Soft Concurrent Constraint Programming

Stefano Bistarelli[1], Ugo Montanari[2], and Francesca Rossi[3]

[1] Istituto per le Applicazioni Telematiche, C.N.R. Pisa,
Area della Ricerca di Pisa, Via G. Moruzzi 1, I-56124 Pisa, Italy.
Email: Stefano.Bistarelli@iat.cnr.it
[2] Dipartimento di Informatica, UniversitaÁdi Pisa,
Corso Italia 40, I-56125 Pisa, Italy.
Email: ugo@di.unipi.it
[3] Dipartimento di Matematica Pura ed Applicata, UniversitaÁdi Padova
Via Belzoni 7, 35131 Padova, Italy.
Email: frossi@math.unipd.it

Abstract. Soft constraints extend classical constraints to represent multiple consistency levels, and thus provide a way to express preferences, fuzziness, and uncertainty. While there are many soft constraint solving algorithms, even distributed ones, by now there seems to be no concurrent programming framework where soft constraints can be handled. In this paper we show how the classical concurrent constraint (cc) programming framework can work with soft constraints, and we also propose an extension of cc languages which can use soft constraints to prune and direct the search for a solution. We believe that this new programming paradigm, called soft cc (scc), can be very useful in many web-related scenarios. In fact, the language level allows web agents to express their interaction and negotiation protocols, and also to post their requests in terms of preferences, and the underlying soft constraint solver can £nd an agreement among the agents even if their requests are incompatible.

1 Introduction

The concurrent constraint (cc) language [Sar93] is a very interesting computational framework able to merge together constraint solving and concurrency. The main idea is to choose a *constraint system* and use constraints to model communication and synchronization among concurrent agents.

Until now, constraints in cc were *crisp* in the sense that only a yes/not answer could be de£ned. Recently the classical idea of *crisp* constraint has been shown to be too weak to represent real problems and a big effort has been done toward the use of soft constraints [FW92,BFBW92,DFP93,Rut94,FL93,SFV95,BMR97,BFM$^+$99,BMR01,Bis01].

Many real-life situations are, in fact, easily described via constraints able to state the necessary requirements of the problems. However, usually such requirements are not hard, and could be more faithfully represented as preferences, which should preferably be satis£ed but not necessarily. In real life, we are often challenged with over-constrained problems, which do not have any solution, and this also leads to the use of preferences or in general of soft constraints rather than classical constraints.

Generally speaking, a soft constraint is just a classical constraint plus a way to associate, either to the entire constraint or to each assignment of its variables, a certain element, which is usually interpreted as a level of preference or importance. Such levels are usually ordered, and the order re¤ects the idea that some levels are better than others. Moreover, one has also to say, via suitable combination operators, how to obtain the level of preference of a global solution from the preferences in the constraints.

Many formalisms have been developed to describe one or more classes of soft constraints. For instance consider the Fuzzy CSPs [DFP93,Rut94], where the crisp constraints are extended with a level of preference represented by a real number between 0 and 1, or the probabilistic CSPs [FL93], where the probability to be in the real problem is assigned to each constraint. Some other examples are the Partial CSPs [FW92] or the hierarchical CSPs [BFBW92] where a preference is assigned to each constraint, in order to also solve overconstrained problems.

We think that many network-related problem could be represented and solved by using soft constraints. Moreover, the possibility to use a concurrent language on top of a soft constraint system, could lead to the birth of new protocols with an embedded constraint satisfaction and optimization framework.

In particular, the constraints could be related to a quantity to be minimized but they could also satisfy policy requirements given for performance or administrative reasons. This leads to change the idea of QoS in routing and to speak of *constraint-based* routing [AMA+99,Cla89,JS00,CF00]. Constraints are in fact able to represent in a declarative fashion the needs and the requirements of agents interacting over the web.

The features of soft constraints could also be useful in representing routing problems where an imprecise state information is given [CN98]. Moreover, since QoS is only a speci£c application of a more general notion of Service Level Agreement (SLA), many applications could be enhanced by using such a framework. As an example consider E-commerce: here we are always looking for establishing an agreement between a merchant, a client and possibly a bank. Also, all auction-based transactions need an agreement protocol. Moreover, also security protocol analysis have shown to be enhanced by using security levels instead of a simple notion of secure/insecure level [BB01]. All these considerations advocate for the need of a soft constraint framework where optimal answers are extracted.

In the paper, we use one of the frameworks able to deal with soft constraints [BMR95,BMR97]. The framework is based on a semiring structure that is equipped with the operations needed to combine the constraints present in the problem and to choose the best solutions. According to the choice of the semiring, this framework is able to model all the speci£c soft constraint notions mentioned above. We compare the semiring-based framework with constraint systems *"a la Saraswat"* and then we show how use it inside the cc framework.

The next step is the extension of the syntax and operational semantics of the language to deal with the semiring levels. Here, the main novelty with respect to cc is that tell and ask agents are equipped with a preference (or consistency) threshold which is used to prune the search.

2 Background

2.1 Concurrent Constraint Programming

The concurrent constraint (cc) programming paradigm [Sar93] concerns the behavior of a set of concurrent agents with a shared store, which is a conjunction of constraints. Each computation step possibly adds new constraints to the store. Thus information is monotonically added to the store until all agents have evolved. The £nal store is a re£nement of the initial one and it is the result of the computation. The concurrent agents do not communicate with each other, but only with the shared store, by either checking if it entails a given constraint (*ask* operation) or adding a new constraint to it (*tell* operation).

