Abstract Constraint Data Types

José Luiz Fiadeiro[1] and Fernando Orejas[2]

[1]Dep. of Computer Science, Royal Holloway University of London,
Egham TW20 0EX, UK
jose.fiadeiro@rhul.ac.uk

[2] Dep. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya,
08034 Barcelona, Spain
orejas@lsi.upc.edu

Abstract. Martin Wirsing is one of the earliest contributors to the area of Algebraic Specification (e.g., [2]), which he explored in a variety of domains over many years. Throughout his career, he has also inspired countless researchers in related areas. This paper is inspired by one of the domains that he explored thirty years or so after his first contributions when leading the FET Integrated Project SENSORIA [14]: the use of constraint systems to deal with non-functional requirements and preferences [13,8]. Following in his footsteps, we provide an extension of the traditional notion of algebraic data type specification to encompass soft-constraints as formalised in [1]. Finally, we relate this extension with institutions [6] and recent work on graded consequence in institutions [3].

1 Introduction

Service-Oriented Architecture (SOA) [10] is a paradigm for the flexible construction of systems based on the dynamic interconnection of components. This interconnection takes place when a given component (*the requester*) needs to discover another component that can provide a service that it needs, i.e., a component (*the provider*) that, through an interface, offers the properties required by the requester. In addition to the usual functional properties, components may express preferences in their interfaces, in which case the requester will choose a provider that can maximise the way those preferences are satisfied.

Interfaces are abstractions through which components can express properties that are independent of their implementations. Algebraic specification of abstract data types [12] are one of the most established formalisms in which interfaces can be defined. However, they are limited to functional properties of the input/output behaviour of the operations that components implement. In this paper, we extend algebraic specifications of component interfaces so that preferences can be expressed as *constraints* and matching can be formalised in terms of *constraint satisfaction and optimisation*.

In [1], Bistarelli, Montanari, and Rossi define a general framework for the definition of constraint systems of several kinds. More precisely, their approach allows us to describe both hard and (different types of) soft constraint systems.

The idea is to consider that constraint values form a semiring, where 0 represents unsatisfiability, 1 represents satisfaction and the rest of the values represent the different degrees of satisfiability of the given constraint system. The approach outlined in this paper combines the ideas presented in [1] with algebraic specification to include preferences in component interfaces. Our ideas are presented using the specification of a travel request as a running example.

The paper is organized as follows. In Section 2, we recall some basic elements of algebraic specification theory (which does not dispense consulting [12]). In Section 3, we extend algebraic specifications for the specification of constraints. Then, in Section 4 we study how we can combine constraint specifications. Section 5 is dedicated to presenting our ideas in the framework of institutions. Finally, in Section 6 we draw some conclusions.

2 Basic algebraic concepts and notation

We assume that the reader has some familiarity with category theory (for example, at the level of the first chapters of [5].)

A signature Σ is a pair $\langle S, \Omega \rangle$ where S is a finite set of sorts, and Ω is a finite family of sets of operation and predicate symbols typed over sorts. A Σ-algebra A consists of an S-indexed family of sets $\{A_s\}_{s \in S}$ and a function op_A (resp., a relation pr_A) for each operation symbol op (resp., predicate symbol pr) in the signature[1]. A Σ-homomorphism $h\colon A \to A'$ consists of an S-indexed family of functions $\{h_s : A_s \to A'_s\}_{s \in S}$ commuting with the functions and preserving the relations. Σ-algebras and Σ-homomorphisms form the category $\mathbf{Alg_\Sigma}$.

Given a signature Σ, we denote by T_Σ the term algebra, which consists of all the possible Σ-(ground) terms – where a ground term is either a nullary function symbol or an expression of an operation symbol being applied to ground terms of the types required by the operation. Given any Σ-algebra A there is a unique homomorphism $h_A\colon T_\Sigma \to A$ through which h_A yields the value of every term of sort $s \in S$ in A_s.

Given a set X of variables typed over S, we denote by $T_\Sigma(X)$ the algebra of all Σ-terms with variables in X, and given a variable assignment $\sigma\colon X \to A$, this assignment extends to a unique homomorphism $\sigma^\#\colon T_\Sigma(X) \to A$ yielding the value of each term after the replacement of each variable x by its value $\sigma(x)$. In particular, when an assignment is defined over the term algebra, i.e. $\sigma\colon X \to T_\Sigma$, then $\sigma^\#(t)$ denotes the term obtained by substituting each variable x in t by the term $\sigma(x)$. However, for simplicity, even if it is an abuse of notation, we will write $\sigma(t)$ instead of $\sigma^\#(t)$.

