Semiring-based Constraint Logic Programming: Syntax and Semantics

STEFANO BISTARELLI and UGO MONTANARI
University of Pisa, Italy
and
FRANCESCA ROSSI
University of Padova, Italy

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

Categories and Subject Descriptors: D.3.1 [**Programming languages**]: Formal Definitions and Theory; D.3.2 [**Programming languages**]: Language Classifications—*Constraint and logic languages*; F.3.2 [**Logics And Meanings Of Programs**]: Semantics of Programming Languages—*Denotational semantics*; *Operational semantics*; I.2.3 [**Deduction and Theorem Proving**]: Logic programming; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods; I.2.5 [**Artificial Intelligence**]: Programming Languages and Software

General Terms: Languages, Theory

Additional Key Words and Phrases: Constraint, Soft Constraints, Constraint Programming Languages, Semantics

1. INTRODUCTION

Classical constraint satisfaction problems (CSPs) [Tsang 1993; Mackworth 1992] are a very expressive and natural formalism to specify many kinds of real-life problems. However, they also have some 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 [Bistarelli et al. 1995; 1997a], which is able to model many desired features, like fuzziness [Dubois et al. 1993], probability [Fargier and Lang 1993], uncertainty,

Authors' addresses: S. Bistarelli and U. Montanari, Dipartimento di Informatica, Corso Italia 40, 56125 Pisa, Italy, e-mail: {bista,ugo}@di.unipi.it. F. Rossi, Dipartimento di Matematica Pura ed Applicata, Via Belzoni 7, 35131 Padova, Italy. e-mail: frossi@math.unipd.it.

partiality [Freuder and Wallace 1992], hierarchy [Borning et al. 1989], 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 or equal to* 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) and will also be called in this paper, more simply, *soft* constraint problems.

A similar approach, called *valued CSPs*, has been developed in [Schiex et al. 1995]. Although being simpler in its use, because the levels of preference are associated to constraints and not to tuples, it is less general than the SCSPs approach, because it requires a *total* order on these levels [Bistarelli et al. 1999].

Constraint logic programming (CLP) [Jaffar and Lassez 1987] languages extend 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 [Wallace. 1996]. 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 soft constraints. In fact, this new programming paradigm, which we will call SCLP (for Semiring-based CLP, or also Soft CLP), has the advantage of treating in a uniform way, and with the same underlying machinery, all constraints that can be seen as instances of the semiring-based approach: from optimization to satisfaction problems, from fuzzy to probabilistic, prioritized, or uncertain constraints, and also multi-criteria problems, without loosing the ability to treat and solve classical hard constraints. This leads to a high-level declarative programming formalism where real-life problems involving constraints of all these kinds can be easily modeled and solved.

In passing from CLP to SCLP languages, we will replace classical constraints with the more general SCSP constraints. By doing this, from the technical point of view, we 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, fix-point, 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. However, in practice classical branch and bound techniques can be adapted to this framework to cut some useless branches.

We also show the equivalence of the three semantics. In particular, we prove that, given the set of all refutations starting from a given goal, it is possible to derive the declarative meaning of both the existential closure of the goal and its universal closure.

We also investigate the decidability of the semantics of SCLP programs, obtaining an interesting semi-decidability result: if a goal has a semiring value greater than, or greater than or equal to, a certain value in the semiring, then we can discover this in finite time. Moreover, for SCLP programs without functions, the problem is completely decidable: the semantics of a goal can be computed in finite and bounded time. In fact, in this case we can consider only a finite number of finite and bounded-length refutations (see Section 8): infinite refutations do not bring more information due to the properties of the semiring operations. Notice that the absence of functions is obviously a restriction, however not all sources of infiniteness are taken away, since nothing is said about the semiring, which could still be infinite.

The paper is organized as follows. Section 2 recalls the main definitions and properties about semiring-based CSPs, then Section 3 defines the syntax of SCLP programs. Afterwards, sections 4, 5, and 6 provide SCLP programs with a model-theoretic, a fix-point, and an operational semantics, respectively. Then, Section 7 presents a semi-decidability result for SCLP programs, and Section 8 adds some more decidability results that hold for programs without functions. Finally, Section 9 discusses the relationship with related work, and Section 10 concludes the paper summarizing its contributions and hinting at some possible lines for future work.

