Semiring-based Constraint Logic Programming

Content areas: constraint satisfaction, automated reasoning
Tracking number: A422

Abstract

We extend the Constraint Logic Programming (CLP) formalism in order to handle semiring-based constraint systems. This allows us to perform in the same language both constraint solving and optimization. In fact, constraint systems based on semirings are able to model both classical constraint solving and more sophisticated features like uncertainty, probability, fuzzyness, and optimization. We then provide this class of languages with three equivalent semantics: model-theoretic, fixpoint, and proof-theoretic, in the style of CLP programs.

1 Introduction

Classical constraint satisfaction problems (CSPs) [Mac92] are a very expressive and natural formalism to specify many kinds of real-life problems. However, they also have evident limitations, mainly when they are used to represent real-life scenarios where the knowledge is not completely available nor crisp. In fact, in such situations, the ability of stating whether an instantiation of values to variables is allowed or not is not enough or sometimes not even possible. Recently, a proposal which extends classical CSPs in this direction has been developed [BMR95], which is able to model many desired features, like fuzzyness [DFP93], probability [FL93], uncertainty, partiality [FW92], hierarchy [BMMW89], and optimization. This framework is based on the observation that a semiring (that is, a domain plus two operations satisfying certain properties) is all what is needed to describe many constraint satisfaction schemes. In fact, the domain of the semiring provides the levels of consistency (which can be interpreted as cost, or degrees of preference, or probabilities, or others), and the two operations define how to combine constraints together. In particular, from one of the operations we can derive a partial order $\leq$ among the elements of the semiring which allows us to compare different elements: if $a \leq b$ then it means that b is *better* than a. This is crucial in situations which involve some kind of optimization. Constraint problems described according to this framework are called SCSP (for Semiring-based Constraint Satisfaction Problems).

Constraint logic programming (CLP) [JL87] languages extended logic programming (LP) by replacing term equalities with constraints and unification with constraint solving. Programming in CLP means choosing a constraint system for a specific class of constraints (for example, linear arithmetic constraints, or finite domain constraints) and embedding it into a logic programming engine. This approach is very flexible since one can choose among many constraint systems without changing the overall programming language, and has shown to be very successful in specifying and solving complex problems in terms of constraints of various kind. However, it can handle only classical constraint solving. Thus it is natural to try to extend the CLP formalism in order to be able to handle also SCSP problems. We will call such an extension SCLP (for Semiring-based CLP).

In passing from CLP to SCLP languages, we will replace classical constraints with the more general SCSP constraints. By doing this, we also have to modify the notions of interpretation, model, model intersection, and others, since we have to take into account the semiring operations and not the usual CLP operations. For example, while CLP interpretations associate a truth value (either *true* or *false*) to each ground atom, here ground atoms must be given one of the elements of the semiring. Also, while in CLP the value associated to an existentially quantified atom is the *logical or* among the truth values associated to each of its instantiations, here we have to replace the *or* with another operation which refers to one of the semiring operations.

After describing the syntax of SCLP programs, we will define three equivalent semantics for such languages: model–theoretic, fixpoint, and operational. These semantics are conservative extensions of the corresponding ones for LP, since by choosing a particular semiring (the one with just two elements, *true* and *false*, and the logical *and* and *or* as the two semiring operations) we get exactly the LP semantics. The extension is in some cases predictable but it possesses some crucial new features. For example, the presence of a partial order among the semiring elements (and not a *total* order like it is in the LP/CLP case, where we just have two comparable elements) brings some conceptual complexity in some aspects of the semantics. In fact, in the operational semantics there could be two refutations for a goal which lead to different semiring elements which are not comparable in the partial order. In this case, these elements have to be combined in order to get the solution corresponding to the given goal, and their combination could be not reachable by any derivation path in the search tree. This means that any constructive way to get such a solution by visiting the search tree would have to follow all the incomparable paths before being able to find the correct answer.