Given a signature $\Sigma = \langle S, \Omega \rangle$ and a set X of variables typed over S, we can build sentences, which are either equalities of the form $(t_1 =_s t_2)$ where t_1 and t_2 are terms of sort s, or predicates of the form $p(t_1, \cdots, t_n)$, or a result of applying the usual Boolean connectives over sentences. All sentences are implicitly

[1] Predicates are not part of the usual staple of algebraic data type specification but they are convenient for our purposes in this paper.

universally quantified[2], i.e., a sentence is true over a Σ-algebra A if, for each possible variable assignment $\sigma\colon X \to A$, the sentence is true, i.e., the two terms of an equality $(t_1 =_s t_2)$ have the same values — $\sigma(t_1)$ is the same as $\sigma(t_2)$, the value of the terms $t_1, \cdots, t_n$ of a predicate $p(t_1, \cdots, t_n)$ belong to the relation p_A — $(\sigma(t_1), \cdots, \sigma(t_n)) \in p_A$, or the Boolean operators return true when applied to the sentences that they connect.

In this paper, we use distinguished variables for defining constraints and use them to extend signatures and algebras. Given a set V of variables typed over S, we denote by $\Sigma \cup V$ the extension of Σ with the variables taken as unary operation symbols and, given a Σ-algebra A and an assignment $\chi\colon V \to A$, we denote by $A \cup \chi$ the extension of A to $\Sigma \cup V$ that coincides with χ on V. New 'normal' variables can be superposed using the usual construction of terms with variables as explained above.

3 Extending algebraic specifications with constraints

We put forward a number of definitions that relate to the so-called c-semiring approach to constraint satisfaction and optimisation proposed in [1]. As explained therein, that approach is quite general and allows us to work with constraints of different kinds, both hard and 'soft', the latter in many grades (fuzzy, weighted, and so on). The c-semiring approach supports selection based on a characterisation of 'best solution' supported by multi-dimensional criteria, for example minimizing the cost of a resource while maximizing the work it supports.

We recall that a c-semiring is a commutative idempotent semiring where addition is extended to infinite sets. In summary, a c-semiring is a tuple $\langle \mathtt{R}, \vee, \wedge, 0, 1\rangle$ such that:

- $\mathtt{R}$ is a set and 0, 1 are elements of that set.
- $\vee$ is a commutative, associative, idempotent operation over subsets of $\mathtt{R}$ with unit 0; we use $\sum$ for sums over sets and reserve $\vee$ for the binary case.
- $\wedge$ is a binary, commutative, associative operation with unit 1 for which 0 is absorbing, i.e., $(a \wedge 0 = 0)$ for every a; we use $\prod$ for products over finite sets.
- $\wedge$ distributes over $\vee$.

The intuition is that $\mathtt{R}$ — the *domain* of the semiring — represents a space of degrees of satisfaction, for example the set $\{0, 1\}$ for 'yes'/'no' or the interval $[0, 1]$ for intermediate degrees of satisfaction (which gives us a constraint model that is richer than Boolean algebra). The operations $\wedge$ and $\vee$ are used for composition (conjunction) and choice, respectively.

A partial order $\leq_R$ (of satisfaction) is defined over $\mathtt{R}$ as follows: $a \leq_R b$ iff $a \vee b = b$. That is, b is better than a iff the choice between a and b is b. It follows that 0 is worse than any other degree of satisfaction — it represents dissatisfaction, and 1 is better than any other degree of satisfaction — it represents total satisfaction. This partial order defines a complete distributive lattice.

[2] Note that implicit quantification in many-sorted equational logic raises problems at the level of proof theory, which we do not discuss herein [7].

In order to define specifications that capture constraints interpreted over c-semirings, we extend the traditional notion of signature as follows.

Definition 1 (constraint signature) *A* constraint signature *(or c-signature for short) is a tuple* $\langle S, \Omega, V, sat, 0, 1, \leq \rangle$ *where*

- $\langle S, \Omega \rangle$ *is a signature as recalled above.*
- V *is a finite set of (constraint) variables (c-variables for short) disjoint from* Ω*.*
- $sat \in S$ *is a distinguished sort,* $0, 1 \in \Omega_{sat}$ *are distinguished constants, and* $\leq$ *is a distinguished predicate symbol over* sat*.*

For simplicity, we will often use Σ to denote both a c-signature and its underlying algebraic signature and denote by V_Σ its set of c-variables. We will denote by Σ^c the algebraic signature $\langle S, \Omega \cup V_\Sigma \rangle$.

Definition 2 (constraint algebra) *Let* Σ *be a c-signature. A* constraint algebra *(c-algebra for short) for* Σ *is a triple* $\langle A, R, \chi \rangle$ *consisting of:*

- *A c-semiring* R*.*
- *A* $\langle S, \Omega \rangle$*-algebra* A *such that* A_{sat} *is the domain of* R*,* 0_A *and* 1_A *are the units of* R*, and* $\leq_A$ *is the partial order defined by* R*.*
- *An assignment* $\chi \colon V \to A$ *of values to the c-variables.*

Notice that, given a term t in T_{Σ^c}, $\chi(t)$ is the value that is assigned to t in the extended algebra $A \cup \chi$.

