The category of constraint systems is cartesian-closed

Vijay Saraswat
Xerox PARC
3333 Coyote Hill Road
Palo Alto, Ca 94304
E-Mail: saraswat@parc.xerox.com
Phone: +1.415.812.4747, Fax:+1.415.812.4334

December 1991

Abstract

The development of constraint programming has brought to the forefront the notion of *constraint systems* as certain first-order systems of partial information [JL87,Sar89,SRP91]. Connections between such systems and domain theory have seemed apparent [Sar89] for some time, but have not yet been explored in depth. The need for such a study becomes unavoidable, however, when one considers the development of *higher-order* constraint programming languages. To give semantics to such languages, it becomes necessary to find mathematical structures in which recursive "domain equations" involving constraint systems can be solved.

In this paper we give a general definition of constraint systems utilizing Gentzen-style sequents. Constraint systems can be regarded as enriching the propositional Scott information systems with minimal first-order structure: the notion of variables, existential quantification and substitution. Reflecting the computational reality that an agent "has access" to only finitely many free variables, we take as morphisms approximable maps that are *generic* in all but finitely many variables. We show that the resulting structure forms a category (called **ConstSys**). Furthermore, the structure of Scott information systems lifts smoothly to the first-order setting — we show that the category is cartesian-closed, and other usual functors over Scott information systems (lifting, sums, Smyth power-domain) are also definable and recursive domain equations involving these functors can be solved. This makes it possible to use this category to define the semantics of higher-order, concurrent constraint programming languages.

1 Introduction

The development of domain theory has been driven to a large extent by the desire to find natural models for the λ-calculus. For this purpose, Scott used ideas from logic and recursion-theory to show that it was possible to construct spaces from propositional systems of partial information that contained an isomorphic copy of their own function space. In particular, the category of *information systems* of [Sco82,LW84] is a simple example of such a class of mathematical spaces.

The rise of (concurrent) constraint programming [JL87,Sar89] imposes some new demands on the theory of programming languages. Such languages take to heart the notion of computation via monotonic accumulation of information. The state of the computation does not necessarily specify fully-formed values for the variables of interest. Rather, a store may itself be a constraint, specifying just that certain pieces of partial information (e.g., $X \geq Y + Z$) hold of the variables of interest. On this novel view of *store-as-constraint* it turns out to be possible to build a smooth theory of concurrent computation in which parallel composition turns out to be exactly the computational analogue of conjunction and local variables (hiding) turns out be exactly the computational analogue of existential quantification [Sar89,SRP91,PSSS91]. Thus this computational framework is fundamentally about *logic* programming, i.e. programming *with* logic, i.e. programming with a store that *is* a logical formula. It should not be surprising, therefore, to find that some (rather modest!) extra logical machinery needs to be imported into the foundations of the theory used to describe such computations.

In particular, the study of higher-order constraint programming [Sar91] makes the development of a theory of *first-order* information systems unavoidable. In higher-order constraint programming, processes — which may be thought of as transducers of first-order systems of partial information — may themselves be passed around as parameters, constructed on-the-fly and applied to other processes (including themselves). Thus the question arises whether there are mathematical spaces based on first-order systems of partial information which contain a copy of their own function space.

In this paper we give a simple treatment of such spaces, starting from the propositional information systems of [Sco82]. The main difficulty encountered is in properly formulating the notion of a constraint system, and identifying morphisms betwen constraint systems. Our earlier approaches were based on the cylindric and polyadic algebras of Tarski, Halmos and others [SRP91]. However, they led to complex and unwieldy formulations. More recently, we have adopted the Lawverian approach to logic, considering constraint systems to be given by certain kinds of hyperdoctrines (functors from a cartesain base category to the category of lattices satisfying certain categorical conditions) [PSSS91]. However, it now seems to me that an elementary formulation of these ideas is possible, based on a simple proof-theoretic approach using Gentzen-style sequent systems.

Such sequent systems can also be used to present information systems directly — indeed Scott has often emphasized the constructive, logical nature of information systems, and concrete presentations of domain theory in logical form have been worked out by [Abr91]. But the logical structure in propositional domain theory seems too simple to demand the use of sequent systems in the same way that first-order domain theory demands.