A shorter version of this paper, where the operational semantics was restricted to ground goals only, programs could not contain functions, and there was no decidability result (neither for SCLP programs nor for function-free SCLP programs), has appeared in [Bistarelli et al. 1997b].

2. SEMIRING-BASED CONSTRAINTS

Here we give the basic notions about constraint solving over semirings, introduced in [Bistarelli et al. 1995; 1997a].

Definition 2.1. (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., $\mathbf{a}, \mathbf{b} \in A$ implies $\mathbf{a} + \mathbf{b} \in A$), commutative (i.e., $\mathbf{a} + \mathbf{b} = \mathbf{b} + \mathbf{a}$) and associative (i.e., $\mathbf{a} + (\mathbf{b} + \mathbf{c}) = (\mathbf{a} + \mathbf{b}) + \mathbf{c}$) operation such that $\mathbf{0}$ is its unit element (i.e., $\mathbf{a} + \mathbf{0} = \mathbf{a} = \mathbf{0} + \mathbf{a}$);

— $\times$, called the multiplicative operation, is a closed and associative operation such that $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element (i.e., $\mathbf{a} \times \mathbf{0} = \mathbf{0} = \mathbf{0} \times \mathbf{a}$);

— $\times$ distributes over $+$ (i.e., $\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = (\mathbf{a} \times \mathbf{b}) + (\mathbf{a} \times \mathbf{c})$).

We recall that the difference between semirings and rings is that semirings do not have the inverse over the additive operation. That is, in a ring, for any element a, there exists an element $-a$ such that $a + (-a) = (-a) + a = \mathbf{0}$.

Semirings are standard and well-known structures. However, to deal with semiring-based constraints we need some more properties, leading to structures that we call c-semirings (again defined in [Bistarelli et al. 1995; 1997a]), where the c stands for *constraint-based*.

Definition 2.2. (C-SEMIRING) A c-semiring is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that

—$+$ is idempotent (i.e., $\mathbf{a} \in A$ implies $\mathbf{a} + \mathbf{a} = \mathbf{a}$);

—$\times$ is commutative;

—$\mathbf{1}$ is the absorbing element of $+$.

Intuitively, the idempotence of $+$ is needed to get a partial order over the elements of the semiring (otherwise we would not have reflexivity), the commutativity of $\times$ allows us to consider sets of constraints (instead of ordered tuples), and the fact that $\mathbf{1}$ is the absorbing element of $+$ makes the element $\mathbf{1}$ the maximum element of the partial order. This is useful for our treatment, since it gives us an upper bound to all the preference levels. We recall, in fact, that the elements of the chosen semiring can be interpreted in many ways: costs, levels of preference, uncertainties, probabilities, etc.

The following is a list of properties about c-semirings which will be used in this paper. Their proof can be found in [Bistarelli et al. 1997a].

— Given any c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, the relation $\leq_S$ over A such that $\mathbf{a} \leq_S \mathbf{b}$ iff $\mathbf{a} + \mathbf{b} = \mathbf{b}$ is a partial order. The intuitive meaning of this partial order is to state when an element is better than another one: if $\mathbf{a} \leq_S \mathbf{b}$ then we mean that $\mathbf{b}$ is better than $\mathbf{a}$.

— Since $\mathbf{0}$ is the unit element of the additive operation, it is the minimum element of the ordering. Thus, for any $\mathbf{a} \in A$, we have $\mathbf{0} \leq_S \mathbf{a}$.

— 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 $\mathbf{a} \leq_S \mathbf{1}$ for all $\mathbf{a}$. Thus $\mathbf{1}$ is the maximum element of the partial ordering. This implies that the $\times$ operation is *intensive*, that is, that $\mathbf{a} \times \mathbf{b} \leq_S \mathbf{a}$. This 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 complete lattice[1]. Moreover, for any $\mathbf{a}, \mathbf{b} \in A$, we have $\mathbf{a} \vee \mathbf{b} = \mathbf{a} + \mathbf{b}$, where $\vee$ is the lub 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 glb operation of the lattice;
(3) $\langle A, \leq_S \rangle$ is a distributive lattice.

The last of the above properties might suggest to base our approach over distributive lattices instead of semirings. However, by using semirings we have the two operations, $+$ and $\times$, that are fundamental in treating constraints. Instead, by using lattices, we would have the least upper bound and the greatest lower bound: while the least upper bound always corresponds to $+$, the greatest lower bound corresponds to $\times$ only when $\times$ is idempotent. Thus by using lattices we could not have treated the cases with non-idempotent multiplicative operators.