We now adapt to our algebraic setting the concepts put forward in [1] for expressing constraints:

Definition 3 (constraints) *Let* Σ *be a c-signature.*

- *A* constraint *is a term* $q \in T_{\Sigma^c sat}$*, i.e., a ground term of sort* sat*.*
- *A* constraint problem *(c-problem for short)* C *is a finite set of constraints.*

In our running example, we consider the case of a customer who wants to book a flight. The data signature of our example could be as depicted in Fig. 1: it sets out the domain of airports, cities, airlines and flights that are relevant for a particular customer.

The c-signature of the customer, i.e., the one in which constraint variables are introduced, could then be the extension of flightDataSign depicted in Fig. 2. This signature includes three c-variables — flight, flightCost and payMode — meaning that the customer wants to optimise the choice of the flight and the payment mode. We use {DC, CC} as an abbreviation for a sort with two different constants DC and CC. In order to express the constraints that apply to that optimisation, three operations are declared for expressing preferences (i.e., operations of sort *sat*): airlinePref, stopsPref, distPref.

The constraints themselves are expressed as terms of type *sat*, for example:

Signature flightDataSign
Sorts nat, bool, city, airport, airline, money, flightCode
Opns distance : airport city → nat
cost : airport city → money
LHR, LGW, LTN, STN, BCN, GRO, ... : airport
Iberia, BritishAirways, EasyJet, RyanAir, Vueling, ... : airline
IB001, IB002, BA001, BA002, EZ001, RN001, VL001, ... : flightCode
Egham, Barcelona, ... : city
stops : flightCode → nat
airline : flightCode → airline
airDept, airDest : flightCode → airport

Fig. 1. The signature flightDataSign

Signature customerSign **extends** flightDataSign **with**
Opns departure, destination : city
totalCost : flightCode → money
airlinePref : airline → sat
payPref : {DC, CC} → sat
stopsPref : nat money → sat
distPref : nat → sat
c-Vars flight : flightCode; flightCost : money; payMode : {DC, CC}

Fig. 2. The signature customerSign

airlinePref(airline(flight)) — meaning that the customer has a preference on the airline.
payPref(payMode) — meaning that the customer has a preference on the payment mode.
stopsPref(stops(flight),totalCost(flight)) — meaning that the customer wishes to optimise the number of stops relative to the total cost of the journey.
distPref(distance(airDest(flight),destination)) — meaning that the customer wishes to optimise the distance between the destination airport and city.

We discuss now how these preferences are evaluated.

Definition 4 (constraint evaluation) *Let Σ be a c-signature and $\langle A, R, \chi\rangle$ a c-algebra for Σ.*

- *The degree of satisfaction of a constraint q is $\chi(q)$.*
- *Given a c-problem C:*
  - *The degree of satisfaction $\chi(C)$ of C is $\prod_{q\in C}\chi(q)$. That is, we take the minimum of the degrees of satisfaction that χ assigns to the constraints in C.*
  - *The* best level of consistency *of C over A and R is $\sum_{\chi:V\to A}\chi(C)$, which we denote by $blevel_{A,R}(C)$. That is, we take the maximum degree of satisfaction across all assignments.*
  - *A c-problem C is* consistent *over A and R iff $blevel_{A,R}(C) > 0$, i.e., if there is an assignment for which all constraints have a non-zero degree of satisfaction.*
  - *A* solution *to a consistent c-problem C over A and R is an assignment χ such that $\chi(C) > 0$. A* best solution *is an assignment χ such that $\chi(C) = blevel_{A,R}(C)$.*

We now consider specifications over a signature.

Definition 5 (constraint specification) *A* constraint specification *(c-spec for short) is a triple $\langle\Sigma, \Phi, C\rangle$ where Σ is a c-signature, Φ is a finite set of sentences over Σ^c and C is a finite set of constraints over Σ.*

A model *of $\langle\Sigma, \Phi, C\rangle$ is a c-algebra $\langle A, R, \chi\rangle$ such that $(A \cup \chi) \models_{\Sigma^c} \Phi$ and $\chi(C) > 0$, i.e., the sentences in Φ are true and χ is a solution to C.*

A best *model of $\langle\Sigma, \Phi, C\rangle$ is a model $\langle A, R, \chi\rangle$ such that $\chi(C) = blevel_{A,R}(C)$.*

Notice that, as usual in algebraic specifications, Φ may involve (data) variables, which should not be confused with the c-variables. The specification is quantified over the former but not the latter.

Consider again our running example. The specification of the underlying data type could be as depicted in Fig. 3. For simplicity, we use a tabular representation for groups of equations; for example, the specification would contain the equations distance(BCN,Barcelona)=10, cost(BCN,Barcelona)=5 and so on.