Once the notion of constraint system is formulated, the next issue is identifying a proper notion of morphisms. On computational grounds, morphisms should be certain kinds of transducers or functions which are able to transform some kinds of constraints into some other kinds of constraints. The main concern is with ruling out "deviant" morphisms which may have a non-trivial affect on an infinite number of (visible) variables. Towards this end, we identify the notion of *genericity* of functions for variables, and demand that functions be generic in all but a finite number of variables.

With these concepts in hand, it turns out not to be difficult to show that the nice properties of propositional information systems, carry over to the first-order setting of constraint systems. Products, function-spaces and other operators can be defined in the usual way — the only task left is to check that these operators do produce constraint systems, a task aided by the crucial presentation theorem (2.2) proven using proof-theoretic ideas. Standard arguments are used to establish that recursive domain equations using these functors can be solved in this category.

I should mention that because of the nature of constraint-based computation, it is natural to conceive of the inconsistent state of affairs. Therefore, I have not bothered to keep the extra consistency structure of information systems around in the first-order treatment — I expect such extra structure can be dealt with adequately, if needed, along the lines of the present treatment.

Rest of this paper. We generally follow the outline of [LW84], to keep the presentation simple. First, we present the notion of constraint systems, and establish the critical notion of genericity of (sets of) constraints for variables. Next, we introduce morphisms, and establish that constraint systems form a category. We briefly discuss how constraint systems themselves may be partially ordered via the "is-a-subsystem-of" ordering, thus providing the basic structure needed to show that recursive domain equations can be solved in this category. Finally, we show that the category is closed under products and exponentiation.

2 Constraint systems

Basic definition

Assume the set of variables *Var* is given beforehand and is infinite. Let D be any set of atomic formulas. By $D^{\wedge,\exists}$ we shall mean the smallest set containing

D and closed under conjunction and existential quantification. By $Fin(D)$ we shall mean the set of all finite subsets of D. We shall assume defined as usual the notions of free and bound variables, and of the substitution of a term for a variable. As conventional, we shall identify formulas different from each other only in that bound variables are renamed consistently. For any syntactic object χ we shall let $var(\chi)$ denote the set of free variables in χ.

Definition 2.1 (Constraint System) A constraint system is a pair $(D, \vdash)$ satisfying the following conditions:

1. D is a set of first-order formulas such that $D = D^{\wedge,\exists}$
2. $\vdash \subseteq Fin(D) \times D$ satisfies the following inference rules. (In the following a set $\{d_1, \ldots, d_n\} \in Fin(D)$ shall be written $(d_1, \ldots, d_n)$.)

$$\Gamma, d \vdash d \;\; (\mathbf{Struct}) \qquad \frac{\Gamma_1 \vdash d \quad \Gamma_2, d \vdash e}{\Gamma_1, \Gamma_2 \vdash e} \; (\mathbf{Cut})$$

$$\frac{\Gamma, d, e \vdash f}{\Gamma, d \wedge e \vdash f} \; (\wedge \vdash) \qquad \frac{\Gamma \vdash d \quad \Gamma \vdash e}{\Gamma \vdash d \wedge e} \; (\vdash \wedge)$$

$$\frac{\Gamma, d \vdash e}{\Gamma, \exists X.\, d \vdash e} \; (\exists \vdash) \qquad \frac{\Gamma \vdash d[t/X]}{\Gamma \vdash \exists X.\, d} \; (\vdash \exists)$$

   In $(\exists \vdash)$, X is assumed not free in Γ, e.

3. $\vdash$ is *generic*: that is $\Gamma[t/X] \vdash d[t/X]$ whenever $\Gamma \vdash d$, for any term t.

□

In the following, we shall say a relation "is closed under the basic inference rules", to mean that it is closed under (**Struct**), (**Cut**), $\wedge \vdash$, $\vdash \wedge$, $\exists \vdash$ and $\vdash \exists$. Also, we shall extend $\vdash$ to a relation on $Fin(D) \times Fin(D)$ by: $u \vdash v$ iff $u \vdash d$ for all $d \in v$. For any $s \subseteq D$, we shall let $\bar{s} \subseteq D$ be the smallest set containing s and closed under the given entailment relation.

Above, we could have equivalently used the weaker structural rule that $\Gamma, d \vdash d$ whenever Γ is a set of atomic formulas and d is an atomic formula.