Constraint Systems. A constraint is a relation among a speci£ed set of variables. That is, a constraint gives some information on the set of possible values which these variables may assume. Such information is usually not complete since a constraint may be satis£ed by several assignments of values of the variables (in contrast to the situation that we have when we consider a valuation, which tells us the only possible assignment for a variable). Therefore it is natural to describe constraint systems as systems of *partial* information [Sar93].

The basic ingredients of a constraint system de£ned following the information systems idea are a set D of *primitive constraints* or *tokens*, each expressing some partial information, and an entailment relation $\vdash$ de£ned on $\wp(D) \times D$ (or its extension de£ned on $\wp(D) \times \wp(D)$)[1] satisfying:

1. $u \vdash P$ for all $P \in u$ (re¤exivity) and
2. if $u \vdash v$, and $v \vdash z$, then $u \vdash z$ (transitivity).

As an example of entailment relation consider D as the set of equations over the integers; then $\vdash$ includes the pair $\langle \{x = 3, x = y\}, y = 3\rangle$, which means that the constraint $y = 3$ is entailed by the constraints $x = 3$ and $x = y$. Given $X \in \wp(D)$, let $\overline{X}$ be the set X closed under entailment. Then, a constraint in an information system $\langle \wp(D), \vdash\rangle$ is simply an element of $\overline{\wp(D)}$ (that is, a set of tokens).

As it is well known, $\langle \overline{\wp(D)}, \subseteq\rangle$ is a complete algebraic lattice, the compactness of $\vdash$ gives us algebraicity of $\overline{\wp(D)}$, with least element $true = \{P \mid \emptyset \vdash P\}$, greatest element D (which we will mnemonically denote $false$), glbs (denoted by $\sqcap$) given by the closure of the intersection and lubs (denoted by $\sqcup$) given by the closure of the union. The lub of chains is, however, just the union of the members in the chain. We use a, b, c, d and e to stand for elements of $\overline{\wp(D)}$; $c \geq d$ means $c \vdash d$.

The hiding operator: Cylindric Algebras. In order to treat the hiding operator of the language, a general notion of existential quanti£er is introduced which is formalized in terms of cylindric algebras. This leads to the concept of *cylindric constraint system* over an in£nite set of variables V such that for each variable $x \in V$, $\exists_x : \overline{\wp(D)} \to \overline{\wp(D)}$ is an operation satisfying:

[1] The extension is s.t. $u \vdash v$ iff $u \vdash P$ for every $P \in v$.

- $u \vdash \exists_x u$
- $u \vdash v$ implies $(\exists_x u) \vdash (\exists_x v)$
- $\exists_x(u \sqcup \exists_x v) \approx (\exists_x u) \sqcup (\exists_x v)$,
- $\exists_x \exists_y u \approx \exists_y \exists_x u$

Procedure calls. In order to model parameter passing, *diagonal elements* are added to the primitive constraints. We assume that, for x, y ranging in V, $\overline{\wp(D)}$ contains a constraint d_{xy} which satis£es the following axioms.

- $d_{xx} = true$,
- if $z \neq x, y$ then $d_{xy} = \exists_z(d_{xz} \sqcup d_{zy})$,
- if $x \neq y$ then $d_{xy} \sqcup \exists_x(c \sqcup d_{xy}) \vdash c$.

Note that the in the previous de£nition we assume the cardinality of the domain for x, y and z greater than 1. Note also that, if $\vdash$ models the equality theory, then the elements d_{xy} can be thought of as the formulas $x = y$.

The language. The syntax of a cc program is show in Table 1: P is the class of programs, F is the class of sequences of procedure declarations (or clauses), A is the class of agents, c ranges over constraints, and x is a tuple of variables. Each procedure is de£ned (at most) once, thus nondeterminism is expressed via the $+$ combinator only. We also assume that, in $p(x) :: A, vars(A) \subseteq x$, where $vars(A)$ is the set of all variables occurring free in agent A. In a program $P = F.A$, A is the initial agent, to be executed in the context of the set of declarations F. This corresponds to the language considered in [Sar93], which allows only guarded nondeterminism. In order to better understand the

Table 1. cc syntax

$$
\begin{aligned}
P &::= F.A \\
F &::= p(x) :: A \mid F.F \\
A &::= success \mid fail \mid tell(c) \rightarrow A \mid E \mid A \| A \mid \exists_x A \mid p(x) \\
E &::= ask(c) \rightarrow A \mid E + E
\end{aligned}
$$

extension of the language that we will introduce later, let us remind here (at least) the meaning of the tell and ask agents. The other constructs are easily understandable.

- agent "$ask(c) \rightarrow A$" checks whether constraint c is entailed by the current store and then, if so, behaves like agent A. If c is inconsistent with the current store, it fails, and otherwise it suspends, until c is either entailed by the current store or is inconsistent with it;
- agent "$tell(c) \rightarrow A$" adds constraint c to the current store and then, if the resulting store is consistent, behaves like A, otherwise it fails.

A formal treatment of the operational semantics of cc programs can be found in [Sar93,BP91].

2.2 Soft Constraints

Several formalization of the concept of *soft constraints* are currently available. In the following, we refer to the one based on c-semirings [BMR97,Bis01], which can be shown to generalize and express many of the others.

A soft constraint may be seen as a constraint where each instantiations of its variables has an associated value from a partially ordered set which can be interpreted as a set of preference values. Combining constraints will then have to take into account such additional values, and thus the formalism has also to provide suitable operations for combination ($\times$) and comparison ($+$) of tuples of values and constraints. This is why this formalization is based on the concept of c-semiring, which is just a set plus two operations.