Specification flightData

Signature flightDataSign

Axioms distance(X,Y)=D, cost(X,Y)=C **where**:

X	Y	D	C
BCN	Barcelona	10	5
GRO	Barcelona	60	15
LHR	Egham	10	5
LGW	Egham	35	20
LTN	Egham	40	30
STN	Egham	70	50
...	...	...	...

airDept(X)=X1, airDest(X)=X2, airline(X)=X3, stops(X)=X4 **where**:

X	X1	X2	X3	X4
IB001	LHR	BCN	Iberia	0
IB002	LGW	BCN	Iberia	1
BA001	LHR	BCN	BritishAirways	0
BA002	LGW	BCN	BritishAirways	1
EZ001	LTN	BCN	EasyJet	0
RN001	STN	GRO	RyanAir	0
VL001	LGW	BCN	Vueling	0
...	...	...	...	...

Fig. 3. The specification flightData

The specification of the customer could be as depicted in Fig. 4 (note that all sentences are universally quantified). For example, the customer is not satisfied with any flight that has two or more stops, no matter the total cost; nor is the customer satisfied with any flight whose destination airport is fifty or more miles away from the destination city, though if that distance is less than 50 miles, the closer the better. The customer is also willing to pay 20% more for a non-stop flight but prefers the cheaper between any two flights with the same number of stops.

Specification customer **extends** flightData **with**
Signature customerSign
Axioms d, d', n, m, m' : nat
departure = Egham
destination = Barcelona
totalCost(F) = flightCost +
cost(airDest(F),destination) + cost(airDept(F),departure)
payPref(CC) > payPref(DC)
airlinePref(Iberia) > airlinePref(RyanAir)
airlinePref(Iberia) > airlinePref(EasyJet)
airlinePref(Iberia) = airlinePref(Vueling)
airlinePref(Iberia) = airlinePref(BritishAirways)
stopsPref(0,m) < stopsPref(1,m') **if** m > m'*1.2
stopsPref(0,m) > stopsPref(1,m') **if** m ≤ m'*1.2
stopsPref(n,m) > stopsPref(n,m') **if** m < m' ∧ n ≤ 1
stopsPref(n,m) = 0 **if** n ≥ 2
distPref(d) = 0 **if** d ≥ 50
distPref(d) > distPref(d') **if** d < d' < 50
Constraints
airlinePref(airline(flight))
payPref(payMode)
stopsPref(stops(flight),totalCost(flight))
distPref(distance(airDest(flight),destination))

Fig. 4. The specification customer

Notice that a specification does not necessarily fix a c-semiring: a specifier is more likely to express conditions on preferences, as is the case of customer, which will determine what c-semirings can be chosen to accommodate them.

Because customer does not have information on the actual cost of flights, we cannot get a best choice for the c-constraints flight, flightCost and payMode. In the next section we show how, by connecting the customer to a supplier (with flight costs and own preferences) it is more meaningful to compute a best choice, and also how to compare between different suppliers so that the one that offers the best solution can be chosen by the customer.

4 Composing specifications

We start by stating some of the category-theory properties of c-specs, which are useful to bring abstract constraint data types to the more established mathematical frameworks of algebraic specification (see [4,12]).

Definition 6 (morphisms of c-signatures) *A morphism of c-signatures* $\sigma\colon \Sigma \to \Sigma'$ *consists of:*

- *A morphism between the algebraic signatures that preserves the distinguished elements.*
- *A total function between the sets of c-variables.*

The image of a c-problem C *by a c-signature morphism* $\sigma\colon \Sigma \to \Sigma'$ *is* $\sigma(C)$ *where* $\sigma(\langle V_k, q_k\rangle)$ *is the* Σ'*-constraint* $\langle \sigma(V_k), \sigma(q_k)\rangle$.

Proposition 7 (category of c-signatures) *C-signatures form a finitely co-complete category, which we denote by* c-SIG. *The c-signature* $\langle \{sat\}, 0, 1, \leq, \emptyset\rangle$, *which we denote by* $\Sigma_\emptyset$, *is an initial object and pushouts operate independently over the algebraic signature and the c-variables.*

Proposition and Definition 8 (category of c-specs) *A* morphism of constraint specifications $\sigma\colon \langle \Sigma, \Phi, C\rangle \to \langle \Sigma', \Phi', C'\rangle$ *is a morphism of c-signatures* $\sigma\colon \Sigma \to \Sigma'$ *such that* $\Phi' \models_{\Sigma'^c} \sigma(\Phi)$ *and* $\sigma(C) \subseteq C'$.

