Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 176 (2007) 181–197
www.elsevier.com/locate/entcs

A Rewriting Logic Framework for Soft Constraints

Martin Wirsing[1,2]

Ludwig-Maximilians-Universität München, Institut für Informatik, 80538 München

Grit Denker, Carolyn Talcott, Andy Poggio, Linda Briesemeister[3]

SRI International, 333 Ravenswood Ave, Menlo Park, California 94025

Abstract

Soft constraints extend classical constraints to deal with non-functional requirements, over-constrained problems and preferences. Bistarelli, Montanari and Rossi have developed a very elegant and abstract semiring based theory of soft constraints where many different kinds of soft constraints can be represented and combined in a uniform way over so-called constraint semirings. In this paper we present a framework for prototyping of soft constraints à la Bistarelli, Montanari and Rossi in Rewriting Logic. As a case study we present an application of soft constraints to the new area of software-defined radio networks. We model the problem of "optimal" parameter assignments for software-defined radios as a soft constraint solving problem, prove the correctness of the constraint solving algorithm, implement the solution in our prototypical Rewriting Logic framework for soft constraints, and embed our soft constraint solver in SRI's Policy-Aware, Goal-Oriented Distributed Architecture (PAGODA) for modelling radio networks.

Keywords: Rewriting logic, soft constraint, software-defined radio.

1 Introduction

Soft constraints are an extension of classical constraints to deal with non-functional requirements, over-constrained problems and preferences. Instead of determining just a subset of admissible domain elements, a soft constraint assigns a grade - to be chosen from a set of finite or infinitely many "preference" values - to each element of the application domain. Bistarelli, Montanari and Rossi [4][1] have developed a very elegant and abstract semiring based theory of soft constraints where many

[1] This work has been partially sponsored by the project SENSORIA, IST-2005-016004.
[2] wirsing@lmu.de
[3] firstname.lastname@sri.com

doi:10.1016/j.entcs.2007.06.015

different kinds of soft constraints can be represented and combined in a uniform way over so-called constraint semirings.

In this paper, we present a framework for prototyping of soft constraints à la Bistarelli, Montanari and Rossi in Rewriting Logic [7]. To our knowledge this is the first rewriting realisation of soft constraints.

Other implementations are based on constraint logic programming and concurrent constraint programming: $clp(FD, S)$ [13] and $softclp(FD)$ [17] are extensions of $clp(FD)$ by constraint semirings. The former is based a new abstract machine resulting in a good efficiency; the latter is an extension of the $clp(FD)$ library and can in this way reuse a broad class of constraint propagation algorithms and search methods from any parent $clp(FD)$ solver. The approach of [3] uses Frühwirth's Constraint Handling Rules for soft constraint propagation, whereas [9] extend the concurrent constraint language Mozart by soft constraints.

Our rewriting logic framework consists of a package of Maude theories and parameterized functional modules which can be integrated easily in any other Maude application by instantiating the parameter theories with the particular settings of the application. The axiomatic theory of constraint semi-rings is modelled as Maude functional theory; special constraint semi-rings such as weighted sum and fuzzy natural numbers are modelled as functional Maude modules; all other modules of the framework (such as cartesian products of constraint semirings, implicit and explicit soft constraints as well as the constraint solving algorithms) are parameterized by the choice of the constraint domain and semiring. In order to improve the efficiency of constraint solving, all recursive specifications are written in tail-recursive form and domain elements of individual constraints are ordered according to their grade in the constraint semiring. We also prove the correctness of our search algorithm.

As a case study, we present an application of soft constraints to the new area of software-defined radio networks (see e.g. http://www.sdrforum.org). A software-defined radio is a radio, in which most or all frequency control, modulation/demodulation formats, bandwidth, and other parameters are realized by software and thus can be changed during radio operation. This offers a tremendous new flexibility to commercial and amateur radios but controlling such radios in different environments requires subtle fine-tuning and adaptation of their parameter settings. We model the problem of "optimal" parameter assignments for software-defined radios as a soft constraint solving problem using a cartesian product of constraint semirings, implement the solution in our prototypical Rewriting Logic framework for soft constraints, and embed our soft constraint solver in SRI's Policy-Aware, Goal-Oriented Distributed Architecture (PAGODA) for modelling radio networks. Note that although other implementations such as [3] and [9] can also cope with cartesian products of semirings, none of the other published case studies has exploited this feature. Test runs show that most optimal parameter assignments for software-defined radio can be computed in a few seconds; only the most difficult constraint sets need more than one minute for the construction of the solutions.

2 Example Application

As the driving application for developing constraint solving mechanisms in Maude, we focus on communicating wireless devices or radios that optimize resource usage such as bandwidth and power with respect to given goals. This application is set in the broader context of policy and goal-based applications, in which devices cooperatively strive to achieve goals while satisfying policies that constrain possible solutions.

PAGODA (see http://pagoda.csl.sri.com) is a modular architecture for design of (partially) autonomous systems. A PAGODA node interacts with its environment by sensing and affecting, driven by goals to achieve and constrained by policies. A PAGODA system is a collection of PAGODA nodes cooperating to achieve some mutual goal. The PAGODA architecture was inspired by the study of architectures developed for autonomous space systems, especially the MDS architecture [12] and its precursors [15].

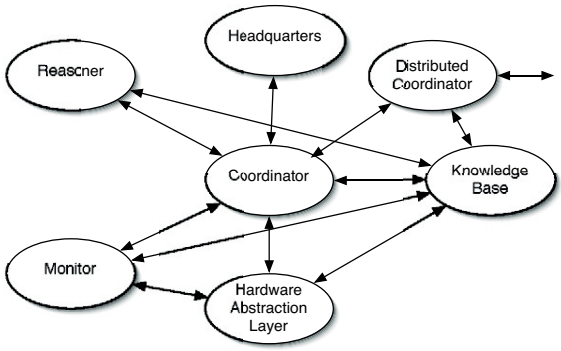

Fig. 1. PAGODA node architecture

Figure 1 shows the principal components of a PAGODA node: a knowledge base, a coordinator, a reasoner, a monitor, and a hardware abstraction layer. There is also a component for communication with a system operator (headquarters), and a component responsible for distributed coordination with other PAGODA nodes (distributed coordinator).