Example 2.1 (The trivial constraint system.) $\bot = (\emptyset, \emptyset)$ is a constraint system. □

Example 2.2 Any intuitionistic (or classical) theory T generates a constraint system. Take D to be any subset of formulas in the vocabulary of T, closed under existential quantification and conjunction, and define $\Gamma \vdash' d$ iff Γ entails d in the logic, with non-logical axioms T. Then $(D, \vdash')$ can be verified to be a constraint system. □

Pre-constraint systems. In the following we shall need a quick way to construct constraint systems. The following theorem is very helpful for this purpose since it establishes some simple conditions that need to be checked in order to verify that we have a constraint system. Essentially, the point of the theorem is that the extra conjunctive and existential structure demanded in a constraint system can be constructed "generically" for any entailment system satisfying certain reasonable conditions.

Definition 2.2 (Pre-constraint systems) A pre-constraint system is a pair $(D, \vdash)$ where D is a set of formulas and $\vdash \subseteq Fin(D) \times D$ satisfies (**Struct**), (**Cut**) and is generic. A pre-constraint system is *propositional* if D is a set of propositional formulas. □

Propositional pre-constraint systems are nothing more than information systems with their consistency structure removed, as the following proposition shows:

Proposition 2.1 *$(D, \triangle, \mathrm{Fin}(D), \vdash)$ is an information system iff $(D \cup \{\triangle\}, \vdash \cup \{\emptyset, \triangle\})$ is a propositional pre-constraint system.*

The proof is a straightforward checking of all the conditions involved.

Theorem 2.2 *Let $(D, \vdash')$ be a pre-constraint system. Let $\vdash \subseteq \mathrm{Fin}(D^{\wedge,\exists}) \times D^{\wedge,\exists}$ be the smallest relation containing $\vdash'$ and closed under the basic inference rules. Then:*

1. *$(D^{\wedge,\exists}, \vdash)$ is a constraint system.*
2. *$\vdash$ is a conservative extension of $\vdash'$. That is:*

$$\Gamma \vdash d, \Gamma \in \mathrm{Fin}(D), d \in D \iff \Gamma \vdash' d$$

The proof of (1) is straightforward (only genericity needs to be checked). The proof of (2) is a consequence of a cut-elimination theorem that all $\vdash$-cuts can be eliminated in a proof of a $\vdash$-sequent, if the non-logical axioms satisfy the given properties. The theorem holds even if D already has $\exists$ and $\wedge$, provided that its $\exists$ and $\wedge$ are treated as being syntactically different from the existential quantifier and conjunction introduced in the construction.

Dependency. We now introduce a central concept that considerably simplifies and smoothens subsequent presentation.

Let s be an arbitrary set of constraints. What set of variables does it depend on? Certainly, this set must be contained in the set of variables occurring in s. When can s not depend on a variable occurring in s? Consider a constraint system with the sequent $a(X) \vdash b(Y)$. Any entailment-closed set s containing $a(X)$ must also contain $b(Z)$, for every variable Z. But the dependence on all these variables Z different from X is generic, since s depends on *all* variables in exactly the same way. This motivates the following definition.

Definition 2.3 (dep(s)) Let $s \subseteq D$ be a set of constraints. s is said to be *dependent* on a variable X if $e[t/X] \notin s$ for some $e \in s$ and some term t. The collection of all such variables is denoted $dep(s)$. □

Note that $dep(s) \subseteq var(s)$, and $dep(D) = \emptyset$. Further, if D is a set of propositional formulas, then $dep(s) = \emptyset$ for every $s \subseteq D$. The most crucial property of $dep(s)$ is that it is preserved by entailment, in the following sense:

Proposition 2.3 *Let $\mathcal{A} = (D, \vdash)$ be a constraint system. For any $s \subseteq D$,* $\mathrm{dep}(\bar{s}) \subseteq \mathrm{dep}(s)$.

Proof 2.3 Suppose $X \in dep(\bar{(s)}) \setminus dep(s)$. Let $d \in \bar{s}$ be some constraint such that $X \in vard$. Then there is a set $\Gamma \subseteq s$ such that $\Gamma \vdash d$. But $X \notin dep(s)$ implies that for any t, $Gamma[t/X] \in s$. Since $\vdash$ is generic, we have, for any t, $\Gamma[t/X] \vdash d[t/X]$, so that $d[t/X] \in \bar{s}$. Thus, for every $d \in \bar{s}$ such that $X \in vard$, $d[t/X] \in \bar{s}$. But this contradicts our assumption that $X \in dep(s)$. □