Morphisms of c-specs define a category, which we denote by CCS. *This category is finitely co-complete.*

Proof. That a category is defined is trivial to prove. Finite co-completeness is proved as follows:

existence of initial objects *It is easy to prove that* $\langle \Sigma_\emptyset, \emptyset, \emptyset\rangle$, *where* $\Sigma_\emptyset = \langle \{sat\}, \{0 : sat,\ 1 : sat,\ \leq\colon sat\ sat\}, \emptyset, sat, 0, 1, \leq\rangle$ *is the initial c-signature, is an initial object of CCS.*

existence of pushouts *Let* $\sigma_i\colon \langle \Sigma, \Phi, C\rangle \to \langle \Sigma_i, \Phi_i, C_i\rangle$ *(*$i = 1, 2$*) be two morphisms and* $\mu_i\colon \Sigma_i \to \Sigma'$ *a pushout of the corresponding c-signature morphisms. Then,* $\mu_i\colon \langle \Sigma_i, \Phi_i, p_i\rangle \to \langle \Sigma', \mu_1(\Phi_1) \cup \mu_2(\Phi_2), \mu_1(C_1) \cup \mu_2(C_2)\rangle$ *is easily proved to be a pushout of c-specs.*

Pushouts compute amalgamated unions, which provide the means for composing specifications. The amalgamation is done over what is designated to be the 'intersection' of the two signatures, i.e., the sorts, operations, predicates and c-variables that they are designated to share (composition is not based on syntactic sharing, i.e., names are not considered to be universal but local to specifications). The exceptions are the sort *sat* and the constants 0 and 1 of the c-semiring, which are shared by construction – i.e., the initial c-signature $\Sigma_\emptyset$ is shared by all c-specifications. An example of composition is given below.

Definition 9 (reducts) *Let σ: $\Sigma \to \Sigma'$ be a morphism of c-signatures. The σ-reduct of a c-algebra $\langle A', R', \chi' \rangle$ for Σ' is the c-algebra $\langle A'|_\sigma, R'|_\sigma, \chi'|_\sigma \rangle$ for Σ where:*

- *$A'|_\sigma$ is the σ-reduct of A' in the usual algebraic sense, i.e., $(A'|_\sigma)_s = A'_{\sigma(s)}$ for every sort s and, for every operation or predicate op, $op_{A'|_\sigma} = \sigma(op)_{A'}$.*
- *$R'|_\sigma = R'$.*
- *$\chi'|_\sigma = \sigma; \chi'$.*

That is, models are translated back along a morphism by adopting the same data carriers and c-semiring, and giving symbols and variables at the source the interpretations that their translations have in the models being translated.

It is important to study how properties of models relate to those of their reducts:

Proposition and Definition 10 *Let σ: $\langle \Sigma, \Phi, C \rangle \to \langle \Sigma', \Phi', C' \rangle$ be a morphism of c-specs and $\langle A', R', \chi' \rangle$ a model of $\langle \Sigma', \Phi', C' \rangle$. The following properties hold:*

1. *$\langle A'|_\sigma, R'|_\sigma, \chi'|_\sigma \rangle$ is a model of $\langle \Sigma, \Phi, C \rangle$.*
2. *$\chi'(\sigma(C)) = \chi'|_\sigma(C) \geq \chi'(C')$.*
3. *$blevel_{A'|_\sigma, R'|_\sigma}(C) \geq blevel_{A', R'}(\sigma(C)) \geq blevel_{A', R'}(C')$.*
4. *If σ is injective on V_Σ, then $blevel_{A'|_\sigma, R'|_\sigma}(C) = blevel_{A', R'}(\sigma(C))$.*
5. *If C' is consistent over $\langle A', R' \rangle$, then C is consistent over $\langle A'|_\sigma, R'|_\sigma \rangle$ and, for every solution χ' of C', $\chi'|_\sigma$ is a solution of C.*

Proof. All properties are easy to prove.

Notice that the difference between 3 and 4 is that, if σ is not injective on V_Σ, the range of assignments to c-variables allowed by Σ' is more restricted than that allowed by Σ — certain c-variables are identified through σ. Property 5 is particularly important because it shows that consistency and solutions of c-problems are preserved by reducts, i.e., by extending a specification, one does not create new solutions for or make existing c-problems consistent; naturally, one may lose solutions or make c-problems inconsistent because new constraints can be introduced through the extension.

Let us analyze how we can check when a requester component and a provider component can be connected. If the constraint specification $\langle \Sigma, \Phi, C \rangle$ states the preferences and conditions defined by the requester, and $\langle \Sigma', \Phi', C' \rangle$ states the functionality offered and the conditions that the provider can accept, then what we need is that requester and the provider constraints are consistent. More precisely, that if we put together the two specifications:

$$\begin{array}{ccc} \langle \Sigma_0, \emptyset, \emptyset \rangle & \longrightarrow & \langle \Sigma, \Phi, C \rangle \\ \downarrow & po & \downarrow \\ \langle \Sigma', \Phi', C' \rangle & \longrightarrow & \langle \Sigma'', \Phi'', C'' \rangle \end{array}$$

where Σ_0 is the common subsignature of Σ and Σ', then the resulting specification must be satisfiable, meaning that there must be a model of $\langle \Sigma'', \Phi'', C'' \rangle$.