A recent approach to multi-valued logic programming [MPS97] uses bilattices with two orderings to model both truth and knowledge levels. While the motivations behing this approach is very similar to ours, the development is rather different, and the resulting logic programming semantics is just operational and fixpoint, while no model-theoretic semantics is presented. Moreover, the presence in our approach of just one ordering (modelling truth levels) is not a restriction, since the vectorization of several semirings is still a semiring (see [BMR95]) and thus optimization based on multiple criteria can be cast in our framework as well.

2 Semiring-based CSPs

Here we give the basic notions about constraint solving over semirings, introduced in [BMR95].

Definition 1 (semiring) *A semiring is a tuple* $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ *such that*

- A *is a set and* $\mathbf{0}, \mathbf{1} \in A$;
- $+$, *called the additive operation, is a closed (i.e.,* $a, b \in A$ *implies* $a + b \in A$*), commutative (i.e.,* $a + b = b + a$*) and associative (i.e.,* $a + (b + c) = (a + b) + c$*) operation such that* $a + \mathbf{0} = a = \mathbf{0} + a$ *(i.e.,* $\mathbf{0}$ *is its unit element);*
- $\times$, *called the multiplicative operation, is a closed and associative operation such that* $\mathbf{1}$ *is its unit element and* $a \times \mathbf{0} = \mathbf{0} = \mathbf{0} \times a$ *(i.e.,* $\mathbf{0}$ *is its absorbing element);*
- $\times$ *distributes over* $+$ *(i.e.,* $a \times (b + c) = (a \times b) + (a \times c)$*).* □

Definition 2 (c-semiring) *A c-semiring is a semiring* $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ *such that*

- $+$ *is idempotent (i.e.,* $a \in A$ *implies* $a + a = a$*);*
- $\times$ *is commutative;*
- $\mathbf{1}$ *is the absorbing element of* $+$. □

The following is a list of properties about c-semirings which will be used in this paper:

- Given any c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, the relation $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$ is a partial order.
- Since $\mathbf{0}$ is the unit element of the additive operation, it is the minimum element of the ordering. Thus, for any $a \in A$, we have $\mathbf{0} \leq_S a$.
- Both the additive and the multiplicative operation are monotone on the ordering $\leq_S$.
- Since $\mathbf{1}$ is also the absorbing element of the additive operation, then $a \leq_S \mathbf{1}$ for all a. Thus $\mathbf{1}$ is the maximum element of the partial ordering. This implies that the $\times$ operation is *intensive*, that is, that $a \times b \leq_S a$. This is important since it means that combining more constraints leads to a worse (w.r.t. the $\leq_S$ ordering) result.
- Given a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, and its partial order $\leq_S$, $\langle A, \leq_S\rangle$ is a lattice. Moreover, for any $a, b \in A$, we have $a \vee b = a + b$, where $\vee$ is the sup operation of the lattice.
- Given a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, consider the corresponding lattice $\langle A, \leq_S\rangle$. If $\times$ is idempotent, then we have that:
  1. $+$ distributes over $\times$;
  2. $\times = \wedge$, where $\wedge$ is the inf operation of the lattice;
  3. $\langle A, \leq_S\rangle$ is a distributive lattice.

Definition 3 (SCSPs) *A constraint problem based on semirings (SCSP) consists of a set of variables with a finite domain* D *and a set of constraints. Each constraint, which connects a subset of the variables* V*, is defined by associating an element of the semiring with each tuple of values of* D *for the variables in* V*.* □

Note that the elements of the chosen semiring can be interpreted in many ways: cost, level of preference, certainty, probability, etc. Note also that the intuitive meaning of the partial order $\leq_S$ is to state when an element is better than another one: if $a \leq_S b$ then we mean that b is better than a. Finally, it is interesting to notice that classical CSPs are just SCSPs where the semiring has just two values: *true* and *false*, and the two operations are *logical and* and *logical or*. That is, the semiring is $S_{CSP} = \langle\{true, false\}, \vee, \wedge, false, true\rangle$.

3 Syntax of SCLP programs

SCLP(S,D) programs are just Constraint Logic Programming (CLP) programs [JL87] where constraints are handled according to the chosen semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ and variables can be assigned values over a finite set D. As usual, a program is a set of *clauses*. Each clause is then composed by a *head* and a *body*. The head is just an atom and the body is either a collection of atoms, or a value of the semiring. Finally, a *goal* is a collection of atoms. The BNF for this syntax follows.

