each variable a sets of possible values for that variable, and a set of constraints.

Definition 1 Let $n \in \mathbb{N}$. A *constraint satisfaction problem* with n variables is a pair $P = (D, C)$, where D is a finite set of *values*, called the *domain* of P, and C is a finite collection of n-ary relations over D, called the *constraints* of P. □

Given a constraint satisfaction problem $P = (D, C)$, we write $\mathrm{dom}(P)$ to refer to D, $\mathrm{con}(P)$ to refer to C, and $\mathrm{var}(P)$ to refer to the set $\{ x_i \mid i \in [n] \}$ of *problem variables*, where n is the arity of the relations in C. Note that problem variables are implicit in our formalization of constraint satisfaction problems; this is to keep things simple.

Definition 2 Let P be a constraint satisfaction problem. An *assignment* for P is a function $\alpha : \mathrm{var}(P) \to \mathrm{dom}(P)$. An assignment α is called a *solution*, if $\alpha \in \bigcap_{C \in \mathrm{con}(P)} C$. □

Note that assignments and solutions for P can also be seen as $|\mathrm{var}(P)|$-tuples over $\mathrm{dom}(P)$. We freely switch between the two views. We write $\mathrm{ass}(P)$ for the set of all assignments and $\mathrm{sol}(P)$ for the set of all solutions of a constraint satisfaction problem P.

Example 1 Consider the problem $P_1 = ([3], \{C_1, C_2\})$, where

$$C_1 = \{(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)\} \quad \text{and}$$
$$C_2 = \{(1,1),(2,1),(3,1),(2,2),(3,2),(3,3)\}.$$

This problem has two problem variables, x_1 and x_2. We can understand C_1 as the constraint $x_1 \leq x_2$, and C_2 as the constraint $x_2 \leq x_1$. There are nine possible assignments for P_1; each variable can take values from $[3]$. There are three solutions: $(1,1)$, $(2,2)$, $(3,3)$. We can understand the set of solutions as the equality constraint $x_1 = x_2$. □

Constraint satisfaction problems provide a formal model for constraint programming, but they only say *what* the problem and its solutions are, not *how* to compute them. In the remainder of this document, we develop a more implementation-orientated model for constraint programming.

2 Constraint stores

The first ingredient in our formal model is the *constraint store*. Informally, a constraint store represents our 'current knowledge' about a constraint satisfaction problem. Formally, a store maps each problem variable to a set of possible values for that variable. In contrast to a constraints satisfaction problem, this mapping can be efficiently implemented—for example, by using lists of integer intervals for integer variables, and lower bound and upper bound representations for set variables.

Definition 3 Let P be a constraint satisfaction problem. A *constraint store* for P is a function $s : \mathrm{var}(P) \to \wp(\mathrm{dom}(P))$. □

We write $S(P)$ for the set of all stores for a constraint satisfaction problem P. A constraint store can be understood as a compact description of a set of assignments.

Definition 4 Let s be a constraint store for P. The set of *assignments* licensed by s is defined as $\mathrm{ass}(s) := \{\, \alpha \in \mathrm{ass}(P) \mid \forall x \in \mathrm{var}(P).\ \alpha(x) \in s(x) \,\}$. □

By their ability to represent sets of assignments, constraints stores can approximate solutions of constraint satisfaction problems. But not every set of solutions can be represented as the set of assignments licensed by a constraint store:

Example 2 Consider the constraint satisfaction problem $P_2 = ([2]; \{(1,2),(2,1)\})$. The (single) constraint of this problem cannot be represented as the set of assignments of any store. In fact, every store s for which $\mathrm{sol}(P) \subseteq \mathrm{ass}(s)$ should hold needs to represent the set of all assignments for P, $\mathrm{ass}(P) = \{(1,1),(1,2),(2,1),(2,2)\}$. □

Note that every single assignment can be represented as the set of assignments licensed by a constraint store. This in particular means that a constraint store can represent a single solution of a constraint satisfaction problem.

Strengths of stores The cardinality of the set of assignments licensed by a store is a measure for how much we know about the constraint satisfaction problem associated to it: the fewer the number of licensed assignments, the fewer the uncertainty about the possible values for the problem variables. To model this formally, we impose a partial order on the set of all stores:

Definition 5 Let P be a constraint satisfaction problem. Given two constraint stores s_1 and s_2 for P, we say that s_1 is *stronger* than s_2, and write $s_1 \leq s_2$, if $s_1(x_i) \subseteq s_2(x_i)$ holds for all variables $x_i \in \mathrm{var}(P)$. □

The weakest store in this order is the store that maps every variable to the full domain; in this case, $\mathrm{ass}(s) = \mathrm{ass}(P)$. The basic idea behind our formal model of constraint programming is to develop a way in which the weakest store can be successively strengthened to either a store that licenses a single solution of the original problem, or to a store that does not license any solutions at all (a *failure*).

3 Propagators

The task of a propagator is to strengthen constraint stores by excluding impossible values from the domains of one or more variables. Informally, propagators implement constraints. Formally, a propagator for a constraint satisfaction problem P is a function $p : S(P) \to S(P)$ with two additional properties. These properties mirror two intuitions about inferential processes.

Contracting A constraint store reflects our 'current knowledge' about a constraint satisfaction problem. The first property of propagators says that applying a propagator (an inference) should at most increase this knowledge.

Definition 6 Let P be a constraint satisfaction problem. A function $p : S(P) \to S(P)$ is called *contracting*, if $p(s) \leq s$ holds for all stores $s \in S(P)$. □

Monotonicity The second property of propagators says that once ensured information should always be accessible in the remainder of the inferential process—if we have derived $A \wedge B$, we should still be able to use implications $A \Rightarrow C$ and $B \Rightarrow C$.

Example 3 Consider the stores $s_1 = \{x \mapsto \{1,2\}\}$ and $s_2 = \{x \mapsto \{1\}\}$ and the propagator $p(s) = \textbf{if } s(x) = \{1,2\} \textbf{ then } x \mapsto \emptyset \textbf{ else } s$. Then $s_1 > s_2$, but $p(s_1) < p(s_2)$. Arguably, this is counter-intuitive. It is also problematic on a practical level, as it makes the result of propagation dependent on the order in which propagators are executed. □

Definition 7 Let P be a constraint satisfaction problem. A function $p : S(P) \to S(P)$ is called *monotonic*, if $s_1 \leq s_2$ implies that $p(s_1) \leq p(s_2)$, for all stores $s_1, s_2 \in S(P)$. □

Definition 8 Let P be a constraint satisfaction problem. A *propagator* for P is a function $p : S(P) \to S(P)$ that is both contracting and monotonic. □