The knowledge base contains knowledge that is shared and updated by the remaining components. This knowledge covers a wide range of information including (1) goals a node or system is trying to achieve, (2) policies that constrain the actions or interactions a node or system is allowed to do, thus reducing the number of choices for setting parameters, and (3) a device model that specifies the parameters that can be set (knobs) and read (sensors) and their relationships.

The effects of knob settings can be observed in terms of improved connectivity, higher bandwidth and other observable results that are related to goals. For example, increasing the transmission power or choosing lower frequencies results in better connectivity between the radios. These relationships are defined in tables such as the one given in Figure 2 (left). Note that a certain knob setting, such as high frequencies for transmission, can have contradicting effects (decreased connec-

Effect Category:	improved connectivity	high bandwidth	...
Knob:			
TransPwr (high)	+		
" (low)	-		
TransFreq (high)	-	+	
" (low)	+	-	
Compression (high)		+	
" (low)		-	
ECC (high)		-	
" (low)		+	
...			

Sensor:	SignalStr	ConnLoss	Throughput	...
Knob:				
TransPwr (high)	50	50	50	
" (low)				
TransFreq (high)			30	
" (low)	50	15		
Compression (high)			20	
" (low)				
ECC (high)		10	25	
" (low)				
...				

Fig. 2. Partial knob-effect (left) and partial knob-sensor (right) tables

tivity but higher bandwidth). Similarly, the dependencies between knob settings and measurable sensor reading are also defined. Figure 2 (right) shows an excerpt defining the impact of choosing of a certain knob setting on a sensor reading. By definition, if the sum of weights within a column is greater than 50 (which is the case in the full table), then the combination of knob settings means that the corresponding sensor in the column reads "good." Thus, high transmission power and low transmission frequency promotes an increased signal strength reading as well as good sensor readings for connectivity loss, whereas high transmission power and high transmission frequency promotes better readings for throughput.

The reasoner component determines proper knob settings so that goals are achieved or desired effects take place. The reasoner uses information from the knowledge base as a basis for its deductions: the device model—that is the relationships between knobs, effects, and sensor readings; the goals; the policies; and the current state. When new parameter settings are determined, the reasoner also provides justifications such as what sensor values and/or what relationships from the device model were used to infer the new settings. This can be used for diagnostics if things do not go as expected. The reasoner also specifies sensors that should be monitored and conditions of sensor readings that do not fulfill goals, so that the reasoner can take corrective action.

We developed a formal executable specification of the PAGODA architecture in the Maude language [7] and instantiated it with an abstract device model of a radio to test the ideas.

Goals are treated as soft constraints on subsets of sensor readings. The relationships between affectors (knobs) and sensor readings and between sensor readings and goals are formalized as constraint semi-rings, which provides a clean mathematical basis for solving soft constraints [4,10]. In particular, the sensor tables (see Figure 2 right) assign weights to knob settings. The sum of these weights indicates the benefit of a specific knob setting for a given sensor: the higher the sum of its weights, the better. The goal is to compute for a list of interesting sensors the valuations of the knobs that optimize the sums of these weights. We model this by the constraint semiring of fuzzy natural numbers for grading the weights. The effect constraints indicate the impact of a specific knob setting against a given effect (see Figure 2 left). The goal is to find knob settings that satisfy a maximum of required effects or equivalently violate a minimum of required effects. The best situation occurs if there are no violations of the effects [2]. We model this using the constraint semiring of weighted sums.

We instantiated the PAGODA abstract device specification with a specification of a concrete radio, MadRad, that simulates actual radio hardware/software including random, unusual and faulty behavior. Test scenarios allow us then to explore some possible system behaviors.

3 C-Semirings and Soft Constraints in Maude

3.1 Brief Overview of Maude

Maude [7] is a multiparadigm executable specification language based on rewriting logic [14]. Maude sources, executables for several platforms, the manual, a primer, cases studies, and papers are available from the Maude Web site at `http://maude.cs.uiuc.edu`.

We use *functional theories*, *functional modules* and *parameterized modules* in our framework. Functional modules are equational theories used to specify algebraic data types such as particular constraint semirings; they are declared with the syntax `fmod ... endfm`. Functional modules can have sorts, subsort relationships, operators, variables, membership axioms, and equations, and can import other theories or modules. Functional theories (syntax `fth ... endfth`) are similar to functional modules, they are used to declare module interfaces such as the axiomatic theory of constraint semirings; but opposed to functional modules they have a loose interpretation (as opposed to an initial algebra semantics of functional modules) and do not need to be executable (expressed by the attribute `nonexec`). Parameterized modules (available in Maude 2.2) are used to represent our generic approach to soft constraints. Such a module `P` has the syntax `fmod P{X1 :: T1, ..., Xn :: Tn} ... endfm` where `Xi` are formal parameters and `Ti` functional theories representing the type of the parameters ($i = 1, ..., n$). A *view* `V` (written `view V from T to M is ... endv`) specifies how a particular target module `M` is claimed to satisfy a source theory `T`. An instantiation `P(V1,..., Vn)`of `P` requires a view `Vi`from the type `Ti` of each formal parameter `Xi` to a corresponding actual parameter `Mi` such that each `Mi` satisfies the axioms of `Ti` modulo the renaming specified by `Vi` (i.e. each `Vi` is a theory morphism).

3.2 Constraint Semirings

A *semiring* S is an algebra $\langle G; +; *; 0; 1\rangle$ with carrier set G, two constants $0, 1 \in G$ and two binary operations $+, *$ having the following properties: $+$ is commutative and associative and 0 is its neutral element; $*$ is associative and distributes over $+$, 1 is its neutral element and 0 is its absorbing element. A *constraint semiring* S (*c-semiring* for short) is a semiring $\langle G; +; *; 0; 1\rangle$ with the following two additional properties: $*$ is commutative, and 1 is the absorbing element of $+$. Then - as one of the anonymous referees observed – $+$ is also idempotent. The addition induces a partial ordering relation $\leq$ between the elements of G defined by $a \leq b$ iff $a + b = b$. This partial ordering will be used to compare solutions of constraints and to determine the best solution. If $a \leq b$ we say also that b is better than a. $+$ and $*$ are monotonic w.r.t. $\leq$, 0 is the least element and 1 is the greatest element

of G; $\langle G; \leq \rangle$ forms a complete lattice with $+$ as least upper bound.