In order to illustrate this construction, we connect customers with suppliers. Consider the following specification of a supplier given in Fig. 6 based on the signature given in Fig. 5. A supplier has a number of flights available for sale, for

Signature supplierSign **extends** flightDataSign **with**
Opns price : flightCode money payMode → sat
available : flightCode → sat
c-Vars flight : flightCode; flightCost : money; payMode : {DC, CC}

Fig. 5. The c-signature supplierSign

Specification supplier **extends** flightData **with**
Signature supplierSign
Axioms available(F)=1 **iff** F∈{IB001,IB002,BA001,BA002,EZ001,RN001}
price(F,M,C)=S **where**:

id	F	M	C	S
A1	IB001	120	DC	1
A2	IB001	120	CC	1
A3	IB002	150	DC	1
A4	IB002	150	CC	1
A5	BA001	250	DC	1
A6	BA001	250	CC	1
A7	BA002	145	DC	1
A8	BA002	145	CC	1
A9	EZ001	60	DC	1
A10	EZ001	65	CC	1
A11	RN001	40	DC	1
A12	RN001	45	CC	1
	F	M	C	0

Constraints available(flight)
price(flight,flightCost,payMode)

Fig. 6. The specification supplier

each of which it has a price depending on the payment mode. All the constraints are crisp meaning that the supplier will only accept to sell flights that it has available and for the stated prices.

As before, we have used a tabular form to simplify the specification. In addition, we have named equations (using the attribute *id*): this is just for convenience when discussing constraint optimisation and is not part of the formal specification, i.e., it has no semantics.

Consider now the amalgamated sum (pushout) of customer and supplier assuming that sorts, operations, predicates and c-variables with the same names are shared. The set of constraints is, as explained in Def. 8:

c_0 : airline(flight)
c_1 : payPref(payMode)
c_2 : stopsPref(stops(flight),totalCost(flight))
c_3 : distPref(distance(airDest(flight),destination))
c_4 : available(flight)
c_5 : price(flight,flightCost,payMode)

Again, we have named constraints to simplify the way we refer to them.

The crisp constraint c_4 reduces the space of solutions to the triples (F,M,C) — corresponding to flight code, flight cost and payment mode, respectively — on rows A1-A12. The constraint c_3 eliminates the triples (F,M,C) on rows A11-A12 from that set. For each of the remaining constraints, we can derive the following properties where we use = and > to compare the way those assignments order the satisfaction of the corresponding constraint:

c_0 : A1=A2=A3=A4 > A9=A10, and A5=A6=A7=A8 > A9=A10
c_1 : A2=A4=A6=A8=A10 > A1=A3=A5=A7=A9
c_2 : A9 > A10 > A1=A2 > A7=A8 > A3=A4 > A5=A6
c_3 : A1=A2=A3=A4=A5=A6=A7=A8=A9=A10
c_4 : A1=A2=A3=A4=A5=A6=A7=A8=A9=A10
c_5 : A1=A2=A3=A4=A5=A6=A7=A8=A9=A10

In order to calculate c_2, we computed the total costs as specified in the specification customer (see Fig. 4):

id	total cost	id	total cost	id	total cost	id	total cost
A1	130	A2	130	A3	175	A4	175
A5	260	A6	260	A7	160	A8	160
A9	95	A10	100	A11	105	A12	110

No best solution can be derived from these inequalities: for example, the paymode preference conflicts with those that relate costs. Notice that, for every algebra that satisfies the specification, a best solution can be obtained because a specific level of satisfaction is assigned to every constraint. What happens in this case is that there is no solution that is optimal for all such algebras.

In general, we can think that a customer could also wish to express an ordering of importance on constraints, for example that c_2 is the most important,

followed by c_3, then c_1, and then c_0. This can be achieved by means of another preference function, this time applied to *sat*:

$$\mathsf{constPref} : \mathsf{sat\ sat} \rightarrow \mathsf{sat}$$

axiomatized by

$\mathsf{constPref(N,M)} = 0$ **if** $\mathsf{N}{=}0$
$\mathsf{constPref(N,M)} < \mathsf{constPref(N',M)}$ **if** $\mathsf{N}{<}\mathsf{N'}$
$\mathsf{constPref(N,M)} > \mathsf{constPref(N',M)}$ **if** $\mathsf{N}{>}\mathsf{N'}{>}0$
$\mathsf{constPref(N,M)} > \mathsf{constPref(N',M')}$ **if** $\mathsf{N}{\neq}0 \wedge \mathsf{M}{>}\mathsf{M'}$

One would then replace c_0, c_1, c_2, c_3 by

$$c : \mathsf{constPref}(c_0,\mathsf{constPref}(c_1,\mathsf{constPref}(c_3,c_2)))$$

which would again exclude the triples on rows A11-A12 from the space of solutions and return:

c : A9 > A10 > A1 > A2 > A7 > A8 > A3 > A4 > A5 > A6
c_4 : A1=A2=A3=A4=A5=A6=A7=A8=A9=A10
c_5 : A1=A2=A3=A4=A5=A6=A7=A8=A9=A10

This time, there is a best solution for the customer: the triple (EZ001,95,DC).