Definition 2.3. (SCSPs) Given a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and a finite domain D, a constraint over such a c-semiring is characterized by a set of k variables and a function from D^k to A. A semiring-based constraint problem (SCSP) is a set of such constraints.

Thus, each constraint associates an element of the semiring with each tuple of values of D for the variables involved in the constraint.

It is interesting to note that classical constraints, as defined for example in [Tsang 1993; Mackworth 1992], are just semiring-based constraints 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 $S_{CSP} = \langle \{true, false\}, \vee, \wedge, false, true \rangle$. This means that a classical constraint is either satisfied (by the tuples whose associated semiring element is *true*) or violated (by the tuples whose associated semiring value is *false*), and that solving a classical constraint problem means satisfying all constraints (via the multiplicative operation, in this case the *logical and*). The additive operation (in this case *logical or*) is used when projecting a constraint set over a subset of the variables; in fact, in classical constraints this means that at least one (thus the use of the *logical or* operation) of the tuple extensions (from the subset of variables to all involved variables) satisfies the constraint set.

Fuzzy constraints [Dubois et al. 1993; Schiex et al. 1995] can instead be modeled in the SCSP framework by choosing the c-semiring: $S_{FCSP} = \langle [0, 1], max, min, 0, 1 \rangle$. In this case each tuple of values gets a value between 0 and 1, constraints are combined via the min operator, and are compared via the max operator.

3. SYNTAX OF SCLP PROGRAMS

For readers familiar with Constraint Logic Programming (CLP) programs, we can say that SCLP(S) programs (also written SCLP when the semiring is obvious or not important) are just CLP programs [Jaffar and Lassez 1987] where constraints are defined over a certain c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$.

[1]Actually, for this result to hold we must assume that there exists the sum of an infinite number of elements.

As usual, a program is a set of *clauses*. Each clause is 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, or a special symbol ($\Box$) to denote that it is empty. Clauses where the body is empty or it is just a semiring element are called *facts* and define predicates which represent constraints. When the body is empty, we interpret it as having the best semiring element (that is, $\mathbf{1}$).

Atoms are n-ary predicate symbols followed by a tuple of n *terms*. Each term is either a constant or a variable or an n-ary function symbol followed by n terms. *Ground* terms are terms without variables. Finally, a *goal* is a collection of atoms.

The BNF for this syntax follows.

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

As an example, consider the following SCLP(S) program where the semiring is $S = \langle [0,1], max, min, 0, 1\rangle$. We recall that this is the fuzzy semiring, where tuples of values are given values between 0 and 1, and constraints are combined via the min operator and compared via the max operator. Note that the ordering $\leq_S$ in this semiring coincides with the $\leq$ ordering over the reals in $[0,1]$.

Our example is a generalization of the usual n-queens problem, which can be found for example in [Hentenryck 1989]. The classical formulation requires that n queens are placed on a $n \times n$ chessboard in such a way that they do not attack each other. In our formulation, we allow also attacking queens but we give a higher preference to solutions where queens attacking each other are farther apart. Thus, if there are solutions where no queens attack each other, these solutions will remain the best ones. But there are also other solutions, and among these additional solutions the best ones are those where the queens attacking each other are as far apart as possible.

We assume here that the reader is familiar at least with the logic programming [Lloyd 1987] concepts and notation.

```
myqueens([],N).
myqueens([X|Y],N) :-
      noattack(X,Y,N), myqueens(Y,N).
noattack(X,Xs,N) :-
      noattack(X,Xs,N,1).
noattack(X,[],N,Nb).
noattack(X,[Y|Ys],N,Nb) :-
      row(X,Y,N,Nb), diag1(X,Y,N,Nb), diag2(X,Y,N,Nb),
      noattack(X,Ys,N,Nb+1).
row(X,X,N,Nb) :- Nb/N.
row(X,Y,N,Nb) :- different(X,Y).
diag1(X,X+Nb,N,Nb) :- Nb/N.
diag1(X,Y,N,Nb) :- different(Y,X+Nb).
```

```
diag2(X,X-Nb,N,Nb) :- Nb/N.
diag2(X,Y,N,Nb) :- different(Y,X-Nb).
```

In this program, the n queens are represented by a list of n variables, say `[X1,...,Xn]`, where variable `Xi` represents the queen in column i and its value represents the row index where this queen is located. This is one of the usual formulations of the n-queens problem, and it is worth to notice that by adopting this formulation we assume that different queens are in different columns.