Constraint semirings have important closure properties: they are closed under cartesian products and the formation of power sets. An *ic-semiring* is a constraint semiring where $*$ is also idempotent. Then $+$ distributes over $*$ and $\langle G; \leq \rangle$ is a complete distributive lattice with $*$ as its greatest lower bound.

We formalize semirings, constraint semirings, and ic-semirings as Maude theories in a straight-forward way. The sort is called Grade since its elements will be used to grade soft constraints. Note that the last axiom induces transitivity and symmetry of the equivalence relation.

```
fth SEMIRING is
 pr BOOL .
 sort Grade .
 op zero : -> Grade  .   op one  : -> Grade  .
 op _+_ : Grade Grade -> Grade [assoc comm id: zero prec 33] .
 op _*_ : Grade Grade -> Grade [assoc id: one prec 31] .
 op _<=_ : Grade Grade -> Bool [prec 37] .
 op _equiv_ : Grade Grade -> Bool [prec 37] .
 vars X Y Z : Grade .
 eq X * (Y + Z) =  (X * Y) + (X * Z)  [nonexec] .
 eq X * zero = zero                   [nonexec] .
 eq X <= Y  =  (X + Y) == Y           [nonexec] .
 eq X equiv X = true                  [nonexec] .
 ceq X equiv Z = true
     if X equiv Y = true /\  Z equiv Y = true [nonexec] .
endfth

fth C-SEMIRING is
 inc SEMIRING .
 vars X : Grade .
 eq  X + one = one  [nonexec] .
endfth
```

The cartesian product of two c-semirings forms a c-semiring whose operations are defined in the obvious componentwise way. The induced ordering relation $<=_\times$ is a partial ordering. We also introduce the lexicographic ordering $<=$ for applications with a preferred component. The lexicographic ordering is a total ordering which extends $<=_\times$; it is also well-suited for the radio application where the optimization of the effects is preferred to the optimization of the minimal sensor value. Note that in contrast to $<=_\times$, multiplication is in general not monotonic w.r.t. the lexicographic ordering $<=$ but behaves well for the radio application [18]. The cartesian product is modelled as a parameterized module with two constraint semirings as parameters.

```
fmod PAIR{X :: C-SEMIRING, Y :: C-SEMIRING} is
 protecting BOOL .
 sort Pair{X, Y} .
 op pair : X$Grade Y$Grade -> Pair{X, Y} [ctor] .

 op zero : -> Pair{X, Y} . eq zero = pair(X$zero, Y$zero) .
 op one : -> Pair{X, Y} .  eq one = pair(X$one, Y$one) .

 var A A1 A2 : X$Grade . var B B1 B2 : Y$Grade .
 op _+_ : Pair{X, Y} Pair{X, Y} -> Pair{X, Y} [prec 33] .
 eq pair(A1, B1) + pair(A2, B2) = pair(A1 + A2, B1 + B2) .

 op _*_ : Pair{X, Y} Pair{X, Y} -> Pair{X, Y} [prec 31] .
 op _<=*_ : Pair{X, Y} Pair{X, Y} -> Bool [prec 37] .
 op _<=_ : Pair{X, Y} Pair{X, Y} -> Bool [prec 37] .
 op _equiv_ : Pair{X, Y} Pair{X, Y} -> Bool [prec 37] .
 op fst : Pair{X, Y} -> X$Grade .  op snd : Pair{X, Y} -> Y$Grade .
```

```
 . . .
endfm
```

3.3 Boolean, Fuzzy, and Weighted Sum Semirings

Examples of constraint semirings are boolean algebras, and in particular, $Bool = \langle \{false; true\}; \vee; \wedge; false; true \rangle$, "fuzzy algebras" such as "fuzzy natural numbers" $F_N = \langle N \cup \{\infty\}; max; min; 0; \infty \rangle$, and weighted sum algebras, e.g. over natural numbers $W_N = \langle N \cup \{\infty\}; min; +; \infty; 0 \rangle$. Except for weighted sums, all of these algebras are ic-semirings. In Boolean algebras, *true* is better than $false$; for fuzzy natural numbers, 0 is the least and ∞ the greatest element, whereas the ordering for weighted sum is the converse to the usual ordering: ∞ is the least element and 0 the greatest element.

In Maude, *fuzzy natural numbers* are specified as functional module in the following way. We introduce the sort `NatFN` of fuzzy natural numbers with infinity and two subsorts `IftyFN` and `NiNatFN` for representing the ∞ element and the "non-infinity" natural numbers, which are constructed by an embedding of natural numbers.

```
fmod  FUZZYNAT is
 pr NAT .
 sorts NiNatFN IftyFN NatFN .
 subsort NiNatFN IftyFN < NatFN .
 op fn : Nat -> NiNatFN [ctor] .
 op iftyFN : -> IftyFN [ctor] .

 op _+_ : NiNatFN NiNatFN
          -> NiNatFN [prec 33].
 op _+_ : NatFN NatFN
          -> NatFN [prec 33] .
 eq fn(M) + fn(N)
           = fn(max(M, N)) .
 eq fn(M) + iftyFN = iftyFN .
 eq iftyFN + U = iftyFN .

 op _<=_ : NatFN NatFN -> Bool .
 op _equiv_ : NatFN NatFN -> Bool .
 op zero : -> NiNatFN .
 eq zero = fn(0) .
 op one : -> IftyFN .
 eq one = iftyFN .
 vars N M : Nat .
 var U : NatFN .

 op _*_ : NiNatFN NatFN
      -> NiNatFN [prec 31]  .
 op _*_ : NatFN NatFN
          -> NatFN [prec 31] .
 eq fn(M) * fn(N) =
             fn(min(M, N)) .
 eq fn(N) * iftyFN = fn(N) .
 eq iftyFN * fn(N) = fn(N) .
 eq iftyFN * iftyFN = iftyFN .
 . . .
endfm
```

Then, we define a view from `C-SEMIRING` to `FUZZYNAT` and can easily prove by structural induction that all axioms are satisfied, i.e., the view forms a theory morphism.

```
view FuzzyNat from C-SEMIRING to FUZZYNAT is
 sort Grade to NatFN .
endv
```

The constraint *semiring of weighted sums* `WSUM`over natural numbers with infinity and the theory morphism `WSum`from `C-SEMIRING`to the weighted sum algebra are specified in an analogous way.