C-semirings. A semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that:

- A is a set and $\mathbf{0}, \mathbf{1} \in A$;
- $+$ is commutative, associative and $\mathbf{0}$ is its unit element;
- $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element.

A *c-semiring* is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that: $+$ is idempotent, $\mathbf{1}$ is its absorbing element and $\times$ is commutative. Let us consider the relation $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that (see [BMR97]):

- $\leq_S$ is a partial order;
- $+$ and $\times$ are monotone on $\leq_S$;
- $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum;
- $\langle A, \leq_S \rangle$ is a complete lattice and, for all $a, b \in A$, $a + b = lub(a, b)$.

Moreover, if $\times$ is idempotent, then: $+$ distribute over $\times$; $\langle A, \leq_S \rangle$ is a complete distributive lattice and $\times$ its glb. Informally, the relation $\leq_S$ gives us a way to compare semiring values and constraints. In fact, when we have $a \leq_S b$, we will say that *b is better than a*. In the following, when the semiring will be clear from the context, $a \leq_S b$ will be often indicated by $a \leq b$.

Problems. Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, a £nite set D (the domain of the variables) and an ordered set of variables V, a *constraint* is a pair $\langle def, con \rangle$ where $con \subseteq V$ and $def : D^{|con|} \rightarrow A$. Therefore, a constraint speci£es a set of variables (the ones in *con*), and assigns to each tuple of values of these variables an element of the semiring. Consider two constraints $c_1 = \langle def_1, con \rangle$ and $c_2 = \langle def_2, con \rangle$, with $|con| = k$. Then $c_1 \sqsubseteq_S c_2$ if for all k-tuples t, $def_1(t) \leq_S def_2(t)$. The relation $\sqsubseteq_S$ is a partial order.

A *soft constraint problem* is a pair $\langle C, con \rangle$ where $con \subseteq V$ and C is a set of constraints: *con* is the set of variables of interest for the constraint set C, which however may concern also variables not in *con*. Note that a classical CSP is a SCSP where the chosen c-semiring is: $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true \rangle$. Fuzzy CSPs [Sch92] can instead be modeled in the SCSP framework by choosing the c-semiring $S_{FCSP} = \langle [0,1], max, min, 0, 1 \rangle$. Many other "soft" CSPs (Probabilistic, weighted, ...)

can be modeled by using a suitable semiring structure ($S_{prob} = \langle [0,1], max, \times, 0, 1 \rangle$, $S_{weight} = \langle \mathcal{R}, min, +, 0, +\infty \rangle, \ldots$).

Figure 1 shows a fuzzy CSP. Variables are inside circles, constraints are represented by undirected (unary, binary or n-ary) arcs, and semiring values are written to the right of the corresponding tuples. The variables of interest (that is the set *con*) are represented with a double circle. Here we assume that the domain D of the variables contains only elements a and b.

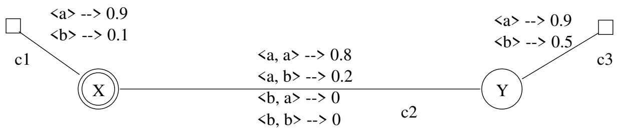


Fig. 1. A fuzzy CSP

Combining and projecting soft constraints. Given two constraints $c_1 = \langle def_1, con_1 \rangle$ and $c_2 = \langle def_2, con_2 \rangle$, their *combination* $c_1 \otimes c_2$ is the constraint $\langle def, con \rangle$ de£ned by $con = con_1 \cup con_2$ and $def(t) = def_1(t \downarrow^{con}_{con_1}) \times def_2(t \downarrow^{con}_{con_2})$, where $t \Downarrow^X_Y$ denotes the tuple of values over the variables in Y, obtained by projecting tuple t from X to Y. In words, combining two constraints means building a new constraint involving all the variables of the original ones, and which associates to each tuple of domain values for such variables a semiring element which is obtained by multiplying the elements associated by the original constraints to the appropriate subtuples.

Given a constraint $c = \langle def, con \rangle$ and a subset I of V, the *projection* of c over I, written $c \Downarrow_I$ is the constraint $\langle def', con' \rangle$ where $con' = con \cap I$ and $def'(t') = \sum_{t / t \downarrow^{con}_{I \cap con} = t'} def(t)$. Informally, projecting means eliminating some variables. This is done by associating to each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables. In short, combination is performed via the multiplicative operation of the semiring, and projection via the additive one.

Solutions. The *solution* of an SCSP problem $P = \langle C, con \rangle$ is the constraint $Sol(P) = (\bigotimes C) \Downarrow_{con}$. That is, we combine all constraints, and then project over the variables in *con*. In this way we get the constraint over *con* which is "induced" by the entire SCSP.

For example, the solution of the fuzzy CSP of Figure 1 associates a semiring element to every domain value of variable x. Such an element is obtained by £rst combining all the constraints together. For instance, for the tuple $\langle a, a \rangle$ (that is, $x = y = a$), we have to compute the minimum between 0.9 (which is the value assigned to $x = a$ in constraint c_1), 0.8 (which is the value assigned to $\langle x = a, y = a \rangle$ in c_2) and 0.9 (which is the value for $y = a$ in c_3). Hence, the resulting value for this tuple is 0.8. We can do the same work for tuple $\langle a, b \rangle \to 0.2$, $\langle b, a \rangle \to 0$ and $\langle b, b \rangle \to 0$. The obtained tuples are then projected over variable x, obtaining the solution $\langle a \rangle \to 0.8$ and $\langle b \rangle \to 0$.