The first predicate, `myqueens/2`, traverses the whole list of n variables and sets the constraints between any pair of queen, by using the predicate `noattack/4`, which sets the constraints between any queen and all the queens in subsequent columns. The actual constraints are set by predicates `row/4`, `diag1/4`, and `diag2/4`. In the classical (hard) formulation, these predicates basically say that different queens cannot stay in the same row, nor in the same diagonal. In our case, these constraints are made soft by accepting also situations where different queens are on the same row or diagonal. For example, for the row constraint, we give the semiring value Nb/N to two queens in the same row, where Nb is the distance between the two queens (that is, the number of columns between them) and N is the total number of columns. This means that the farther apart will the two queens be, the higher this value will be. The same reasoning holds also for the two diagonals, except that the presence of both queens X and Y on the same diagonal corresponds to having $Y = X + Nb$ or $Y = X - Nb$.

Notice that each solution of this generalized n-queens problem has a semiring value which is obtained by minimizing the semiring values of all its constraints. This comes from the choice of the fuzzy semiring, where the multiplicative operation is the min. Therefore, if a solution contains three pairs of attacking queens, each of such pairs will have a semiring value given by one of the clauses defining predicates `row/4`, `diag1/4`, or `diag2/4` (proportional to the distance between the two queens), and then the value of this solution will be the minimum among such three values. Different solutions are then ordered using the other semiring operation, which in this case is the max. Note that this same program can be used also with a different semiring, obtaining a different way to compute a solution and a different ordering. For example, we could have chosen the semiring $\{\mathcal{R} \cup +\infty, min, +, +\infty, 0\}$, where the value of each solution would have been obtained by summing the values of each attacking pair, and solutions would have been compared using the min operator.

To use this program over a specific value for N, we need to add some clauses to set the domain for the constraint variables (from 1 to N) and to define predicate `different/2`. For example, for $N = 5$, we have to write the following clauses:

```
fivequeens([X1,X2,X3,X4,X5]) :-
      domain5(X1), domain5(X2), domain5(X3), domain5(X4), domain5(X5),
      myqueens([X1,X2,X3,X4,X5],5).
domain5(a) :- 1.
```
(for all $a \in \{1, \ldots, 5\}$)
```
different(a,b) :- 1.
```
(for all $a, b \in \{1, \ldots, 5\}$ such that $a \neq b$)

We now anticipate the behaviour of this SCLP program, which obeys to the semantic development of the future sections.

— Given the goal `:- fivequeens(L).`, the program will instantiate L to be a list of 5 values, indicating the row positions of the 5 queens. Since there are solutions where queens do not attack each other, the solution returned by the program will be one of these, with semiring value 1.

— Given the goal `:- fivequeens([1,3,X3,X4,X5]).`, the program will return a position for each queen such that the first queen is in row 1 and the second one in row 3. Given this additional constraints (i.e., the positions of the first two queens), it is possible that all solutions will have some attacking pairs of queens. Then, among these possible solutions, the program will return one with the highest level of preference, which means that the attacking queens are as far apart as possible.

— Assume to delete one or more of the facts defining predicate `domain5`. Then, if we give the goal `:- fivequeens(L).`, it means that we have five queens (thus five columns) but a a smaller number of domain elements (thus rows). Even in this case, it is possible that all solutions will have some attacking pairs of queens. Thus the program will return one where the attacking queens are as far apart as possible.

While this example of an SCLP program uses semiring values to be able to find the best quasi-solutions in an otherwise over-constrained problem, other examples exploit the power of the chosen semiring to express features which are intrinsic to the considered problem. Consider the problem that can arise when a client in a restaurant wants to select items from the menu in a way that his/her preferences over the combinations of drinks and dishes are satisfied in the best way. In this example, each combination (for example, beer and pizza, or white wine and fish) is associated to a level of preference, according to the taste of the client. Then, solving the problem means finding the menu with the highest level of preference. As in the previous example, here the semiring that is used is the fuzzy one, but its role is definitely different. This menu problem is one of those that have been used to show the expressive power of the clp(fd,S) system [Georget and Codognet 1998], a very general implementation of SCLP programming.

