Binary CSP Solving as an Inference Process

Carlos Castro
INRIA-Lorraine & CRIN
615 rue du jardin botanique, BP 101,
54602 Villers-lès-Nancy Cedex, France
e-mail: Carlos.Castro@loria.fr

Abstract

We describe constraint solving using a rule-based approach. The distinction made between deduction rules and strategies by computational systems allows us to improve our understanding of the existing algorithms for solving Binary CSP once they are expressed as rewriting rules coordinated by strategies.

Keywords: CSP, Computational Systems, Rewriting Logic.

1. Introduction

In this work we formalise solving of Constraint Satisfaction Problems (CSP) [3] as an inference process. We associate actions with rewriting rules and control with strategies that establish the order of applications of the inferences. This general approach was firstly proposed by Kirchner, Kirchner and Vittek in [2], where they point out some advantages of this schema over systems where solvers are encapsulated in black boxes. A constraint solver is viewed as a computational system aimed at computing solved forms for a class of considered formulas called constraints. Computation is exactly application of rewriting rules on a term and strategies describe the intended set of computations.

An elementary constraint is an atomic formula built on a signature \( \Sigma = (F, P) \), where \( F \) is a set of ranked function symbols and \( P \) a set of ranked predicate symbols, and a denumerable set \( X \) of variable symbols. Elementary constraints are combined with usual first-order connectives. We denote the set of constraints built from \( \Sigma \) and \( X \) by \( C(\Sigma, X) \). Given a structure \( D = (D, I) \), where \( I \) is an interpretation function and \( D \) the domain of this interpretation, a \( (\Sigma, X, D) \)-CSP is any set \( C = \{ c_1 \wedge \ldots \wedge c_n \} \) such that \( c_i \in C(\Sigma, X) \) \( \forall i = 1, \ldots, n \). A solution of \( C \) is a mapping from \( X \) to \( D \) that associates to each variable \( x \in X \) an element in \( D \) such that \( c_i \) is satisfiable in \( D \). A solution of \( C \) is a mapping such that all constraints \( c_i \in C \) are satisfiable in \( D \). Given a variable \( x \in X \) and a non-empty set \( D_x \subseteq D \), the membership constraint of \( x \) is a relation given by \( x \in^D D_x \). We use these membership constraints to make explicit the domain reduction process during the constraint solving. In practice, the sets \( D_x \) have to be set up to \( D \) at the beginning of the constraint solving, and constraint propagation will eventually reduce them. We only consider unary and binary predicate symbols and a finite domain of interpretation. This class of CSP is known as Binary CSP.

We are interested in description of constraint solving using rule-based algorithms because of the explicit distinction made in this approach between deduction rules and control. Our goal is to improve our understanding of the algorithms developed for solving Binary CSP once they are expressed as rewriting rules coordinated by strategies. To verify our approach we have implemented a prototype in the system ELAN, an interpreter of computational systems\(^1\). We hope that this work leads the way to the design of a formalism allowing to apply the knowledge already developed in the domain of Automated Deduction.

2. A Computational System for Solving CSP

A basic form for a CSP \( P \) is a conjunction of formulas of the form

\[
\bigwedge_{i \in I} (x_i \in^{D_{x_i}} D_{x_i}) \wedge \bigwedge_{j \in J} (x_j \in^{V_j} V_j) \wedge C
\]

equivalent to \( P \) and such that for each variable we have associated a membership constraint or an equality constraint, variables appearing in equality constraints must not appear in membership constraints, and the set associated to each variable in the membership constraints must not be empty. A basic assignment is obtained by assigning each variable in the equality constraints to the associated value \( v \) and each variable \( x \) in the membership constraints to any

\(^1\)ELAN is available via anonymous ftp at ftp.loria.fr in the directory /pub/loria/protheo/softwares/Elan
value in the set $D_x$. A solved form for a CSP $P$ is a conjunction of formulas in basic form and such that all basic assignments satisfy all constraints.

### 2.1. Rewriting Rules

Figure 1 presents **ConstraintSolving**, a set of rewriting rules for constraint solving in CSP. We use a constant $\mathbf{F}$ to denote an unsatisfiable CSP. The rule **Node-Consistency**, where $RD(x \in D_x, c^s(x))$ stands for the set $D'_x = \{ v \in D_x | \ c^s(v) \}$, verifies node consistency and it is based on NC-1 algorithm. The rule **Arc-Consistency**, where $RD(x_1 \in D_{x_1}, x_2 \in D_{x_2}, c^s(x_1, x_2))$ stands for the set $D'_{x_1} = \{ v \in D_{x_1} | \ (\exists w \in D_{x_2}) \ c^s(v, w) \}$, verifies arc consistency and it is based on AC-3 algorithm. Rule **Instantiation** corresponds to a variable assignment. **Elimination** expresses the fact that once a variable has been instantiated, we can propagate its value through all constraints where the variable is involved in. **Falsity** expresses the obvious fact of unsatisfiability. The rule **Generate** expresses the simple fact of branching to carry out exhaustive search.

### 2.2. Strategies

The expressive power of computational systems allows to express different heuristics through the notion of strategy. In this way, for example, node consistency can be obtained by applying [(Node-Consistency | Falsity)*, and arc consistency can be obtained by applying [(Arc-Consistency | Falsity)* 2. We can integrate constraint propagation and searching in order to get a solved form, for example, following the heuristic Forward Checking. Let us define the following strategies:

- **NodeC**:
  
  **Node-Consistency**[(Instantiation Elimination)]Falsity)*

- **ArcC**:
  
  **Arc-Consistency**[(Instantiation Elimination)]Falsity)*

Using these substrategies we can implement a new one:

- **Forward Checking**:
  
  [NodeC|ArcC]* Generate Elimination)*

### 2.3. Implementation

We have implemented a prototype in the system ELAN. Arc consistency is achieved using the heuristics of AC-1 or AC-3. The non determinism of ELAN allowed an easy implementation of Forward Checking. The benchmarks carried out encourage us to continue in this way.

### 3. Conclusion

We have verified how computational systems are an easy and natural way to describe and manipulate Binary CSP. The main contribution of this work is the formalisation of algorithms to solve Binary CSP in a way that makes explicit difference between actions and control that until now were embeded in black box like algorithms. We hope that powerful strategy languages will allow us to evaluate existing hybrid techniques for constraint solving and design new ones. More details about this work can be obtained in [1].

### References