$P :: CL \mid CL, P$
$CL :: H : -B$
$H :: AT$ where AT is the category of atoms
$LAT :: \Box \mid LAT, AT$
$B :: LAT \mid a$ where $a \in A$
$G :: \quad : -LAT$

As an example, consider the following SCLP(S,D) program where $S = \langle -N \cup \{-\infty\}, max, +, -\infty, 0\rangle$, $-N$ is the set of non-positive integers, and $D = \{a, b, c\}$.

```
s(x)   :- p(x,y).
p(a,b) :- q(a).
p(a,c) :- r(a).
q(a)   :- t(a).
t(a)   :- -2.
r(a)   :- -3.
```

The intuitive meaning of a semiring value like -3 associated to the atom $p(a)$ is that $p(a)$ costs 3 units. Note that the ordering $\leq_S$ in this semiring coincides with the $\leq$ ordering over integers.

4 Model-theoretic semantics

An interpretation is a function which takes a predicate and an instantiation of its arguments (that is, a ground atom), and returns an element of the semiring: $I : \bigcup_n (P_n \to (D^n \to A))$, where P_n is the set of n-ary predicates. The notion of interpretation can be used to associate elements of the semiring also to formulas which are more complex than ground atoms. More precisely, we can define the function $value_I$ which takes any formula appearing in a program (not a clause) and returns an element of the semiring:

- The value associated to a non-ground atom of the form $F = \exists x.A(x)$ is computed by considering the sup of the values associated to all the ground atoms $A(x/d)$, where d is any domain element. That is, $value_I(F) = sup\{I(A(d)), \text{for all } d \in D\}$. Formulas of this kind occur in SCLP languages since variables appearing in the body of a clause but not in its head are considered to be existentially quantified. For example, in the special case of logic programming the clause `p(a) :- q(x,a)` is just a shorthand for the formula $p(a) \leftarrow \exists x.q(a, x)$.
- The value associated to a conjunction of atomic formulas of the form $A \wedge B$ is the product of the values associated to A and B: $value_I(A \wedge B) = value_I(A) \times value_I(B)$.
- For any semiring element a, $value_I(a) = a$.

Note that the meaning associated to formulas by function $value_I$ coincides with the usual logic programming interpretation [Llo93] when considering the semiring $S_{CSP} = \langle\{true, false\}, \vee, \wedge, false, true\rangle$. In fact, in this case the ordering $\leq_S$ is defined by $false \leq_S true$, the sup operation of the lattice $\langle\{true, false\}, \leq_S\rangle$ is $\vee$, and the inf is $\wedge$. Thus, for example, $value_I(\exists x.A(x)) = sup\{I(A(d)), \text{for all } d \in D\} = \vee\{I(A(d)), \text{for all } d \in D\}$. Thus it is enough that one of the $A(d)$ is assigned the value *true* that the value associated to the whole formula $\exists x.A(x)$ is *true*. Note also that in this special instance the sup and inf of the lattice coincide with the two semiring operations, but this is not true in general (see Section 2).

Given a clause of the form $H : -B$ and an interpretation I, we say that the clause is *satisfied* in I if and only if $value_I(H) \geq value_I(B)$. This is consistent with the usual treatment of clauses in logic programming, where a clause is considered to be satisfied w.r.t. an interpretation if the body logically implies the head, and by noting that logical implication in the semiring S_{CSP} coincides with the ordering $\leq_{S_{CSP}}$.

For example, the clause `p(a) :- q(b)` is satisfied in I if $I(p(a)) \geq I(q(b))$; the clause `p(x) :- q(x,a)` is satisfied if $\forall x.(I(p(x)) \geq I(q(x, a)))$; the clause `p(a) :- q(x,a)` is satisfied if $I(p(a)) \geq value_I(\exists x.q(x, a))$; the clause `p(x) :- q(x,y)` is satisfied if $\forall x.(I(p(x)) \geq value_I(\exists y.q(x, y)))$.