Notice that, by just changing the semiring, but maintaining the same program structure of the menu example, we could model rather different situations. Consider for example the situation in which a conference organizer has to decide the menu for the conference dinner, trying to satisfy the participants as much as possible. Of course the organizer cannot ask for their preferences, but can reason with probabilities. Therefore, he/she can associate, to each combination of drink and dish, the estimated probability that it will please the conference attendees. Then, solving the problem means finding the menu which has the highest probability to please the participants. To model this situation, it is enough to keep the same program above (modulo the new semiring values), but choose the semiring $\langle [0,1], max, \times, 0, 1\rangle$, which can represent probabilities combined via $\times$ (assuming their independence) and compared via the max operator.

4. MODEL-THEORETIC SEMANTICS

In this section we will generalize the usual development of the model-theoretic semantics in logic programming [Lloyd 1987] to be able to deal correctly with semiring values. The main generalization will involve the assignment of semiring values to atoms and formulas, instead of truth values. As usual, we will just consider Herbrand interpretations in the following, which however we will call just interpretations for sake of conciseness.

Definition 4.1. (PRE-INTERPRETATION) A pre-interpretation maps each ground term in a program into a chosen domain. More precisely, it consists of a domain D plus a mapping from each constant to an element of D and, for each n-ary function, a mapping from D^n to D.

Notice that the domain D of the pre-interpretations contains the domain of the constraints, as defined in Definition 2.3.

Definition 4.2. (INTERPRETATION) An interpretation I is a pre-interpretation plus a function which takes a predicate and an instantiation of its arguments (that is, a ground atom), and returns an element of the semiring:

$$I : \bigcup_n (P_n \rightarrow (D^n \rightarrow A)),$$

where P_n is the set of n-ary predicates.

This notion of interpretation can now be extended and used to associate elements of the semiring also to formulas which are more complex than ground atoms. In the following, this extension of an interpretation I will be called an *interpretation* and denoted by I as well, since the extension is uniquely determined. When we will want to consider the restriction of an interpretation I to ground atoms we will sometimes write $GA(I)$.

— The value associated to a formula of the form $F = \exists x.F'(x)$ is computed by considering the lub of the values associated to all ground formulas $F'(x/d)$, where d is any domain element. That is, $I(F) = lub\{I(F'(d))$, 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 $p(a) \leftarrow \exists x.q(x,a)$.

— The value associated to a formula of the form $F = \forall x.F'(x)$ is computed by considering the greatest lower bound (glb) of the values associated to all the ground formulas $F'(x/d)$, where d is any domain element. That is, $I(F) = glb\{I(F'(d))$,for all $d \in D\}$. Formulas of this kind occur when a variable appears in the head of a clause. In fact, for example, in logic programming, a clause like `p(X) :- q(X,a)` is a shorthand for the formula $\forall x.(p(x) \leftarrow q(x,a))$.

— The value associated to a conjunction of atomic formulas of the form (A, B) is the semiring product of the values associated to A and B: $I(A, B) = I(A) \times I(B)$. Such formulas appear in the body of the clauses, when the body contains more than one atom.

— For any semiring element a, $I(a) = a$. Such elements appear in the body of the facts.

Note that the meaning associated to formulas by function I coincides with the usual logic programming interpretation [Lloyd 1987] when considering constraints over 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 lub operation of the lattice $\langle \{true, false\}, \leq_S \rangle$ is $\vee$, and the glb is $\wedge$. Thus, for example, $I(\exists x.A(x)) = lub\{I(A(d)), \text{ for all } d \in D\} = \vee\{I(A(d)), \text{ for all } d \in D\}$. 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 lub and glb of the lattice coincide with the two semiring operations, but this is not true in general for the multiplicative operation (see Section 2).

Definition 4.3. (CLAUSE SATISFACTION) 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, for any ground instantiation of H, say $H\theta$, we have that $I(H\theta) \geq_S I(\exists B\theta)$.

Note that the existential quantification over the body $B\theta$ is needed, since there may be variables in B which do not appear in H. Thus $B\theta$ could be not ground. This definition of clause satisfiability is consistent with the usual treatment of clauses in logic programming, where a clause is considered to be satisfied if the body logically implies the head, and by noting that logical implication in the semiring S_{CSP} coincides with the ordering $\leq_{S_{CSP}}$.