Definition 2.4 (V-constraint.) Say that $s \subseteq D$ is a V-constraint if s is entailment-closed, and $dep(s) \subseteq V$. Let $|D|_V$ be the set of all V-constraints of D, and $|D| = \cup_{V \in Fin(Var)} |D|_V$. □

Theorem 2.4 *Let $\mathcal{A} = (D_{\mathcal{A}}, \vdash_{\mathcal{A}})$ be an arbitrary constraint system. For every finite set of variables V, $|D|_V$ is a complete, ω-algebraic lattice, with sups given by closure of the union, and infs given by intersection. The finite elements of $|D|_V$ are exactly those generated by the entailment-closure of finite sets of constraints whose dependent variables are contained in V.*

Note that $|D|$ may not be a complete lattice, since it is possible to have an increasing chain of sets of constraints such that the set of dependent variables of the union is infinite. This shall be of no concern to us, however, since a computational agent will be able to check and impose constraints on only a finite number of variables. Hence the denotation of an agent will be given by some function on $|D|_V$, for some $V \in Fin(Var)$.

The total elements are of less interest to us, since interesting states of the computation are typically described by partial elements.

2.1 Morphisms of constraint systems

We would like to take the morphisms between constraint systems to be information transducers: given some input constraints, such transducers may produce some other constraints. On very natural grounds, it is reasonable to require that such transducers be monotone and continuous in the information ordering. However, because of the first-order structure we shall require more: we shall require that the morphisms be *generic* in all but finitely many variables. To be generic in a variable X means that if the transducer can produce the information d on input c, then it can also produce the information $d[t/X]$ for input

$c[t/X]$, for any t. For $r \subseteq Fin(D_1) \times Fin(D_2)$ for some sets D_1, D_2 of formulas, say that $X \in dep(r)$ iff there exist $u \in Fin(D_1)$ and $v \in Fin(D_2)$, and a term t such that $u\ r\ v$ but not $u[t/X]\ r\ v[t/X]$. (This agrees with the definition for $dep()$ we gave earlier, if we think of r as a set of formulas (u, v).)

Definition 2.5 (Approximable maps.) An approximable map $r : \mathcal{A} \to \mathcal{B}$ is a relation $r \subseteq Fin(D_{\mathcal{A}}) \times Fin(D_{\mathcal{B}})$ such that

1. $\emptyset\ r\ \emptyset$
2. $u\ r\ v_1 \cup v_2$ whenever $u\ r\ v_1$ and $u\ r\ v_2$,
3. $u'\ r\ v'$ whenever $u' \vdash_{\mathcal{A}} u$, $u\ r\ v$ and $v \vdash_{\mathcal{B}} v'$, and,
4. $dep(r)$ is finite.

□

The identity approximable map on a constraint system $\mathcal{A}$ is just $\mathbf{Id}_{\mathcal{A}} \stackrel{\triangle}{=} \vdash_{\mathcal{A}}$; $dep(\mathbf{Id}) = \emptyset$ since $\vdash_{\mathcal{A}}$ is generic. The composition of two morphisms is given by relational composition — it is easy to check that $dep(r \circ r') \subseteq dep(r) \cup dep(r')$, and the other conditions to be checked have the same proof as in the propositional case (that of information systems).

A minor aside. Since constraint systems are closed under conjunction, we might be tempted to consider approximable maps to be relations on constraints, rather than on finite sets of constraints. For example, we could then do away with Condition 1, and regard $\emptyset$ as a legitimate map. However, this notion of morphisms would give rise to a different category, in which there is a unique morphism — $\emptyset!$ — *to* and *from* the trivial constraint system $\perp$, which thus becomes an initial and final object.

The definition of approximable maps is a smooth generalization of the definition for the propositional case: only Condition (4) is additional (and vacuously true in the propositional case). Therefore, morphisms between information systems *qua* information systems are exactly the morphisms between information systems *qua* constraint systems. We have:

Theorem 2.5 *Constraint systems with approximable maps as morphisms form a category (to be denoted* **ConstSys***), with terminal object $\perp$. Furthermore,* **ConstSys** *contains* **InfoSys***, the category of information systems and approximable maps, as a full sub-category.*