Sometimes it may be useful to £nd only a semiring value representing the least upper bound among the values yielded by the solutions. This is called the *best level* *of consistency* of an SCSP problem P and it is de£ned by $blevel(P) = Sol(P) \Downarrow_{\emptyset}$ (for instance, the fuzzy CSP of Figure 1 has best level of consistency 0.8). We also say that: P is α-consistent if $blevel(P) = \alpha$; P is consistent iff there exists $\alpha > \mathbf{0}$ such that P is α-consistent; P is inconsistent if it is not consistent.

3 Concurrent Constraint Programming over Soft Constraints

Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and an ordered set of variables V over a £nite domain D, we will now show how soft constraints with a suitable pair of operators form a semiring, and then, we evidentiate the properties needed to map soft constraints over constraint systems *"a la Saraswat"*.

We start by giving the de£nition of the carrier set of the semiring.

De£nition 1 (functional constraints). *We de£ne* $\mathcal{C} = (V \to D) \to A$ *as the set of all possible constraints that can be built starting from S, D and V.*

A generic function describing the assignment of domain elements to variables will be denoted in the following by $\eta : V \to D$. Thus a constraint is a function which, given an assignment η of the variables, returns a value of the semiring.

Note that in this *functional* formulation, each constraint is a function and not a pair representing the variable involved and its de£nition. Such a function involves all the variables in V, but it depends on the assignment of only a £nite subset of them. We call this subset the *support* of the constraint. For computational reasons we require each support to be £nite.

De£nition 2 (constraint support). *Consider a constraint* $c \in \mathcal{C}$. *We de£ne his support as* $supp(c) = \{v \in V \mid \exists \eta, d_1, d_2 . c\eta[v \rightsquigarrow d_1] \neq c\eta[v \rightsquigarrow d_2]\}$, *where*

$$\eta[v \rightsquigarrow d]v' = \begin{cases} d & \textit{if } v = v', \\ \eta v' & \textit{otherwise}. \end{cases}$$

Note that $c\eta[v \rightsquigarrow d_1]$ means $c\eta'$ where η' is η modi£ed with the association $v \rightsquigarrow d_1$ (that is the operator [ ] has precedence over application).

De£nition 3 (functional mapping). *Given any soft constraint* $\langle def, \{v_1, \ldots, v_n\} \rangle \in C$, *we can de£ne its corresponding function* $c \in \mathcal{C}$ *as* $c\eta[v_1 \rightsquigarrow d_1] \ldots [v_n \rightsquigarrow d_n] = def(d_1, \ldots, d_n)$. *Clearly* $supp(c) \subseteq \{v_1, \ldots, v_n\}$.

De£nition 4 (Combination and Sum). *Given the set* $\mathcal{C}$, *we can de£ne the combination and sum functions* $\otimes, \oplus : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ *as follows:*

$$(c_1 \otimes c_2)\eta = c_1\eta \times_S c_2\eta$$
$$(c_1 \oplus c_2)\eta = c_1\eta +_S c_2\eta.$$

Notice that function $\otimes$ has the same meaning of the already de£ned $\otimes$ operator (see Section 2.2) while function $\oplus$ models a sort of disjunction (similar to that introduced in [BCGR00].

By using the $\oplus_S$ operator we can easily extend the partial order $\leq_S$ over $\mathcal{C}$ by de£ning $c_1 \sqsubseteq c_2 \iff c_1 \oplus_S c_2 = c_2$.

We can also de£ne a unary operator that will be useful to represent the unit elements of the two operations $\oplus$ and $\otimes$. To do that, we need the de£nition of constant functions over a given set of variables.

De£nition 5 (constant function). *We de£ne function $\bar{a}$ as the function that returns the semiring value a for all assignments η, that is, $\bar{a}\eta = a$. We will usually write $\bar{a}$ simply as a.*

It is easy to verify that each constant has an empty support. An example of constants that will be useful later are $\bar{\mathbf{0}}$ and $\bar{\mathbf{1}}$ that represent respectively the constraint associating $\mathbf{0}$ and $\mathbf{1}$ to all the assignment of domain values.

Theorem 1 (Higher order semiring). *The structure $S_C = \langle \mathcal{C}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ where*

- *$\mathcal{C} : (V \to D) \to A$ is the set of all the possible constraints that can be built starting from S, D and V as de£ned in De£nition 1,*
- *$\otimes$ and $\oplus$ are the functions de£ned in De£nition 4, and*
- *$\mathbf{0}$ and $\mathbf{1}$ are constant functions de£ned following De£nition 5,*

is a c-semiring.

The next step is to look for a notion of token and of entailment relation. We de£ne as tokens the functional constraints in $\mathcal{C}$ and we introduce a relation $\vdash$ that is an entailment relation when the multiplicative operator of the semiring is idempotent.

De£nition 6 ($\vdash$ relation). *Consider the high order semiring carrier set $\mathcal{C}$ and the partial order $\sqsubseteq$. We de£ne the relation $\vdash \subseteq \wp(\mathcal{C}) \times \mathcal{C}$ s.t. for each $C \in \wp(\mathcal{C})$ and $c \in \mathcal{C}$, we have $C \vdash c \iff \bigotimes C \sqsubseteq c$.*

The next theorem shows that when the multiplicative operator of the semiring is idempotent, the $\vdash$ relation satis£es all the properties needed by an entailment.