Our running example. As our running example in this paper, we will use the following program, which is not as expressive as the program in the previous section but is simple enough to be analyzed in detail. This is an SCLP(S) program over the semiring $S = \langle N \cup \{+\infty\}, min, +, +\infty, 0 \rangle$, where N is the set of non-negative integers. This semiring allows to model constraint optimization problems where each tuple of values is assigned an integer, to be interpreted as its cost, constraints are combined by summing their costs, and are compared by using the min operator. Note that the ordering $\leq_S$ in this semiring coincides with the $\geq$ ordering over integers.

```
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.
```

In this program, the constraints are represented by predicates `t` and `r`. The intuitive meaning of a semiring value like 3 associated to the atom $r(a)$ is that $r(a)$ costs 3 units. Thus the set $N \cup \{+\infty\}$ contains all possible costs, and the choice of the two operations min and $+$ implies that we intend to minimize the sum of the costs. This gives us the possibility to select the atom instantiation which gives the minimal cost overall. □

As an example of clause satisfiability, consider the following four clauses:

—the clause `p(a) :- q(b)` is satisfied in I if $I(p(a)) \geq_S I(q(b))$;
—the clause `p(X) :- q(X,a)` is satisfied if $\forall x.(I(p(x)) \geq_S I(q(x, a)))$;
—the clause `p(a) :- q(X,a)` is satisfied if $I(p(a)) \geq_S I(\exists x.q(x, a))$;
—the clause `p(X) :- q(X,Y)` is satisfied if $\forall x.(I(p(x)) \geq_S I(\exists y.q(x, y)))$.

As in logic programming, an interpretation I 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 [Lloyd 1987], 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 greatest lower bound in the lattice $\langle A, \leq_S \rangle$.

Definition 4.4. (MODEL INTERSECTION) Consider an SCLP program over the c-semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, the corresponding ordering $\leq_S$ and the lattice $\langle A, \leq_S \rangle$. For every ground atomic formula F and a family of models $\{M_i\}_{i \in I}$, we define $\circ_{i \in I} M_i(F) = glb_{i \in I}\{M_i(F)\}$, where glb is the greatest lower bound over the lattice $\langle A, \leq_S \rangle$.

THEOREM 4.1. (MODEL INTERSECTION) *Consider a family of models $\{M_i\}_{i \in I}$ for a CLP(S,D) program P. Then $\circ_{i \in I} M_i$ is a model for P as well.*

PROOF. Since M_i is a model for P for all $i \in I$, it must be that, for every clause $H : -B$, and for all θ such that $H\theta$ is ground, $M_i(\exists B\theta) \leq_S M_i(H\theta)$. Consider now the model $M = \circ_{i \in I} M_i$. We need to prove that, for all $H : -B$, and for all θ such that $H\theta$ is ground, also $M(\exists B\theta) \leq_S M(H\theta)$ holds.

Without loss of generality, assume that $B = A_1, A_2$. Thus, for all $i \in I$, $M_i(\exists B\theta) = lub\{M_i(B\theta\theta')$, for all θ' such that $B\theta\theta'$ is ground$\} = lub\{M_i(A_1\theta\theta') \times M_i(A_2\theta\theta')$, for all θ' such that $B\theta\theta'$ is ground$\}$. Moreover, $M(\exists B\theta) = lub\{M(B\theta\theta')$, for all θ' such that $B\theta\theta'$ is ground$\} = lub\{M(A_1\theta\theta') \times M(A_2\theta\theta')$, for all θ' such that $B\theta\theta'$ is ground$\}$= $lub\{glb_{i \in I}\{M_i(A_1\theta\theta')\} \times glb_{i \in I}\{M_i(A_2\theta\theta')\}$ for all $\theta'\}$. Also, $M(H\theta) = glb_{i \in I}\{M_i(H\theta)\}$.