An interpretation is a *model* for a program P if all clauses of P are satisfied in I. Given a program and all its models, one would like to identify a unique single model as the representative one. In logic programming this is done by considering the minimal model [Llo93], which is obtained by intersecting all the models of the program. This works because models in logic programming are assimilable to sets of ground atoms, those with associated value *true*. Here we follow the same approach, but we have to generalize the notion of intersection of two models, written as "$\circ$", as their inf in the lattice $\langle A, \leq\rangle$.

Definition 4 (model intersection) *For every ground atomic formula A and two models I_1 and I_2, $I_1 \circ I_2(A) = inf(I_1(A), I_2(A))$.* $\Box$

Theorem 5 (model intersection) *Consider two models M_1 and M_2 for a CLP(S,D) program P. Then $M_1 \circ M_2$ is a model for P.*

Proof: Since M_i is a model for P for $i = 1, 2$, it must be that, for every clause $H : -B$, $M_i(B) \leq M_i(H)$. Consider now the model $M = M_1 \circ M_2$. We need to prove that, for all $H : -B$, also $M(B) \leq M(H)$ holds. Without loss of generality, assume that $B = A_1, A_2$. Thus $M_i(B) = M_i(A_1) \times M_i(A_2)$ for $i = 1, 2$ and $M(B) = M(A_1) \times M(A_2) =$

$inf(M_1(A_1), M_2(A_1)) \times inf(M_1(A_2), M_2(A_2))$. Also, $M(H) = inf(M_1(H), M_2(H))$.

Since $inf(M_1(A_1), M_2(A_1)) \leq M_1(A_1)$ and $inf(M_1(A_2), M_2(A_2)) \leq M_1(A_2)$ by definition of inf, and recalling that $\times$ is monotone, we have that $M(B) \leq M_1(B)$. The same reasoning applies also for proving that $M(B) \leq M_2(B)$.

By transitivity of $\leq$, we thus get $M(B) \leq M_1(H)$ and $M(B) \leq M_2(H)$. Since $M(H)$ is the inf of $M_1(H)$ and $M_2(H)$, and since the inf of two elements if the greatest among the elements which are smaller than both, we have that $M(B) \leq M(H)$. $\square$

It is easy to see that the operation of model intersection is associative, idempotent, and commutative. Thus its application can be extended to more then two models. Given a set of models MS, we will write $\circ(MS)$ as the model obtained by intersecting all models in MS.

Given a program P and the set of all its models, its *minimal model* is obtained by intersecting all models: $MM_P = \circ(\{M \mid M \text{ is a model for } P\})$. The *model-theoretic semantics* of a program P is its minimal model, MM_P.

As an example, consider the program P described at the end of last section. The minimal model for such a program must assign a non-positive integer to each ground atom. In this case, the minimal model MM_P is a function defined as follows:
$MM_P(t(a)) = -2$
$MM_P(q(a)) = -2$
$MM_P(r(a)) = -3$
$MM_P(p(a, c)) = -3$
$MM_P(p(a, b)) = -2$
$MM_P(s(a)) = max(-2, -3) = -2$
For each other atom different from the ones considered above, MM_P returns $-\infty$.

5 Fixpoint semantics

We define now the operator T_P which extends the one used in logic programming [Llo93] by following the same approach as in the above definition of interpretations and models. The resulting operator maps interpretations into interpretations, that is, $T_P : IS_P \rightarrow IS_P$, where IS_P is the set of all interpretations for P. Given an interpretation I and a ground atom A, assume that program P contains k clauses defining the predicate in A. Clause i is of the form $A : -B_1^i, \ldots, B_{n_i}^i$. Then

$$T_P(I)(A) = \textstyle\sum_{i=1}^{k}(\prod_{j=1}^{n_i} I(B_j^i)).$$

This function coincides with the usual immediate consequence operator of logic programming when considering the semiring $S_{CSP} = \langle\{true, false\}, \vee, \wedge, false, true\rangle$.

Consider now an ordering $\preceq$ among interpretations which respects the semiring ordering.