Theorem 2 ($\vdash$ with idempotent $\times$ is an entailment relation). *Consider the higher order semiring carrier set $\mathcal{C}$ and the partial order $\sqsubseteq$. Consider also the relation $\vdash$ of De£nition 6. Then, if the multiplicative operation of the semiring is idempotent, $\vdash$ is an entailment relation.*

Note that in this setting the notion of token (constraint) and of set of tokens (set of constraints) closed under entailment is used indifferently. In fact, given a set of constraint functions C_1, its closure w.r.t. the entailment is a set $\bar{C_1}$ that contains all the constraints greater than $\bigotimes C_1$. This set is univocally representable by the constraint function $\bigotimes C_1$.

The de£nition of the entailment operator $\vdash$ on top of the higher order semiring $S_C = \langle \mathcal{C}, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ and of the $\sqsubseteq$ relation leads to the notion of *soft constraint system*. It is also important to notice that in [Sar93,dBGMP97] they claim the constraint system to be a *complete algebraic* lattice. Here we do not ask for this algebricity since the algebricity of the structure $\mathcal{C}$ strictly depends on the properties of the semiring.

Non-idempotent $\times$. If the constraint system is defined on top of a non-idempotent multiplicative operator, we cannot obtain a $\vdash$ relation satisfying all the properties of an entailment. Nevertheless, we can give a *denotational* semantics to the constraint store, as described in Section 4, using the operations of the higher order semiring.

To treat the hiding operator of the language, a general notion of existential quantifier has to be introduced by using notions similar to those used in cylindric algebras. Note however that cylindric algebras are first of all boolean algebras. This could be possible in our framework only when the $\times$ operator is idempotent.

Definition 7 (hiding). *Consider a set of variables V with domain D and the corresponding soft constraint system $\mathcal{C}$. We define for each $x \in V$ the hiding function*

$$(\exists_x c)\eta = \sum_{d_i \in D} c\eta[x \rightsquigarrow d_i].$$

Notice that x does not belong to the support of $\exists_x c$.

To model parameter passing we need instead to define what diagonal elements are.

Definition 8 (diagonal elements). *Consider an ordered set of variables V and the corresponding soft constraint system $\mathcal{C}$. Let us define for each $x, y \in V$ a constraint $d_{xy} \in \mathcal{C}$ s.t., $d_{xy}\eta[x \rightsquigarrow a, y \rightsquigarrow b] = \mathbf{1}$ if $a = b$ and $d_{xy}\eta[x \rightsquigarrow a, y \rightsquigarrow b] = \mathbf{0}$ if $a \neq b$. Notice that $supp(d_{xy}) = \{x, y\}$.*

4 Soft Concurrent Constraint Programming

The next step in our work is now to extend the syntax of the language in order to directly handle the cut level. This means that the syntax and semantics of the tell and ask agents have to be enriched with a threshold to specify when ask/tell agents have to fail, succeed or suspend.

Given a soft constraint system $\langle S, D, V \rangle$ and the corresponding structure $\mathcal{C}$, the syntax of agents in soft concurrent constraint *scc* programming is given in Table 2. We present here a structural operational semantics for scc programs, in the SOS style, which consists of defining the semantic of the programming language by specifying a set of *configurations* Γ, which define the states during execution, a relation $\rightarrow \subseteq \Gamma \times \Gamma$ which describes the *transition* relation between the configurations, and a set T of *terminal* configurations.

Table 2. scc syntax

$$\begin{aligned}
P &::= F.A \\
F &::= p(X) :: A \mid F.F \\
A &::= stop \mid tell(c) \rightarrow^a A \mid E \mid A \| A \mid \exists X.A \mid p(X) \\
E &::= ask(c) \rightarrow^a A \mid E + E
\end{aligned}$$

The set of con£gurations represent the evolutions of the agents and the modi£cations in the constraint store.

De£nition 9 (con£gurations). *The set of con£gurations for a soft cc system is the set* $\Gamma = \{\langle A, \sigma\rangle\} \cup \{\langle success, \sigma\rangle\}$ *where* $\sigma \in \mathcal{C}$. *The set of terminal con£gurations is the set* $T = \{\langle success, \sigma\rangle\}$ *and the transition rule for the scc language are de£ned in Table 3.*

Table 3. Transition rules for scc

$$\langle stop, \sigma\rangle \longrightarrow \langle success, \sigma\rangle \qquad \text{(Stop)}$$

$$\frac{(\sigma \otimes c) \Downarrow_{\emptyset} \not< a}{\langle tell(c) \rightarrow^{a} A, \sigma\rangle \longrightarrow \langle A, \sigma \otimes c\rangle} \qquad \text{(Valued-tell)}$$

$$\frac{\sigma \vdash c, \sigma \Downarrow_{\emptyset} \not< a}{\langle ask(c) \rightarrow^{a} A, \sigma\rangle \longrightarrow \langle A, \sigma\rangle} \qquad \text{(Valued-ask)}$$

$$\frac{\langle A_1, \sigma\rangle \longrightarrow \langle A'_1, \sigma'\rangle}{\begin{array}{c}\langle A_1 \| A_2, \sigma\rangle \longrightarrow \langle A'_1 \| A_2, \sigma'\rangle \\ \langle A_2 \| A_1, \sigma\rangle \longrightarrow \langle A_2 \| A'_1, \sigma'\rangle\end{array}} \quad \frac{\langle A_1, \sigma\rangle \longrightarrow \langle success, \sigma'\rangle}{\begin{array}{c}\langle A_1 \| A_2, \sigma\rangle \longrightarrow \langle A_2, \sigma'\rangle \\ \langle A_2 \| A_1, \sigma\rangle \longrightarrow \langle A_2, \sigma'\rangle\end{array}} \qquad \text{(Parallelism)}$$