In our application to software-defined radios we use the cartesian product of the above constraint semirings on natural numbers with infinity. It is defined by instantiating the parameterized module `PAIR` using the views `WSum` and `FuzzyNat`. The cartesian product forms also a c-semiring which is expressed by the view `WSumFuzzyPair`.

```
fmod  WSUMFUZZYPAIR is
 pr PAIR{WSum, FuzzyNat} .
endfm
```

```
view WSumFuzzyPair from C-SEMIRING to WSUMFUZZYPAIR is
 sort Grade to Pair{WSum, FuzzyNat} .
endv
```

3.4 Soft Constraints

A soft constraint assigns grades to different valuations of a set of problem variables. Let S be a constraint semiring with carrier set G, D a finite problem domain, and Var the set of all problem variables. A *valuation* $\upsilon : Var \longrightarrow D$ is a partial map from Var to D which has a finite support. Given an ordered list $al \in Var$ of variables, a soft constraint assigns a grade in G to each possible valuation of variables in al; more formally, a *soft constraint* is a pair $\langle al; cst \rangle$ where $cst \in (Var \longrightarrow D) \longrightarrow G$ is a mapping from valuations to elements of G such that every valuation of the domain of cst has finite support al. As the set of all such valuations is finite, any soft constraint has a finite domain of definition and can therefore be represented either in an *explicit* way by a finite set of pairs $(\upsilon \mapsto g)$ with $\upsilon \in (al \longrightarrow D)$ and $g \in G$ or in a more *implicit* way by particular constructors or function declarations as in a programming language. In our framework we support both possibilities; here we present only the explicit form and a particular implicit form we need for our application to software-defined radios.

For representing valuations in Maude, we define a total ordering on the set Var of all variables. The module VALUATION is parameterized by theories for an ordered set X of variables and an ordered domain D (using the theory TAO-SET provided in the Maude prelude); each valuation $\upsilon = (a_1 \mapsto d_1, ..., a_k \mapsto d_k)$ is represented by a pair $[al \mapsto dl]$ consisting of the ordered list $al = a_1, ..., a_k$ of variables and the list $dl = d_1, ..., d_k$ of corresponding elements of D. For fixed al, $(al \longrightarrow D) \longrightarrow G$ is isomorphic to $D^k \longrightarrow G$ and thus any constraint definition cst can be represented as a map from D^k to G. In Maude we specify such maps by a parameterized module LC-MAP similar to the MAP module of the Maude prelude, except that the range is a constraint semiring S and, for efficiency reasons, the domain of any map consists of a *list of elements* of D. As in MAP, we introduce a parameterized sort LC-Map for maps and a subsort for simple entries of the form $(dl \mapsto g)$; for reasons of space efficiency, we omit entries $(dl \mapsto 0)$ from the LC-Map terms. Moreover, the specification contains efficient operations for sorting maps according to the values of different lexicographic orderings of the domain and according to the values of the codomain. SORTABLE-LIST1 is an extension of SORTABLE-LIST (see the Maude 2.2 prelude) by an operation noDupmerge for merging two ordered lists without duplicating elements.

```
fmod VALUATION{X :: TAO-SET, D :: TAO-SET} is
 pr EXT-BOOL . pr SORTABLE-LIST1{X} . pr SORTABLE-LIST1{D} .
 sort Valuation{X, D} .
 op [_ |-> _] :  List{X} List{D} -> Valuation{X, D} [ctor] .
 op variabs : Valuation{X, D} -> List{X} .
 op values : Valuation{X, D} -> List{D} .
 op consistent : Valuation{X, D} Valuation{X, D}  -> Bool .
 *** checks the equality of the values of the common variables
 *** of both valuations.
 op mergeEntry : Valuation{X, D} Valuation{X, D} -> List{D} .
 *** merges the values of two consistent valuations.
 ...
endfm

fmod LC-MAP{D :: TAO-SET, S :: C-SEMIRING} is
```

```
 protecting SORTABLE-LIST{D} .
 sorts LC-Entry{D,S} LC-Map{D, S} .
 subsort LC-Entry{D,S} < LC-Map{D, S} .
 op empty : -> LC-Map{D,S} [ctor] .
 op (_|->_) : List{D} S$Grade -> LC-Entry{D,S} [ctor] .
 op __ : LC-Map{D, S} LC-Map{D, S} -> LC-Map{D, S}
         [ctor assoc id: empty] .
 ...
 op sortCoDomain : LC-Map{X, Y} -> LC-Map{X, Y} .
 . . .
endfm
```

The specification CONSTRAINT is parameterized by theories X for Variables, D for the problem domain, and S for the c-semiring. It has parameterized sorts for constraints and list of constraints, and subsorts *EConstraint* and *IConstraint* for explicit and implicit constraints as well as the sort *ZeroConstraint* for the constraint with constant grade 0 and the sort *NoConstraint* for the situation with no constraint at all. The explicit representation of a soft constraint has the form

$$[al \mid (val_1 \longmapsto g_1), ..., (val_m \longmapsto g_m)]$$

where all grades $g_1, ..., g_m$ are different from 0 (as in LC-MAP). The operations of c-semirings and the operations of LC-MAP are lifted to constraints as follows. The application of a constraint $c = [al \mid cst]$ to a valuation v is defined as map application: $c\,[v] = cst\,[v]$; *zeroConstraint* and *noConstraint* define the constraints with constant values *zero* and *one*; constraint multiplication is defined to satisfy the equation $c_1 * c_2\,[v] = c_1\,[v] * c_2\,[v]$, and constraint addition is defined analogously. Then it is easy to prove that the set of constraints forms again a c-semiring [5].

