A Category-Based Equational Logic Semantics to Constraint Programming *

Răzvan Diaconescu

Institute of Mathematics of the Romanian Academy

Abstract. This paper exploits the point of view of constraint programming as computation in a logical system, namely *constraint logic*. We define the basic ingredients of constraint logic, such as *constraint models* and *generalised polynomials*. We show that constraint logic is an institution, and we internalise the study of constraint logic to the framework of category-based equational logic. By showing that constraint logic is a special case of category-based equational logic, we integrate the constraint logic programming paradigm into equational logic programming. Results include a Herbrand theorem for constraint logic programming characterising Herbrand models as initial models in constraint logic.

1 Introduction

1.1 Extensible Constraint Logic Programming

Constraint logic programming has been recently emerging as a powerful programming paradigm and it has attracted much research interest over the past decade. Constraint logic programming merges two declarative programming paradigms: constraint solving and logic programming. Mathematical Programming, Symbolic Computation, Artificial Intelligence, Program Verification and Computational Geometry are examples of application areas for constraint solving. Constraint solving techniques have been incorporated in many programming systems; CLP [20], PrologIII [5], and Mathematica are the best known examples. The computational domains include linear arithmetic, boolean algebra, lists, finite sets. Conventional logic programming (i.e., Prolog) can be regarded as constraint solving over term models (i.e., Herbrand universes). In this way, constraint logic programming can be regarded as a generalisation of logic programming that replaces unification with constraint solving over computational domains. In general, the actual constraint logic programming systems allow constraint solving for a fixed collection of data types or computational domains.[2] Constraint logic programming allowing constraints over *any* data type will be called **extensible** (abbreviatted **ECLP**).

* This research was partially supported by a grant for basic research in information science and technology from the Romanian Academy of Sciences.

[2] A computational domain can be regarded as a model (not necessarily the standard one) for a certain data type specification.

This paper presents a model theoretic semantics for constraint logic programming, without directly addressing the computational aspect.[3] Our approach departs from the usual ones by following [19] in proving a Herbrand theorem for *constraint logic*, which is the logic underlying ECLP. As with the CLP approach of Jaffar and Lassez [20], both constraint relations and programs are (sets of) sentences in the same logical system. But our constraint logics are much more general than Horn clause logic. Also, the computational domain plays a primary rôle in our definition of constraint logic, rather than being axiomatised in Horn clause logic, as in [20].

When regarded as a model in constraint logic, the computational domain is an *initial* model. This is mathematically linked to the semantics of OBJ-like module systems, the fundamental idea being to regard the models of ECLP as expansions of an appropriate *built-in model* A along a signature inclusion $\iota\colon \Sigma \hookrightarrow \Sigma'$, where Σ is the signature of built-in sorts, operations and relations, and Σ' adds new "logical" symbols. In practice, the constraint relations (i.e., the logical relations one wishes to impose on potential solutions) are limited to atomic sentences involving both Σ-symbols and elements of the built-in model A. However, at the theory level there is no reason to restrict constraint relations to be atomic formulae. The models for ECLP are expansions of the built-in model to the larger signature Σ', and morphisms of constraint models must preserve the built-ins. Thus the constraint models form a comma category, $(A{\downarrow}\mathrm{MOD}(\iota))$.

Example 1. Consider the example of a specification of the Euclidean plane as a vector space over the real numbers.

```
obj R2 is
  pr FLOAT * (sort Float to Real) .
  sort Vect .

  op 0 : -> Vect .
  op <_,_> : Real Real -> Vect .
  op _+_ : Vect Vect -> Vect .
  op -_ : Vect -> Vect .
  op _*_ : Real Vect -> Vect .

  vars a b a' b' k : Real .
  eq 0 = < 0 , 0 > .
  eq < a , b > + < a' , b' > = < a + a' , b + b' > .
  eq k * < a , b > = < k * a , k * b > .
  eq - < a , b > = < - a , - b > .
endo
```

The signature Σ of built-in sorts, operation and relation symbols contains one sort `Real`[4] for the real numbers together with the usual ring operation symbols

[3] However computational aspects are briefly discussed in Section 7.

[4] Obtained here by renaming the sort `Float` of the imported built-in OBJ module `FLOAT` implementing the real numbers as floating point reals.