$$\frac{\langle E_1, \sigma\rangle \longrightarrow \langle A_1, \sigma'\rangle}{\begin{array}{c}\langle E_1 + E_2, \sigma\rangle \longrightarrow \langle A_1, \sigma'\rangle \\ \langle E_2 + E_1, \sigma\rangle \longrightarrow \langle A_1, \sigma'\rangle\end{array}} \qquad \text{(Nondeterminism)}$$

$$\frac{\langle A[y/x], \sigma\rangle \longrightarrow \langle A', \sigma'\rangle}{\langle \exists_x A, \sigma\rangle \longrightarrow \langle A', \sigma'\rangle} \text{ with } y \textit{ fresh} \qquad \text{(Hidden variables)}$$

$$\langle p(y), \sigma\rangle \longrightarrow \langle A[y/x], \sigma\rangle \text{ when } p(x) :: A \qquad \text{(Procedure call)}$$

Here is a brief description of the most complex rules:

Valued-tell The valued-tell rule checks for the α-consistency of the SCSP de£ned by the store $\sigma \cup c$. The rule can be applied only if the store $\sigma \cup c$ is b-consistent with $b \not< a$. In this case the agent evolves to the new agent A over the store $\sigma \otimes c$.

Valued-ask The semantics of the valued-ask is extended in a way similar to what we have done for the valued-tell action. This means that to apply the rule we need to check if the store σ entails the constraint c and also if the store is "consistent enough" w.r.t. the threshold a set by the programmer.

Nondeterminism and parallelism The composition operators $+$ and $\|$ are not modi£ed w.r.t. the classical ones: a parallel agent will succeed if all the agents succeeds; a nondeterministic rule chooses any agent whose guard succeeds.

Hidden variables The semantics of the existential quanti£er is similar to that described in [Sar93] by using the notion of *freshness* of the new variable added to the store.

Observables. Given the transition system as de£ned in the previous section, we now de£ne what we want to observe of the program behaviours described by the transitions. To do this we de£ne for each agent A the set of constraints

$$\mathcal{S}_A = \{\sigma \Downarrow_{var(A)} \mid \langle A, \mathbf{1} \rangle \rightarrow^* \langle success, \sigma \rangle\}$$

that collects the results of the successful computations that the agent can perform. The computed store σ is projected over the variables of the agent A to discard any *fresh* variable introduced in the store by the $\exists$ operator.

The observable $\mathcal{S}_A$ could be re£ned by considering, instead of the set of successful computations starting from $\langle A, \mathbf{1} \rangle$, only a subset of them. One could be interested in considering for example only the *best* computations: in this case, all the computations leading to a store worse than one already collected are disregarded. With a pessimistic view, the representative subset could instead collect all the worst computations (that is, all the computations better than others are disregarded). Finally, also a set containing both the best and the worst computations could be considered. These options are reminiscent of Hoare, Smith and Egli-Milner powerdomains respectively.

Let us also notice that different cut levels in the ask and tell operations could lead to a different £nal sets $\mathcal{S}_A$. In fact, it can be proved that if the thresholds of the ask and tell operations of the program are not worse than a given α, we can be sure to £nd in the £nal store only solutions not worse than α. This observation can be useful when we are looking just for the best stores reachable from an initial given agent. In fact, we can move the cut up and down (in a way similar to a binary search) and perform a branch and bound exploration of the search tree in order to £nd the £nal success sets.

In this paper we only consider a semantics that collects success states. We plan to extend the operational semantics to collect also failing and hunging computations.

5 A simple example

In the following we will show the behaviour of some of the rules of our transition system. We consider in this example a soft constraint system over the fuzzy semiring. Consider the constraints

$$c(x,y) = \frac{1}{1+|x-y|} \quad \text{and } c'(x) = \begin{cases} 1 & \text{if } x \leq 10, \\ 0 & \text{otherwise.} \end{cases}$$

Let's now evaluate the following agent in the empty starting store 1:

$$\langle tell(c) \rightarrow^{0.4} ask(c') \rightarrow^{0.8} stop, 1 \rangle.$$

By applying the *Valued-tell* rule we need to check $(1 \otimes c) \Downarrow_{\emptyset} \not< 0.4$. Since $\mathbf{1} \otimes c = c$ and $c \Downarrow_{\emptyset} = 1$, the agent can perform the step, and it reaches the state $\langle ask(c') \rightarrow^{0.8} stop, c \rangle$. Now we need to check (by following the rule of *Valued-ask*) if $c \vdash c'$ and $c \Downarrow_{\emptyset} \not< 0.8$. While the second relation easily holds, the £rst one does not hold (in fact, for $x = 11$ and $y = 10$ we have $c'(x) = 0$ and $c(x,y) = 0.5$). If instead we consider the constraint

$$c"(x,y) = \frac{1}{1+2 \times |x-y|}$$

in place of c', then the condition $c \vdash c$" easily holds and the agent $ask(c") \rightarrow^{0.8} stop$ can perform its last step, reaching the $stop$ and $success$ states: $\langle stop, c \otimes c" \rangle \rightarrow \langle success, c \otimes c" \rangle$.

6 A Possible Application

We consider in this section a network problem, involving a set of processes running on distinct locations and sharing some variables, over which they need to synchronize.

Each process is connected to a set of variables, shared with other processes, and it can perform several moves. Each of such moves involves performing an action over some or all the variables connected to the process. An action over a variable consists of giving a certain value to that variable. A special value "idle" models the fact that a process does not perfom any action over a variable. Each process has also the possibility of not moving at all: in this case, all its variables are given the idle value.