Moreover, we define a projection operator *project* and (for constraint propagation) a partial evaluation operator *peval*. For any constraint $c = [al \mid cst]$ and any valuation $v = [bl \mapsto dl]$ with $bl \subseteq al$, $peval(c, v)$ restricts c to the valuations consistent with v; in particular, for $bl = al$ we have $peval(c, v) = [al \mid (dl \mapsto cst\,[v])]$. For a constraint c and an ordered list of variables xl, $project(c, al)$ computes the sum $\sum_{dl \in D^k} peval(c, [al \mapsto dl])$ of all possible partial evaluations of al. For efficiency, all operations on constraints are specified in a tail recursive way. As an example, we show the specification of constraint multiplication.

```
fmod CONSTRAINT{X :: TAO-SET, D :: TAO-SET, S :: C-SEMIRING} is
 protecting EXT-BOOL . protecting NAT . protecting SORTABLE-LIST1{X} .
 protecting VALUATION{X, D} . protecting LC-MAP{D, S} .
 sorts NoConstraint{X, D, S}    ZeroConstraint{X, D, S}
       SimpleConstraint{X, D, S}  EConstraint{X, D, S}
       Constraint{X, D, S}        ListConstraint{X, D, S} .
 subsort NoConstraint{X, D, S} ZeroConstraint{X, D, S}
    < SimpleConstraint{X, D, S} < EConstraint{X, D, S}
    < Constraint{X, D, S}       < ListConstraint{X, D, S} .
 op noConstraint : -> NoConstraint{X, D, S}  [ctor] .
 op zeroConstraint : -> ZeroConstraint{X, D, S}  [ctor] .
 op [_|_] : List{X} LC-Entry{D, S} -> SimpleConstraint{X, D, S} [ctor] .
 op [_|_] : List{X} LC-Map{D, S} -> EConstraint{X, D, S} [ctor] .
 op _+_ : Constraint{X, D, S} Constraint{X, D, S} -> Constraint{X, D,S} .
 op _*_ : Constraint{X, D, S} Constraint{X, D, S} -> Constraint{X, D,S} .
 op _[_] : Constraint{X, D, S} Valuation{X, D} -> S$Grade .
 op peval : EConstraint{X, D, S} Valuation{X, D} -> EConstraint{X, D,S} .
 op project : EConstraint{X, D, S} List{X} -> EConstraint{X, D, S} .
   ...
```

```
 vars AL BL : List{X} . vars VL WL L : List{D} .
 vars EN1 EN2 : LC-Entry{D, S} .  vars M M1 M2 Result : LC-Map{D, S} .
 vars P Q : S$Grade .  var C :  Constraint{X, D, S} .

 eq noConstraint * C = C . eq [AL | M] * noConstraint = [AL | M] .
 eq zeroConstraint * C = zeroConstraint .
 eq [AL | M] * zeroConstraint = zeroConstraint .
 eq [AL | empty] * [BL | M] = [noDupMerge(AL, BL) | empty] .
 eq [AL | (VL |-> P) M1] * [BL | M2] =
  if P =/= zero
      then [noDupMerge(AL, BL) | [AL | (VL |-> P) M1] *Map [BL | M2]]
      else [noDupMerge(AL, BL) | [AL | M1] *Map [BL | M2]] fi .

 op _*Map_ : SimpleConstraint{X, D, S}
             SimpleConstraint{X, D, S} -> LC-Map{D, S} .
 eq [AL|(VL |-> P)] *Map [BL|(WL |-> Q)] =
    if (P * Q =/= zero) and-then consistent([AL |-> VL], [BL |-> WL])
       then (mergeEntry([AL |-> VL], [BL |-> WL]) |-> P * Q)
       else empty
    fi .

 op _*Map_ : EConstraint{X, D, S} EConstraint{X, D, S} -> LC-Map{D, S} .
 op _$*Map_with_is_ : EConstraint{X, D, S} EConstraint{X, D, S}
                      LC-Map{D, S} LC-Map{D, S} -> LC-Map{D, S} .
 eq [AL | M1] *Map [BL | M2] =
        [AL | M1] $*Map [BL | M2] with M2 is empty .
 eq [AL | EN1 M1] $*Map [BL | EN2 M2] with M is Result =
        [AL | EN1 M1] $*Map [BL | M2] with M
          is (Result ([AL| EN1]*Map [BL| EN2])) .
 eq [AL | EN1 M1] $*Map [BL | empty] with M is Result =
       [AL | M1] $*Map [BL | M] with M is Result .
 eq [AL | empty] $*Map [BL | M2] with M is Result = Result .
   ...
endfm
```

3.5 Hard Constraints and Implicit Constraints

Hard constraints can be considered as a special class of soft constraints: those over the two-valued boolean semiring $Bool = \langle \{false; true\}; \vee; \wedge; false; true \rangle$ where *true* indicates the satisfaction of a hard constraint and *false* its violation [4]. Often hard and soft constraints occur together in the same problem. Let S be the c-semiring of the soft constraints. Then there are several possibilities for encoding hard constraints:

- Choose S as c-semiring and represent satisfaction of a hard constraint by 1_S and its violation by 0_S. This yields non-zero grades for consistent combinations of soft and hard constraints that satisfy all hard constraints; if one of the hard constraints is violated the resulting weight is 0_S.
- Build the cartesian product $Bool \times S$ with the lexicographic ordering. Then satisfaction of a hard constraint is expressed by $\langle 1_{Bool}, g \rangle$ and violation by $\langle 0_{Bool}, g \rangle$ for some $g \in S$. This gives us a finer grained analysis of hard constraints violation: one can not only distinguish the grades of those combinations of constraints that satisfy all hard constraints, but also the grades of those soft constraints that violate one of the hard constraints.

For our radio application, we choose a refinement of the latter solution: The c-semiring is the lexicographically ordered cartesian product WSUMFUZZYPAIRof weighted sum and fuzzy naturals (see Section 3.3). Hard constraints of the form $n \geq constant$ occur for the sensor constraints, which have values in the c-semiring of "fuzzy natural number constraints". We represent satisfaction of such a hard constraint by $\langle 1_{WSum}, n\rangle$ and violation by $\langle 0_{WSum}, n\rangle$, thus giving also information about the grades of the sensor constraint.

In general, implicit constraints are defined by structural recursive expressions and by particular application-dependent constructors. As operations, we provide partial evaluation and transformation into explicit constraints.