Every approximable map $r : \mathcal{A} \to \mathcal{B}$ determines a continuous function $|r| : |D_{\mathcal{A}}|_V \to |D_{\mathcal{B}}|_{V \cup dep(r)}$ (for V any finite set of variables) given by:

$$|r|(z) = \bigcup \{v \mid \exists u \subseteq z.\ u\ r\ v\}$$

In contrast with (the propositional) information systems, constraint systems come equipped with approximable maps corresponding to existential quantification:

Proposition 2.6 *Let $\mathcal{A}$ be a constraint system. Let $\{d_0, \ldots, d_{n-1}\}$ $(\exists X)$ v iff $n > 0$ and $\exists X. d_0 \wedge \ldots \wedge d_{n-1} \vdash_{\mathcal{A}} v$, or $n = 0$ and $\vdash_{\mathcal{A}} v$. Then $\exists X$ is an approximable map on $\mathcal{A}$, with* $\mathrm{dep}(\exists X) = \emptyset$.

2.2 Ordering on constraint systems

Definition 2.6 (Subsystem) For $\mathcal{A}, \mathcal{B}$ constraint systems, define $\mathcal{A} \trianglelefteq \mathcal{B}$ iff

1. $D_{\mathcal{A}} \subseteq D_{\mathcal{B}}$, and,
2. $\Gamma \vdash_{\mathcal{A}} d$ iff $\Gamma \subseteq D_{\mathcal{A}}$ and $d \in D_{\mathcal{A}}$ and $\Gamma \vdash_{\mathcal{B}} d$.

$\square$

Essentially, $\mathcal{A}$ is a subsystem of $\mathcal{B}$ iff A is a restriction of $\mathcal{B}$ to the tokens of $\mathcal{A}$.

Theorem 2.7 *The relation $\trianglelefteq$ on* **ConstSys** *is a partial order with least element $\perp = (\emptyset, \emptyset)$. Moreover, for any ω-chain*

$$\mathcal{A}_0 \trianglelefteq \mathcal{A}_1 \trianglelefteq \ldots \trianglelefteq \mathcal{A}_i \trianglelefteq \ldots$$

in **ConstSys**, *there exists a least upper bound, given by*

$$\bigcup_i \mathcal{A}_i = (\bigcup_i D_{\mathcal{A}_i}, \bigcup_i \vdash_{\mathcal{A}_i})$$

The main thing to check is the above construction produces a constraint system. For this the finitariness of entailment is crucial, the details are not interesting. That the constraint system so produced must be the least upper bound is obvious.

With this definition in hand, the rest of the discussion in [LW84, Section 3] immediately generalizes to the setting at hand. Namely, we can define the notion of monotonicity and continuity of an operation F on constraint systems, show that contintuity of an operation from a product is equivalent to continuity in each argument of the product separately (this allows simultaneous recursive domain equations to be handled). Thus we can expect to obtain least fixed point solutions for any system of "domain equations" built using continuous operations on constraint systems.

2.3 Constructions on constraint systems

What constructions are available on constraint systems? Briefly, all the operations available on (propositional) information systems (lifting, sums, products, etc.) can also be defined on constraint systems. We focus on the product and exponentiation construction to establish that central point of the paper, that the category of constraint systems is Cartesian-closed.

Definition 2.7 (Product) The product of two constraint systems $\mathcal{A} = (D_{\mathcal{A}}, \vdash_{\mathcal{A}})$ and $\mathcal{B} = (D_{\mathcal{B}}, \vdash_{\mathcal{B}})$ is given by $\mathcal{A} \times \mathcal{B} = (D, \vdash)$ where

- $D = \{(d, \triangle) \mid d \in D_{\mathcal{A}}\} \cup \{(\triangle, d) \mid d \in D_{\mathcal{B}}\}^{\wedge,\exists}$, for some token $\triangle \notin D_{\mathcal{A}} \cup D_{\mathcal{B}}$,
- $\vdash$ is the smallest relation closed under the basic inference rules and the rules:

$$\frac{d_0, \ldots, d_{n-1} \vdash_{\mathcal{A}} d}{\Gamma, (d_0, \triangle), \ldots, (d_{n-1}, \triangle) \vdash (d, \triangle)} \ (\times_1)$$

$$\frac{d_0, \ldots, d_{n-1} \vdash_{\mathcal{B}} d}{\Gamma, (\triangle, d_0), \ldots, (\triangle, d_{n-1}) \vdash (\triangle, d)} \ (\times_2)$$

□

That the above construction defines a constraint system is easy to verify using Theorem 2.2 on the evident pre-constraint system.