The desired behavior of a network of such processes is that, at each move of the entire network:

1. processes sharing a variable perform the same action over it;
2. as few processes as possible remain idle.

To describe a network of processes with these features, we use an SCSP where each variable models a shared variable, and each constraint models a process and connects the variables corresponding to the shared variables of that process. The domain of each variable in this SCSP is the set of all possible actions, including the idle one. Each way of satisfying a constraint is therefore a tuple of actions that a process can perform on the corresponding shared variables.

In this scenario, softness can be introduces both in the domains and in the constraints. In particular, since we prefer to have as many moving processes as possible, we can associate a penalty to both the idle element in the domains, and to tuples containing the idle action in the constraints. As for the other domain elements and constraint tuples, we can assign them suitable preference values to model how much we like that action or that process move.

For example, we can use the semiring $S = \langle [-\infty, 0], max, +, -\infty, 0 \rangle$, where 0 is the best preference level (or, said dually, the weakest penalty), $-\infty$ is the worst level, and preferences (or penalties) are combined by summing them. According to this semiring, we can assign value $-\infty$ to the idle action or move, and suitable other preference levels to the other values and moves. Figure 2 gives the details of a part of a network and it shows eigth processes (that is, $c_1, \ldots, c_8$) sharing a total of six variables. In this example, we assume that processes c_1, c_2 and c_3 are located on site a, processes c_5 and c_6 are located on site b, and c_4 is located on site c. Processes c_7 and c_8 are located on site d. Site e connects this part of the network to the rest. Therefore, for example, variables x_d, y_d and z_d are shared between processes located in distinct locations.

As desired, £nding the best solution for the SCSP representing the current state of the process network means £nding a move for all the processes such that they perform the same action on the shared variables and there is a minimum number of idle processes. However, since the problem is inherently distributed, it does not make sense,

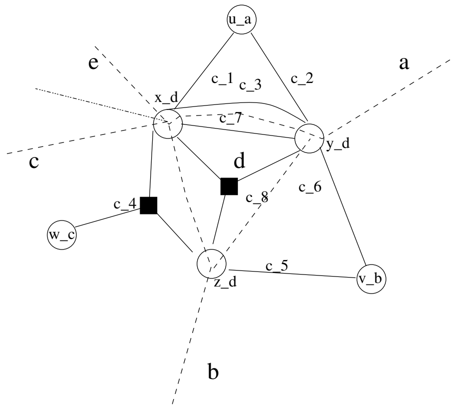


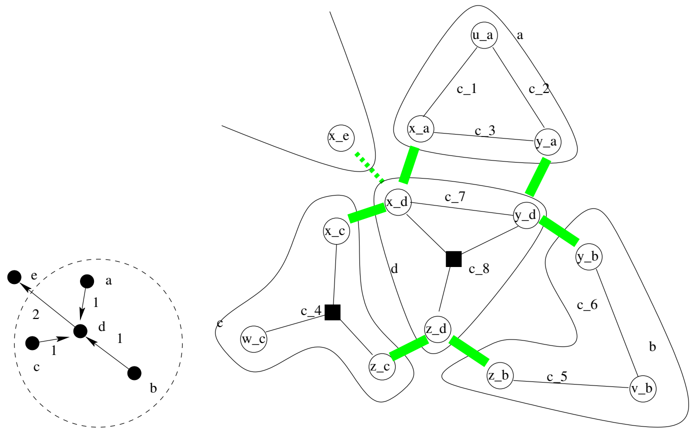
Fig. 2. The SCSP describing part of a process network.

and it might not even be possible, to centralize all the information and give it to a single soft constraint solver.

On the contrary, it may be more reasonable to use several soft constraint solvers, one for each network location, which will take care of handling only the constraints present in that location. Then, the interaction between processes in different locations, and the necessary agreement to solve the entire problem, will be modelled via the scc framework, where each agent will represent the behaviour of the processes in one location.

More precisely, each scc agent (and underlying soft constraint solver) will be in charge of receiving the necessary information from the other agents (via suitable asks) and using it to achieve the synchronization of the processes in its location. For this protocol to work, that is, for obtaining a global optimal solution without a centralization of the work, the SCSP describing the network of processes has to have a tree-like shape, where each node of the tree contains all the processes in a location, and the agents have to communicate from the bottom of the tree to its root [BMR97]. In our example, the tree structure we will use is the one shown in Figure 3(a), which also shows the direction of the child-parent relation links (via arrows). Figure 3(b) describes instead the partition of the SCSP over the four involved locations. The gray connections represent the synchronization to be assured between distinct locations. Notice that, w.r.t. Figure 2, we have duplicated the variables representing variables shared between distinct locations, because of our desire to £rst perform a local work and then to communicate the results to the other locations.

The scc agents (one for each location plus the parallel composition of all of them) are therefore de£ned as follows:



(a) A possible tree structure for our network.

(b) The SCSP partitioned over the four locations.