For the radio application, we use the sensor tables as a space-efficient representation of the sensor constraints. The naive representation of the sensor values by summation leads to constraints with many variables and therefore a potentially exponential number of entries; e.g., the explicit constraint for the "signal strength" has the form (with TP for transmission power, TF for transmission frequency, and CP for compression):

$$[TP\ TF\ Cp\ ...\mid (Hi\ Hi\ Hi ... \mapsto 50)(Lo\ Hi\ Hi ... \mapsto 0)...]$$

Instead we use an implicit constraint (with constructor sm for "sensor map") for a list representation which requires only linear space:

$$sm([TP \mid (Hi \mapsto 50)(Lo \mapsto 0)]\,[TF \mid (Hi \mapsto 0)\ (Lo \mapsto 50)]\,...)$$

During the process of constraint solving we always try to apply partial evaluation as much as possible in order to reduce the size of implicit constraints before transforming them into explicit constraints.

4 Solving Soft Constraints

The solution space of a list of soft constraints $cl = cl_1...cl_n$ is given by the multiplication $cl_1 * ... * cl_n$ of all constraints; the result is again a soft constraint c which can be represented as a sum $sc_1 + ... + sc_m$ of simple constraints (where $m \leq |D|^{|variabs(cl)|}$). Each such simple constraint is called a possible solution of C. Not all of these possible solutions are of interest for applications. In the following we write $\prod cl$ as a shorthand for $cl_1 * ... * cl_n$ and $\sum cl$ for $cl_1 + ... + cl_n$

In many cases we search for the set $maxSol(cl)$ of all best solutions, i.e., all elements of $D^{|variabs(C)|}$ with maximal grades in the solution space, or for the solutions with a grade better than a certain threshold. One can also be more general and compute the projection $project(cl_1 * ... * cl_n, xl)$ to a subset xl of the variables, if only these variables are of interest and then consider the same classes of solutions.

For the radio application, we need algorithms for both: for finding one admissible solution in a short time and, in cases where enough computing power and time are available, for finding all best solutions or all admissible solutions. But we could simplify the job in that all variables (knobs) were of interest and the involved semirings were all totally ordered. Following, we make therefore these simplifying assumptions and present a Maude realisation for searching all best solutions and give a sketch of the implementation of the other search algorithms.

The implementation consists of the following three parameterized modules: a module SOLVECONSTRAINT for the two search algorithms and two modules SOLUTION and CONTINUATION for representing the data type of solutions and storing the necessary backtracking information. The module SOLUTION provides explicit informations about all solutions, their grade and their number; in particular, $solution(scl, g, n)$ is a constructor term consisting of a list scl of simple constraints denoting solutions, the grade g of all elements of scl, and the number $n = |scl|$ of solutions. Continuations are defined by the module CONTINUATION; a continuation $ct(cl0, sc)$ consists of a partial solution sc and a list $cl0$ of unsolved constraints with the intended property that the combination $sc * \prod cl0$ of sc with all constraints in $cl0$ forms again a set of possible solutions of the original constraint problem cl.

The module SOLVECONSTRAINT defines depth-first search algorithms for finding all best solutions, for finding a first solution better than a certain threshold, and for finding all such solutions (see [3] for a similar approach). For any list cl, $search(cl)$ computes the set $maxSol(cl)$ of all best solutions. For efficiency, it assumes for all constraints $[al \mid (val_1 \longmapsto g_1), ..., (val_m \longmapsto g_m)]$ in cl that all grades $g_1, ..., g_m$ are in descending order. $search$ uses an auxiliary tail-recursive operation $\$search(cl, sc, cont, csol)$ where cl denotes the list of constraints to be solved, sc the actual partial solution, $cont$ the continuation, and $csol = solution(scl, b, n)$ the constraint solution obtained so far.

The most interesting case of the recursive definition is $cl = [al \mid (vl \mapsto p)erest]\, rest$ and $sc = [bl \mid (wl \mapsto q)]$ where vl has grade p, $erest$ denotes the map consisting of the remaining assignments of the first constraint of cl, and $rest$ denotes the tail of the list of constraints cl. (All other cases are simpler with $erest$, $rest$ or cl being empty, or sc corresponding to $noConstraint$.)

If the grade $p * q$ of the new solution is not smaller than b and different from $zero$ and if the first subconstraint $c_0 = [al \mid (vl \mapsto p)]$ of cl is consistent with sc then $\$search$ computes a new partial solution $[al \mid (vl \mapsto p)] * sc$ and saves the rest of the constraint in the continuation for later backtracking. Otherwise, if $p * q$ is large enough but c_0 not consistent with sc, the search continues with $erest$; finally, if $p * q < b$, we can use the fact that every constraint is sorted in a descending order of grades which implies that for all grades g of the entries of $erest$ we have also $g * q < b$; therefore the partial solution sc cannot be completed to a best solution and the algorithm backtracks with the continuation.

```
fmod SOLVECONSTRAINT{X :: TAO-SET, D :: TAO-SET, S :: C-SEMIRING} is
 pr CONSTRAINT{X, D, S} . pr CONTINUATION{X, D, S} .
 pr SOLUTION{X, D, S} .

 var N : Nat . vars AL BL : List{X} . vars VL WL : List{D} .
 var ERest : LC-Map{D, S} . vars P Q B : S$Grade  .
 var SC : SimpleConstraint{X, D, S} .
 var EC : EConstraint{X, D, S} . var  C :  Constraint{X, D, S} .
 vars L Rest : ListConstraint{X, D, S} .
 var Cont : ListContinuation{X, D, S} . var Result : Solution{X, D, S} .

 op search : ListConstraint{X, D, S} -> Solution{X, D, S} .
 eq search(L) = $search(sortElements(L), noConstraint, nil,
                        solution(nil, zero, 0) ) .
```

```
 op $search : ListConstraint{X, D, S} Constraint{X, D, S}
     ListContinuation{X, D, S} Solution{X, D, S} -> Solution{X, D, S} .
 ...
 eq $search([AL | (VL |-> P) ERest] Rest, [BL | (WL |-> Q)], Cont,
       solution(L, B, N)) =
    if (B <= P * Q) and (zero < P * Q) then
      if consistent([AL |-> VL], [BL |-> WL]) then
         $search(Rest, [AL | (VL |-> P)] * [BL |  (WL |-> Q)],
           ct([AL | ERest] Rest, [BL | (WL |-> Q)]) Cont,
           solution(L, B, N))
      else $search([AL | ERest] Rest, [BL | (WL |-> Q)], Cont,
                solution(L, B, N))
      fi
    else $backtrack(Cont, solution(L, B, N))
    fi .
 eq $search(C, SC, Cont, Result) = $backtrack(Cont, Result) [owise] .

 op $backtrack : Continuation{X, D, S}
                 Solution{X, D, S} -> Solution{X, D, S} .
 eq $backtrack(nil, Result) = Result .
 eq $backtrack(ct(L, C) Cont, Result) = $search(L, C, Cont, Result) .
 ...
endfm
```

It is easy to see that the search algorithm is terminating. The following lemma is the basis for the correctness proof of the search algorithm.