Furthermore, we have the following basic proposition:

Proposition 2.8 *For any finite set of variables V, $z \in |\mathcal{A} \times \mathcal{B}|_V$ iff $\{d \mid (d, \triangle) \in z\} \in |\mathcal{A}|_V$ and $\{d \mid (\triangle, d) \in z\} \in |\mathcal{B}|_V$. Further, $\_ \times \_$ is a continuous operation on constraint systems, and the categorical product in* **ConstSys**.

It is straightforward to verify the various conditions required by the proposition.

Definition 2.8 (Function Space) The function space of two constraint systems $\mathcal{A} = (D_{\mathcal{A}}, \vdash_{\mathcal{A}})$ and $\mathcal{B} = (D_{\mathcal{B}}, \vdash_{\mathcal{B}})$ is given by $\mathcal{A} \to \mathcal{B} = (C, \vdash)$ where:

- $C = Fin(D_{\mathcal{A}}) \times Fin(D_{\mathcal{B}})$,
- $\vdash$ is the smallest relation closed under the basic inference rules and the rules:

$$\frac{d \vdash_{\mathcal{A}} d_0 \wedge \ldots \wedge d_{n-1} \quad e_0 \wedge \ldots \wedge e_{n-1} \vdash_{\mathcal{B}} e}{\Gamma, (d_0, e_0), \ldots, (d_{n-1}, e_{n-1}) \vdash_{\mathcal{A} \to \mathcal{B}} (d, e)} \ (\to)$$

□

As before, the well-definedness of the above definition is easy to establish using Theorem 2.2 on the evident pre-constraint system.

Proposition 2.9 *For $\mathcal{A}, \mathcal{B}$ constraint systems, $|\mathcal{A} \to \mathcal{B}|_V$ is exactly the set of approximable maps $r : \mathcal{A} \to \mathcal{B}$ with $\mathrm{dep}(r) \subseteq V$. Further, $\_ \to \_$ is a continuous operation on constraint systems, and is the categorical exponentiation.*

The proofs of the first two assertions are straightforward. To show the latter, we have to exhibit a morphism

$$eval : ((\mathcal{B} \rightarrow \mathcal{C}) \times \mathcal{B}) \rightarrow \mathcal{C}$$

such that for any morphism $g : (\mathcal{A} \times \mathcal{B}) \rightarrow \mathcal{C}$ there is a unique arrow $curry(g) : \mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{C})$ such that

$$eval \circ ((curry(g) \circ \pi^1{}_{\mathcal{A}\times\mathcal{B}}) \times \pi^2_{\mathcal{A}\times\mathcal{B}}) = g$$

The definitions of *eval* and *curry* are standard — the only novel part is verifying that they satisfy the genericity condition. But it is not difficult to show that $dep(curry) = dep(eval) = \emptyset$.

Thus we have:

Theorem 2.10 *The category* **ConstSys** *is cartesian-closed and admits least fixed-point solutions of recursive domain equations built using* $\trianglelefteq$*-continuous operations on* **ConstSys**.

Acknowledgements. Conversations with Dana Scott, Prakash Panangaden and Patrick Lincoln have been very helpful.

References

[Abr91] Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 1991.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 111–119. ACM, January 1987.

[LW84] Kim G. Larsen and Glynn Winskel. Using information systems to solve recursive domain equations effectively. In G. Kahn, D.B.MacQueen, and G.Plotkin, editors, *Semantics of Data Types, International Symposium, Sophia Antipolis*, LNCS 173. Springer Verlag, 1984.

[PSSS91] Prakash Panangaden, Vijay Saraswat, Robert Seely, and Phil Scott. What is a constraint system? Submitted for publication, 1991.

[Sar89] Vijay A. Saraswat. *Concurrent Constraint Programming Languages*. PhD thesis, Carnegie-Mellon University, January 1989. To appear, Doctoral Dissertation Award and Logic Programming Series, MIT Press, 1990.

[Sar91] Vijay A. Saraswat. Concurrency in the λ-calculus: Higher-order Concurrent Constraint Programming. Submitted for publication, 1991.

[Sco82] Dana S. Scott. Domains for denotational semantics. In *Proceedings of ICALP*, 1982.

[SRP91] Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. Semantic foundations of concurrent constraint programming. In *Proceedings of Eighteenth ACM Symposium on Principles of Programming Languages, Orlando*, January 1991.