Fig. 3. The ordered process network.

$$
\begin{aligned}
&A_a : \exists_{u_a}(tell(c_1(x_a,u_a) \wedge c_2(u_a,y_a) \wedge c_3(x_a,y_a)) \rightarrow tell(end_a = true) \rightarrow stop)\\
&A_b : \exists_{v_b}(tell(c_5(y_b,v_b) \wedge c_6(z_b,v_b)) \rightarrow tell(end_b = true) \rightarrow stop)\\
&A_c : \exists_{w_c}(tell(c_4(x_c,w_c,z_c)) \rightarrow tell(end_c = true) \rightarrow stop)\\
&A_d : ask(end_a = true \wedge end_b = true \wedge end_c = true \wedge end_d = true) \rightarrow\\
&\quad tell(c_7(x_d,y_d) \wedge c_8(x_d,y_d,z_d) \wedge x_a = x_d = x_c \wedge y_a = y_d = y_b \wedge z_b = z_d = z_c)\\
&\quad \rightarrow tell(end_d = true) \rightarrow stop\\
&A : A_a \mid A_b \mid A_c \mid A_d
\end{aligned}
$$

Agents A_a,A_b,A_c and A_d represent the processes running respectively in the location a, b, c and d. Note that, at each ask or tell, the underlying soft constraint solver will only check (for consistency or entailment) a part of the current set of constraints: those local to one location. Due to the tree structure chosen for this example, where agents A_a, A_b, and A_c correspond to leaf locations, only agent A_d shows all the actions of a generic process: £rst it needs to collect the results computed separately by the other agents (via the ask); then it performs its own constraint solving (via a tell), and £nally it can set its end ¤ag, that will be used by a parent agent (in this case the agent corresponding to location e, which we have not modelled here).

7 Conclusions and Future Work

We have shown that cc languages can deal with soft constraints. Moreover, we have extended their syntax to use soft constraints also to direct and prune the search process at the language level. We believe that such a new programming paradigm could be very useful for web and internet programming.

In fact, in several network-related areas, constraints are already been used [BB01,AMA+99,Cla89,JS00,CF00]. The soft constraint framework has the advantage over the classical one of selecting a "best" solution also in overconstrained or underconstrained systems. Moreover, the need to express preferences and to search for optimal solutions shows that soft constraints can improve the modelling of web interaction scenarios.

References

[AMA+99] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Rfc2702: Requirements for traf£c engineering over mpls. Technical report, Network Working Group, September 1999.

[BB01] G. Bella and S. Bistarelli. Sof constraints for security protocol analysis: Con£dentiality. In I.V. Ramakrishnan, editor, *Proc. of PADL 2001, 3rd international symposium on Practical Aspects of Declarative Languages*, volume 1990 of *LNCS*, pages 108–122. Springer-Verlag, mar 2001.

[BCGR00] S. Bistarelli, P. Codognet, Y. Georget, and F. Rossi. Labeling and partial local consistency for soft constraint programming. In *Proc. of the 2nd International Workshop on Practical Aspects of Declarative Languages (PADL'00)*, volume 1753 of *LNCS*. Springer-Verlag, 2000.

[BFBW92] A. Borning, B. Freeman-Benson, and M. Wilson. Constraint hierachies. *Lisp and Symbolic Computation*, 5(3):223–270, 1992.

[BFM+99] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and Valued CSPs: Frameworks, properties, and comparison. *CONSTRAINTS: An international journal. Kluwer*, 4(3), 1999.

[Bis01] S. Bistarelli. *Soft Constraint Solving and programming: a general framework.* PhD thesis, Dipartimento di Informatica, UniversitaÁdi Pisa, Italy, mar 2001. TD-2/01.

[BMR95] S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95*, San Francisco, CA, USA, 1995. Morgan Kaufman.

[BMR97] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, Mar 1997.

[BMR01] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Logic Programming: Syntax and Semantics. *ACM Transactions on Programming Languages and System (TOPLAS)*, 2001. To Appear.

[BP91] F.S. De Boer and C. Palamidessi. A fully abstract model for concurrent constraint programming. In S. Abramsky and T.S.E. Maibaum, editors, *Proc. TAPSOFT/CAAP*, volume 493. Springer-Verlag, 1991.

[CF00] M. Calisti and B. Faltings. Distributed constrained agents for allocating service demands in multi-provider networks,. *Journal of the Italian Operational Research Society*, XXIX(91), 2000. Special Issue on Constraint-Based Problem Solving.

[Cla89] D. Clark. Rfc1102: Policy routing in internet protocols. Technical report, Network Working Group, May 1989.

[CN98] S. Chen and K. Nahrstedt. Distributed qos routing with imprecise state information. In *Proc. International Conference on Computer, Communications and Networks (ICCCN'98)*, oct 1998.

[dBGMP97] F. S. de Boer, M. Gabbrielli, E. Marchiori, and C. Palamidessi. Proving concurrent constraint programs correct. *TOPLAS*, 19(5):685–725, 1997.

[DFP93] D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for ¤exible constraint satisfaction. In *Proc. IEEE International Conference on Fuzzy Systems*, pages 1131–1136. IEEE, 1993.

[FL93] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proc. European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU)*, volume 747 of *LNCS*, pages 97–104. Springer-Verlag, 1993.

[FW92] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *AI Journal*, 58, 1992.

[JS00] R. Jain and W. Sun. QoS/Policy/Constraint-based routing. In *Carrier IP Telephony 2000 Comprehensive Report*. International Engineering Consortium, aug 2000. ISBN: 0-933217-75-7.

[Rut94] Zs. Ruttkay. Fuzzy constraint satisfaction. In *Proc. 3rd IEEE International Conference on Fuzzy Systems*, pages 1263–1268, 1994.

[Sar93] V.A. Saraswat. *Concurrent Constraint Programming*. MIT Press, 1993.

[Sch92] T. Schiex. Possibilistic constraint satisfaction problems, or "how to handle soft constraints?". In *Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.

[SFV95] T. Schiex, H. Fargier, and G. Verfaille. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. IJCAI95*, pages 631–637, San Francisco, CA, USA, 1995. Morgan Kaufmann.