Lemma 4.1 (search invariant) *For all lists $cl = cl_1...cl_m$ of explicit constraints (where each cl_i is sorted in a descending order of grades), for all partial constraints sc, all lists of continuations cont of the form $ct(contl_1, scont_1)$, ..., $ct(contl_k, scont_k)$, and all solutions csol of the form $solution(scl, b, n)$ where scl is a list of n of possible solutions with grade b, i.e., scl is a list of simple constraints of the form $scl_i = [xl \mid (wl_i \mapsto b)]$, the following holds:*

(i) $\$backtrack(cont, csol) = maxSol(\sum_{j=1}^{k}(scont_j * \prod contl_j) + \sum scl)$;

(ii) $\$search(cl, sc, cont, csol) = maxSol(sc * \prod cl + \sum_{j=1}^{k}(scont_j * \prod contl_j) + \sum scl)$.

Proof. By simultaneneous computational induction on both operations. □

The correctness of the search operation follows directly from termination and the invariant lemma:

Theorem 4.2 (total correctness of search) *For all lists of explicit constraints $cl = cl_1...cl_m$ of explicit constraints (where each cl_i is sorted in a descending order of grades), the following holds:*
$search(cl) = maxSol(cl)$.

Proof.

$$\begin{aligned} search(cl) &= \$search(sortElements(cl), noConstraint, nil, solution(nil, zero, 0)) \\ &= maxSol(noConstraint * \prod cl + zeroConstraint + zeroConstraint) \\ &= maxSol(\prod cl), \end{aligned}$$

where the second equality uses the lemma. □

5 Experimentation Results

One objective of the radio application is to determine radio parameters for a networked radio team such that the mission goals are satisfied. We formalize this problem as a soft constraint problem, implement it in our Maude soft constraint framework and integrate it with the PAGODA system for experiments.

The sensor tables (see Figure 2 right) assign weights to knob settings. The goal is to compute for a list of interesting sensors the valuations of the knobs that optimize the sums of these weights (see Section 2). Thus, we choose the c-semiring of fuzzy natural numbers for grading the weights. Given a sensor s, a list of knobs $knl = kn_1, ..., kn_m$, a valuation $\upsilon = (kn_1 \mapsto d_1, ..., kn_m \mapsto d_m)$ into a domain $KnobVal$ of knob values and associated weights $g_{s,1}, ..., g_{s,m}$, a sensor constraint c_s has the form $(knl; cst_s : KnobVal^m \longrightarrow NatFN)$ where $cst_s(d_1, ..., d_m) = \sum_{i=1}^{m} g_{s,i}$. For a list sl of sensors we maximize the minimal value of these sums, i.e., we compute $max\,\{d | min_{s \in sl}\, cst_s(d)\}$ for all $d \in KnobVal^m$.

The effect constraints indicate the impact of a specific knob setting against a given effect (see Figure 2 left). The more "+"s are in the column of an effect, the more likely it is that the knob setting will achieve the effect. The goal is to find knob settings that violate a minimum of required effects (see Section 2). We model this using the c-semiring of weighted sums by assigning grade *one* to "+" and grade *zero* to "-". For a given effect e, any knob kn_i, value d_i and associated grade $g_{e,i}$, we define a soft constraint $(kn_i; cst_{e,i} : KnobVal \longrightarrow NatWS)$ where $cst_{e,i}(d_i) = g_{e,i}$. For a list el of effects we minimize the sums of these values, i.e., we compute $min\left\{d | \sum_{e \in el} \sum_{i \in \{1,...,m\}} cst_{e,i}(d_i))\right\}$.

For the radio application, we perform a multicriteria optimization by minimizing the violations of the effect goals and maximizing the benefits of the knob settings where we give preference to the optimization of the effect goals. Formally, this means to build the lexicographically ordered cartesian product of the semirings of weighted sums and of fuzzy natural numbers (modelled by `WSUMFUZZYPAIR`). Note that by exchanging the order of the two semirings we could give also priority to the benefits of the knob settings.

To model this application in Maude, we instantiate the constraint framework with the data types of the radio application and integrate it with PAGODA.

For representing knobs, we use the existing sorts `Knob`and `KnobVal`of the PAGODA system. `Knob`defines a list of 14 constants such as `TP`, `TF`, `Cp`, `ECC`representing transmission power, transmission frequency, compression, and error correction code. `KnobVal` has in the current implementation only two values `Hi`and `Lo`.Effect constraints are defined as combinations of simple one-variable constraints with values in the left pair component; e.g., the effect constraint `ImprovedConnectivityCstr` for the goal "Improved Connectivity" which requires sensor values `TP = Hi, TF = Lo,...` is represented by the product of one-variable constraints:

```
[TP | (Hi |-> 0) (Lo |-> 1)] * [TF | (Hi |-> 1) (Lo |-> 0)] * ...
```

no. constraints	duration $search(...)$	durations $searchOne(..., \alpha)$
4	0,22 sec.	0,02 sec
6	0,26 sec.	0,03 sec
8	0,58 sec.	0,03 sec
12	3,31 sec.	0,14 sec
17	82,61 sec.	1,49 sec

Table 1
experimentation results

Sensor constraints are represented by implicit constraints as described in section 3.5. As a hard constraint, we require that sensor weights have a value n greater than a certain threshold α. It is represented by $\langle 1_{WSum}, n\rangle$ for $n \geq \alpha$ and by $\langle 0_{WSum}, n\rangle$ for $n < \alpha$.