Definition 6 (partial order of interpretations) *Given a program P and the set of all its interpretations IS_P, we define the structure $\langle IS, \preceq\rangle$, where for any $I_1, I_2 \in IS$, $I_1 \preceq I_2$ if $I_1(A) \leq_S I_2(A)$ for any ground atom A.* $\square$

It is easy to see that $\langle IS, \preceq\rangle$ is a complete partial order, whose greatest lower bound coincides with the inf operation (suitable extended to interpretations). It is also possible to prove that function T_P is monotone and continuous over the complete partial order $\langle IS, \preceq\rangle$.

By using these properties, classical results on partial orders [Tar55] allow us to conclude that

- T_P has a least fixpoint, $lfp(T_P)$, which coincides with $inf(\{I \mid T_P(I) \preceq I\})$;
- the least fixpoint of T_P can be obtained by computing $T_P \uparrow \omega$. This means starting the application of T_P from the bottom of the partial order of interpretations, called I_0, and then repeatedly applying T_P until a fixpoint.

Consider again the program at the end of Section 3. We recall that in this specific case the semiring is $S = \langle -N \cup \{-\infty\}, max, +, -\infty, 0\rangle$ and $D = \{a, b, c\}$. Thus function T_P is:

$$T_P(I)(A) = max\{\textstyle\sum_{j=1}^{n_1} I(B_j^1, \ldots, \sum_{j=1}^{n_k} I(B_j^k)\}.$$

In the semiring $S = \langle -N \cup \{-\infty\}, max, +, -\infty, 0\rangle$ the bottom interpretation I_0 is the interpretation which maps each semiring element into itself and each ground atom into the bottom of the lattice associated to the semiring, that is, $-\infty$. Note that we slightly abused the notation since interpretations are functions whose domain contains only ground atoms (see Section 4), while here we also included semiring elements. This simplifies the definition of I_0; however, it is possible to obtain the same result with a more complex definition of I_0 which satisfies the definition of interpretation. Given I_0, we obtain I_1 by applying function T_P above. For example, $I_1(r(a)) = -3$. Instead, $I_1(p(a, c)) = -\infty$, and $I_2(p(a, c)) = I_1(r(a)) = -3$. The following table gives the value associated by the interpretations I_i with each ground atom. Some of the atoms are not listed because each interpretation I_i gives them value $-\infty$. All interpretation I_i with $i > 4$ coincide with I_4, thus I_4 is the fixpoint of T_P.

	I_1	I_2	I_3	I_4
t(a)	-2	-2	-2	-2
r(a)	-3	-3	-3	-3
q(a)	$-\infty$	-2	-2	-2
p(a,c)	$-\infty$	-3	-3	-3
p(a,b)	$-\infty$	$-\infty$	-2	-2
s(a)	$-\infty$	$-\infty$	-3	-2
s(b)	$-\infty$	$-\infty$	$-\infty$	$-\infty$
s(c)	$-\infty$	$-\infty$	$-\infty$	$-\infty$

The most interesting case is the computation of the value associated to $s(a)$. In fact, $I_3(s(a)) = max\{I_2(p(a,a)),\ I_2(p(a,b)),\ I_2(p(a,c))\} = max\{-\infty, -\infty, -3\} = -3$. Instead, $I_4(s(a)) = max\{I_3(p(a,a)), I_3(p(a,b)),\ I_3(p(a,c))\} = max\{-\infty,\ -2, -3\} = -2$. Note that the clause `s(x) :- p(x,y)` is considered equivalent to all its instantiations. In particular, when $x = a$, we have the three clauses `s(a) :- p(a,a)`, `s(a) :- p(a,b)`, and `s(a) :- p(a,c)`. These are the clauses to consider when computing $I(s(a))$.

We will now prove that the least fixpoint of function T_P coincides with the minimal model of program P. To do that, we need an intermediate result which shows that the models of a certain program P are the solutions of the equation $T_P(I) \preceq I$.