Our framework can also be used for selecting a best supplier (if one exists), by analysing the composition of customer with that of every other supplier. For example. consider the following specification of a different supplier depicted in Fig. 7. The crisp constraint c_4 now reduces the space of solutions to the triples (F,M,C) on rows B1-B6. The constraint c_3 eliminates the triples (F,M,C) on rows B5-B6 from that set. The total costs are now:

id	total cost
B1	75
B2	80
B3	90
B4	95

For each of the remaining constraints, we can derive the following properties where we use the row numbers to refer to the triples:

c_0 : B1=B2 > B3=B4
c_1 : B2=B4 > B1=B3
c_2 : B1 > B2 > B3 > B4
c_3 : B1=B2=B3=B4
c_4 : B1=B2=B3=B4
c_5 : B1=B2=B3=B4

Applying the order on the customer's constraints we obtain:

Specification otherSupplier **extends** flightData **with**
Signature supplierSign
Axioms available(F)=1 **iff** F∈{EZ001,RN001,VL001}
price(F,M,C)=S **where**:

id	F	M	C	S
B1	VL001	40	DC	1
B2	VL001	45	CC	1
B3	EZ001	55	DC	1
B4	EZ001	60	CC	1
B5	RN001	40	DC	1
B6	RN001	45	CC	1
	F	M	C	0

Constraints available(flight)
price(flight,flightCost,payMode)

Fig. 7. The specification otherSupplier

c : B1 > B2 > B3 > B4
c_4 : B1=B2=B3=B4
c_5 : B1=B2=B3=B4

From this set we can derive that (VL001,40,DC) is the best solution.

Consider now the combined specifications of the customer with the two suppliers, sharing the data specification but nothing else. The amalgamation will distinguish the triples that result from one pairing from those resulting from the other pairing, leading effectively to the union of the two tables. This means that some entries are duplicated but this is how it should be because they refer to c-variables coming from different sources:

id	F	M	C	S		id	F	M	C	S
A1	IB001	120	DC	1		A2	IB001	120	CC	1
A3	IB002	150	DC	1		A4	IB002	150	CC	1
A5	BA001	250	DC	1		A6	BA001	250	CC	1
A7	BA002	145	DC	1		A8	BA002	145	CC	1
A9	EZ001	60	DC	1		A10	EZ001	65	CC	1
B1	VL001	40	DC	1		B2	VL001	45	CC	1
B3	EZ001	55	DC	1		B4	EZ001	60	CC	1

For each of the constraints, we derive:

c_0 : B1=B2=A1=A2=A3=A4 > A9=A10=B3=B4
and A5=A6=A7=A8 > A9=A10=B3=B4
c_1 : B2=B4=A2=A4=A6=A8=A10 > B1=B3=A1=A3=A5=A7=A9

c_2 : B1 > B2 > B3 > B4=A9 > A10 > A1=A2 > A7=A8 > A3=A4 > A5=A6
c_3 : B1=B2=B3=B4=A1=A2=A3=A4=A5=A6=A7=A8=A9=A10

If we use the ordering on the customer's constraints, then we get:

c : B1 > B2 > B3 > B4=A9 > A10 > A1=A2 > A7=A8 > A3=A4 > A5=A6

which means that, from the customer's point of view, the optimal solution is the triple (VL001,40,DC) and, therefore, the customer would prefer otherSupplier over supplier.

5 Relationship with institutions

Algebraic specification of abstract data types has traditionally been studied in the context of institutions [6,11]:

Definition 11 (Institution) *An* institution $\langle Sig, Sen, Mod, \models \rangle$ *consists of*