For integrating this instantiation of the soft constraint framework with PAGODA it suffices to call one of the search functions within the reasoner component. E.g., a call

```
search(ImprovedConnectivityCstr HighBWCstr ...)  .
```

returns two best solutions each of which has four effect inconsistencies and a value 60 for the minimal sum of the weights of the sensors:

```
rewrites: 652800 in 4640438906ms cpu (2142ms real) (0 rewrites/second)
result Solution{Knob,KnobVal,WSumFuzzyPair}: solution(
 [TP TF PktSize Cp TW ECC RU QR Encr Cach SS DT CS
 | Lo Lo Hi Hi Hi Lo Lo Lo Lo Hi Lo Lo Lo |-> pair(ws(4), fn(60))]
 [TP TF PktSize Cp TW ECC RU QR Encr Cach SS DT CS
 | Lo Hi Hi Hi Hi Lo Lo Lo Lo Hi Lo Lo Lo |-> pair(ws(4), fn(60))],
 pair(ws(4), fn(60)), 2)
```

Searching just one solution with the same threshold (using `searchOne`) is typically at least 10 times faster. The following table gives an overview on the experimentation results which were performed using an Intel Pentium M 713 CPU 1.1 GHz, 512 MB RAM notebook running Core Maude Version 86a over Windows XP. The first column lists the number of sensors and effects under consideration, the second and third column indicate the CPU time consumption for finding all best solutions and the first best solution.

The table shows that most optimal parameter assignments for software-defined radio can be computed in a few seconds; only the most difficult constraint sets need more than one minute for the construction of the solutions.

6 Future Directions and Concluding Remarks

In this paper, we presented a Maude framework for prototyping of soft constraints à la Bistarelli, Montanari and Rossi. The framework is written in a modular and parameterized way and easily instantiable to different applications. As a case study, we integrated our framework with SRI's PAGODA architecture for autonomous

systems and instantiated it by an application to software-defined radios. Experimentation shows that the constraint solver is well-suited for prototyping and has acceptable performance for (rather simple) constraint problems.

For more complex (and realistic) models, we need to extend our framework in several directions. In order to increase performance, we plan to use precomputed solutions for standard scenarios and to do heavy computation of optimal solutions only in non-standard situations. In order to get more flexibility for selecting and evaluating constraints, it would be useful to develop a strategy language similar to [6] for designing specialized constraint solvers. Moreover, simple soft constraints as in this paper will not be enough. More complex policies for tasks such as resource allocation and providing quality of service have to be respected and solutions have to be compared using preferences. Based on the successful c-semiring-based approaches to preferences [11], quality of service [10] and soft concurrent constrained programming [5], we are currently developing an expressive modelling language for policies involving preferences and hard/soft constraints which will also be based on the c-semiring approach.

Acknowledgement

Thanks to the anonymous referees for their helpful suggestions and remarks.

References

[1] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming.* volume 2962 of *Lecture Notes in Computer Science* 2004. Springer-Verlag.

[2] S. Bistarelli, E. C. Freuder, B. O'Sullivan. Encoding Partial Constraint Satisfaction in the Semiring-Based Framework for Soft Constraints. In *16th IEEE Internat. Conf. on Tools with Artificial Intelligence (ICTAI 2004). IEEE Computer Society 2004*, 240–245.

[3] S. Bistarelli, T. Frühwirth, M. Marte, F. Rossi: Soft Constraint Propagation and Solving in Constraint Handling Rules. *Computational Intelligence, Special Issue on Preferences in AI and CP, Blackwell Publishing* (to appear).

[4] S. Bistarelli, U. Montanari, and F. Rossi. *Semiring-based Constraint Solving and Optimization. Journal of ACM* 44:201–236, 1997.

[5] S. Bistarelli, U. Montanari and F. Rossi. Soft Concurrent Constraint Programming. To appear in *ACM Transactions on Computational Logic (TOCL).*

[6] Carlos Castro and Eric Monfroy. A Strategy Language for Solving CSPs. In K. Apt, P. Codognet, and E. Monfroy (eds.): Proc. of Third ERCIM Workshop of the Working Group on Constraints, ERCIM'98, 1998.

[7] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Marti-Oliet,J. Meseguer,and C. Talcott. The Maude 2.0 System. Rewriting Techniques and Applications (RTA'03). *Lecture Notes in Computer Science*, 2003. Springer-Verlag.

[8] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and C. Talcott. Maude 2.0 Manual. Computer Science Laboratory, SRI International, 2003, http://maude.csl.sri.com/maude2-manual.

[9] A. Delgado, C. Olarte, J. Perez, and C. Rueda. Implementing Semiring-Based Constraints using Mozart. Proc. of Second International Mozart/Oz Conference. Springer-Verlag LNCS 3389, 2004.

[10] R. De Nicola, G. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A Basic Calculus for Modelling Service Level Agreements. Coordination 2005. Springer-Verlag LNCS 3454, 2005.

[11] C. Domshlak, S. Prestwich, F. Rossi, K. B. Venable, and T. Walsh. Hard and soft constraints for reasoning about qualitative conditional preferences. To appear in *Journal of Heuristics*, special issue on preferences, 2005.

[12] D. Dvorak, R. Rasmussen, G. Reeves, and A. Sacks. Software Architecture Themes in JPL's Mission Data System. IEEE Aerospace Conference, USA, 2000, http://techreports.jpl.nasa.gov/1999/99-1886.pdf

[13] Y. Georget and P. Codognet. Compiling Semiring-based Constraints with clp(FD,S). In Maher, M., and Puget, J.-F., eds., *CP98, Principles and Practice of Constraint Programming*, 205219. Springer-Verlag LNCS 1520, 1998.

[14] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science* 96:73–155, 1992.

[15] N. Muscetolla, P. Pandurang, B. Pell, and B. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before, *Artif. Intelligence* 103:5–48, 1998.

[16] S. Prestwich, F. Rossi, K. B. Venable, T. Walsh. Constrained CP-nets. Preprint n. 13-2004, Dept. of Pure and Applied Mathematics, University of Padova, Italy.

[17] Hana Rudova. Soft CLP(FD). In Susan Haller and Ingrid Russell, editors, *FLAIRS '03, Recent Advances in Artificial Intelligence: Proceedings of the Sixteenth International FLAIRS Conference. AAAI Press*, pages 202–206, 2003.

[18] Martin Wirsing, Matthias Hölzl, Grit Denker, and Carolyn Talcott. A Language for Soft Constraints and Preferences: Syntax, Semantics, and Execution. Forthcoming.