Theorem 7 (models and T_P) *Given any interpretation I for a program P, I is a model for P if and only if $T_P(I) \preceq I$.*

Proof: Consider any ground atom H and assume there are two clauses with H as their head: $H : -B_1$ and $H : -B_2$. By definition of model, each clause $H : -B_i$ is satisfied in I. Thus $I(H) \geq I(B_i)$. Now, function T_P assigns to H the sum of the values assigned by I to B_1 and B_2, thus $T_P(I)(H) = I(B_1) + I(B_2)$. But the $+$ operation coincides with the sup of the semiring, thus any value of the semiring which is greated than both $I(B_1)$ and $I(B_2)$ is also greater than their sum. Therefore $T_P(I)(H) \leq I(H)$. A similar reasoning works also for proving that if $T_P(I)(H) \leq I(H)$ for any ground atom H then I is a model. □

Theorem 8 (model and fixpoint semantics) *Given a SCLP(S,D) program P, we have that $MM_P = lfp(T_P)$.*

Proof: By definition of minimal model, $MM_P = inf(\{I \mid I \text{ is a model for } P\})$. By Theorem 7, we get $MM_P = inf(\{I \mid T_P(I) \preceq I\})$. By the classical results cited above, this coincides with the least fixpoint of T_P. □

6 Proof-theoretic semantics

We will define here a proof-theoretic semantics based on a resolution rule, just like in CLP [JL87]. However, we need first to rewrite the program into a form which is more suitable to our semantics.

First, we rewrite each clause so that the head is an atom whose arguments are only variables. This means that we must explicitely specify the substitution that was written in the head, by inserting it in the body. That is, given a clause $p(t_1, \ldots, t_n) : -B$ we transform it into $p(x_1, \ldots, x_n) : -\langle B, \theta\rangle$ where $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$. Thus bodies now have the following form: $B1 :: \langle B, \theta\rangle$. We recall that B can be either a collection of atoms or a value of the semiring. To give a uniform representation to bodies, we can define them as triples containing a collection of atoms (possibly empty), a substitution, and a value of the semiring (possibly, $\mathbf{1}$). Thus bodies are now of the form $B2 :: \langle LAT, \theta, a\rangle$. If we have a body belonging to the syntactic category $B1$ of the form $\langle a, \theta\rangle$, we get $\langle \square, \theta, a\rangle$. If instead we have $\langle C, \theta\rangle$, where C is a collection of atoms, we get $\langle C, \theta, \mathbf{1}\rangle$. Thus clauses have now the form $CL1 :: H : -B2$. Initial goals need to be transformed as well: given a goal $G = (: -C)$, where C is a collection of ground atoms, we get the goal $G' = (: -\langle C, \varepsilon, \mathbf{1}\rangle)$. The reason why we write the empty substitution and the value $\mathbf{1}$ of the semiring is that both these elements are the unit elements w.r.t. the operations we want to perform on them, that is, composition of substitution and constraint combination.

In summary, given a SCLP(S,D) program, we get a program in an intermediate language, whose syntax is as follows:
$B2 :: \langle LAT, \theta, a\rangle$
$CL1 :: H : -B2$
$P1 :: CL1 \mid CL1, P1$
$G1 :: B2$

Consider again the example at the end of Section 3. The transformed program is then

```
s(x)   :-   ⟨ p(x,y), ε,0⟩.
p(x,y) :-   ⟨ q(a), {x=a,y=b}, 0⟩.
p(x,y) :-   ⟨ r(a), {x=a,y=c}, 0⟩.
q(x)   :-   ⟨ t(a), {x=a}, 0⟩.
t(x)   :-   ⟨□, {x=a}, -2 ⟩.
r(x)   :-   ⟨□, {x=a}, -3 ⟩.
```

Once we have transformed the given SCLP program into a program in the syntax just given, we can apply the following semantic rule. This rule defines the transitions of a nondeterministic transition system whose states are goals (according to the syntactic category G1).