- *a category* Sig *of* signatures *and* signature morphisms,
- *a functor* $Sen : Sig \to Set$*, defining for every signature* Σ *the set* $Sen(\Sigma)$ *of* Σ*-sentences, and for every signature morphism* $\sigma : \Sigma \to \Sigma'$ *a* sentence translation map $Sen(\sigma) : Sen(\Sigma) \to Sen(\Sigma')$,
- *a functor* $Mod : Sig \to Cat^{op}$*, defining for every signature* Σ *the category* $Mod(\Sigma)$ *of* Σ*-models and* Σ*-model homomorphisms, and for every signature morphism* $\sigma : \Sigma \to \Sigma'$ *the* reduct functor $Mod(\sigma) : Mod(\Sigma') \to Mod(\Sigma)$,
- *a family of* satisfaction relations $\models_\Sigma \subseteq |Mod(\Sigma)| \times Sen(\Sigma)$*, indexed by signatures,*

such that the following satisfaction condition *holds:*

$$M' \models_{\Sigma'} Sen(\sigma)(\rho) \quad \textit{if and only if} \quad Mod(\sigma)(M') \models_\Sigma \rho,$$

for every signature morphism $\sigma : \Sigma \to \Sigma'$*,* Σ'*-model* M *and* Σ*-sentence* ρ*.*

The algebraic specification of abstract data types as recalled in Sect. 2 is a variant of the institutions reviewed in [11].

Proposition 12 *The extension to c-specifications as defined in Sect. 3 and 4 defines an institution:*

- *The category of signatures is as defined in Prop. 7;*
- *The sentence functor is the extension of classical conditional equational logic with c-constraints as defined in Def. 3, i.e., sentences are either conditional equations or c-constraints;*
- *The model functor is as in Def. 2 and Def. 9.*
- *The satisfaction relation is as usual on conditional equations and on c-constraints is defined by*

$$\langle A, R, \chi \rangle \models c \quad \text{iff} \quad \chi(c) > 0$$

The results presented in Sect. 4 about c-specifications actually follow from the fact that c-constraints define an institution.

Another way of defining an institution of c-constraints is by framing them in the context of institutions of graded consequence recently proposed by Razvan Diaconescu [3], which differ from institutions by letting the satisfaction relations take values in a space L, i.e., for every signature Σ,

$$\models_\Sigma : |Mod(\Sigma)| \times Sen(\Sigma) \to L$$

In this case, we would just have to take the domain of the c-semiring as the space L and interpret the sentences that are not c-constraints as having the c-semiring values 1 or 0 depending on whether they are satisfied or not satisfied in a model, respectively. That is, we would treat equations as crisp constraints.

6 Conclusions and further work

In this paper, we outlined a way in which specifications of abstract data types can be extended to accommodate constraint specification using the c-semiring approach proposed in [1]. This brings together two areas to which Martin Wirsing has made extensive contributions: algebraic specification theory (e.g., [12]) and the use of constraint systems to deal with non-functional requirements in service-oriented systems [13,14].

This work sets the stage for a more ambitious project of revisiting the formalism of symbolic graphs, proposed in [9], to use it as the basis for describing service-oriented systems, where a symbolic graph is a typed attributed graph together with a set of constraints. In particular, we can use symbolic graphs to describe the states of systems and symbolic graph transformation rules to describe computations including the interconnection of components. In this context, constraints could be used to describe quality of service requirements, so that computing service level agreements could be part of the computation associated with component interconnection.

Acknowledgments

This work was developed while J L Fiadeiro was on study leave at Universitat Politècnica de Catalunya with the generous support of the Dep. de Llenguatges i Sistemes Informàtics.

References

1. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.
2. M. Broy, W. Dosch, H. Partsch, P. Pepper, and M. Wirsing. Existential quantifiers in abstract data types. In H. A. Maurer, editor, *ICALP*, volume 71 of *LNCS*, pages 73–87. Springer, 1979.

3. R. Diaconescu. Graded consequence: an institution theoretic study. *Soft Comput.*, 18(7):1247–1267, 2014.
4. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1985.
5. J. L. Fiadeiro. *Categories for Software Engineering*. Springer, 2004.
6. J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *J. ACM*, 39(1):95–146, 1992.
7. J. A. Goguen and J. Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. In M. Nielsen and E. M. Schmidt, editors, *Automata, Languages and Programming, 9th Colloquium, Aarhus, Denmark, July 12-16, 1982, Proceedings*, volume 140 of *LNCS*, pages 265–281. Springer, 1982.
8. M. M. Hölzl, M. Meier, and M. Wirsing. Which soft constraints do you prefer? *Electr. Notes Theor. Comput. Sci.*, 238(3):189–205, 2009.
9. F. Orejas and L. Lambers. Lazy graph transformation. *Fundam. Inform.*, 118(1-2):65–96, 2012.
10. M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *IEEE Computer*, 40(11):38–45, 2007.
11. D. Sannella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2012.
12. M. Wirsing. Algebraic specification. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 675–788. MIT Press Cambridge, 1990.
13. M. Wirsing, G. Denker, C. L. Talcott, A. Poggio, and L. Briesemeister. A rewriting logic framework for soft constraints. *Electr. Notes Theor. Comput. Sci.*, 176(4):181–197, 2007.
14. M. Wirsing and M. M. Hölzl, editors. *Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing*, volume 6582 of *LNCS*. Springer, 2011.