If the current goal contains an atom which unifies with the head of a clause, then we can replace that atom with the body of the considered clause, performing a step similar to the resolution step in CLP. The main difference here is that we must update the third element of the goal, that is, the semiring value associated to the goal: if before the transition this value is a and the transition uses a clause whose body has value a_1, then the value associated to the new goal is $a \times a_1$. The reason for using the $\times$ operation of the semiring is that this is exactly the operation used when accumulating constraints in the SCSP framework.

$$\frac{\begin{array}{c} C = A, Cr \\ A' : -\langle C_1, \theta_1, a_1 \rangle \text{ is a clause} \\ \exists \theta' = mgu(A\theta, A'\theta_1) \end{array}}{\langle C, \theta, a \rangle \longrightarrow \langle (C_1, Cr), \theta' \circ \theta_1, a \times a_1 \rangle}$$

A *derivation* is a finite or infinite sequence of applications of the above rule. A *refutation* is a finite derivation whose final goal is of the form $\langle \Box, \theta, a \rangle$.

Let us now consider the set

$$S = \{\langle C, a \rangle \mid \langle C, \varepsilon, \mathbf{1} \rangle \Rightarrow^* \langle \Box, \theta, a \rangle \text{ for some } \theta\}$$

which contains all pairs representing all refutations for the given program. Notice that we can forget about the derivation θ accumulated during the refutations, since we assumed to always start with a ground goal. Thus θ only refers to variables introduced during the derivation.

Now we are ready to define function OS_P which, given a ground atom, returns a value of the semiring. More formally, function $OS_P : AT \to A$, where AT is the set of ground atoms and A is the semiring set, is defined as follows:

$$OS_P(C) = \sum_{a_i \mid \langle C, a_i \rangle \in S} a_i$$

Notice that, if the set of all a_i such that $\langle C, a_i \rangle$ is in S is empty, $OS_P(C)$ returns the unit element for $+$, that is, $\mathbf{0}$.

For example, by considering the goal $\langle s(a), \varepsilon, \mathbf{1} \rangle$, we get two refutations, one represented by the pair $\langle s(a), -2 \rangle$, and the other one by $\langle s(a), -3 \rangle$. Thus $OS_P(s(a)) = max(-2, -3) = -2$.

Theorem 9 (model and operational semantics) *Given a SCLP(S,D) program P, we have that $MM_P = OS_P$.*

Proof (sketch): The statement can be proved by induction on the length n of the refutations, and considering at step n the set $S_n = \{\langle C, a \rangle \mid \langle C, \varepsilon, \mathbf{1} \rangle \to^n \langle \Box, \theta, a \rangle$ and such that there is no refutation of length greater than n for $\langle C, a \rangle\}$. The proof is similar to that used in [Llo93] for logic programming, although we have to generalize because of the presence of semiring values. $\Box$

7 Conclusions

We have introduced a framework for constraint programming over semirings. This allows us to use a CLP-like language for both constraint solving and optimization. In fact, constraint systems based on semirings are able to model both classical constraint solving and more sophisticated features like uncertainty, probability, fuzzyness, and optimization. We have then given this class of languages three equivalent semantics: model-theoretic, fixpoint, and proof-theoretic, in the style of CLP programs.

For lack of space and sake of readability, in this paper we treated only the case of goals consisting of a ground atom. However, our results can be extended also to the general case of non-atomic and/or non-ground goals.

References

[BMMW89] A. Borning, M. Maher, A. Martindale, and M. Wilson. Constraint hierarchies and logic programming. In Martelli M. Levi G., editor, *Proc. 6th International Conference on Logic Programming*. MIT Press, 1989.

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

[DFP93] D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. IEEE International Conference on Fuzzy Systems*. 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)*. Springer-Verlag, LNCS 747, 1993.

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

[JL87] J. Jaffar and J.L. Lassez. Constraint logic programming. In *Proc. POPL*. ACM, 1987.

[Llo93] J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1993.

[Mac92] A.K. Mackworth. Constraint satisfaction. In Stuart C. Shapiro, editor, *Encyclopedia of AI (second edition)*, volume 1, pages 285–293. John Wiley & Sons, 1992.

[MPS97] Bamshad Mobasher, Don Pigozzi, and Giora Slutzki. Multi-valued logic programming semantics: An algebraic approach. *Theoretical Computer Science*, 1997. to appear.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.