Consider any model M_j with $j \in I$. Since $glb_{i \in I}\{M_i(A_1\theta\theta')\} \leq_S M_j(A_1\theta\theta')$ and $glb_{i \in I}\{M_i(A_2\theta\theta')\} \leq_S M_j(A_2\theta\theta')$ by definition of glb, and recalling that $\times$ is monotone, we have that $M(\exists B\theta) \leq_S M_j(\exists B\theta)$. By transitivity of $\leq_S$, we thus get $M(\exists B\theta) \leq_S M_j(H\theta)$. Since $M(H\theta)$ is the glb of all $M_i(H\theta)$ for $i \in I$, and since the glb of a set of elements if the greatest among the elements which are smaller than all of them, we have that $M(\exists B\theta) \leq_S M(H\theta)$. $\square$

It is easy to see that the operation of model intersection is associative, idempotent, and commutative.

Definition 4.5. (MINIMAL MODEL) Given a program P and the set of all its models, its *minimal model* is obtained by intersecting all models: $M_P = \circ(\{M \mid M$ is a model for $P\})$. The *model-theoretic semantics* of a program P is its minimal model, M_P.

Consider our running example program P. The minimal model M_P for such a program must assign an integer to each formula, and when restricted to ground atoms it is the following function: $M_P(t(a)) = 2$, $M_P(q(a)) = 2$, $M_P(r(a)) = 3$, $M_P(p(a, c)) = 3$, $M_P(p(a, b)) = 2$, $M_P(s(a)) = min(2, 3) = 2$. For each atom different from the ones considered above, M_P returns $+\infty$. To explain why function M_P returns these values, we give some examples:

— Any model must assign to $t(a)$ a semiring value smaller (that is, better) than 2, because of the clause `t(a) :- 2`. Since M_P is the minimal model, it must assign to $t(a)$ the glb (that is, the max) of all such values, that is, 2.

— The value assigned to $p(a,c)$ must be smaller than that of $r(a)$, which in turn must be smaller than 3. Being in the minimal model, we have that the value of $r(a)$ is exactly 3, and the value of $p(a,c)$ is again 3. The same reasoning holds also for $p(a,b)$, whose value is 2. Instead, $p(a,v)$, for any $v \neq b, c$, gets the value $+\infty$, because a model can give any value to $p(a,v)$, since there is no clause about it, and thus the minimal model gives to it the glb (that is, the max) of all values, that is, the worst element of the semiring.

— For $s(a)$, we know that every model must assign to it a value smaller than the value assigned to $\exists y.p(a,y)$. Now, for any model M, $M(\exists y.p(a,y))$ is the lub (that is, the min) of all the values assigned by M to $p(a,v)$ for any v in the domain. We know that $p(a,b)$ has value 2, $p(a,c)$ has value 3, and any $p(a,v)$, with $v \neq b, c$, has value $+\infty$. Therefore the lub of all such values is 2. Thus any model must assign to $s(a)$ a value smaller than 2, and the minimal model M_P must give it value 2.

For the same program, it is also useful to notice which semiring value is assigned to formulas like $\forall y.p(a,y)$ by the function M_P. In fact, this is one of the kinds of formulas we will consider when studying the relationship between the operational and the model-theoretic semantics, in Section 6. By definition, to get $M_P(\forall y.p(a,y))$ we must compute the glb of all the semiring values assigned by M_P to the ground atoms of the form $p(a,v)$ where v is any element of the domain D. We know, by the paragraph above, that $M_P(p(a,c)) = 3$, $M_P(p(a,b)) = 2$, and $M_P(p(a,v)) = +\infty$ if v is different from both b and c. Thus the glb (that is, the max) of all these values is $+\infty$. Therefore $M_P(\forall y.p(a,y)) = +\infty$.

5. FIX-POINT SEMANTICS

In order to describe the fix-point semantics, we need to define the operator T_P which extends the one used in logic programming [Lloyd 1987]. We will do that by following the same approach as in the previous section. 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.

Definition 5.1. (T_P OPERATOR) 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 (see [Lloyd 1987]) when considering the semiring S_{CSP}.

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

Definition 5.2. (PARTIAL ORDER OF INTERPRETATIONS) Given a program P and the set of all its interpretations IS_P, we define the structure $\langle IS_P, \preceq \rangle$, where for any $I_1, I_2 \in IS_P$, $I_1 \preceq I_2$ if $I_1(A) \leq_S I_2(A)$ for any ground atom A.

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

By using these properties, classical results on partial orders [Tarski 1955] allow us to conclude the following:

— T_P has a least fix-point, $lfp(T_P)$, which coincides with $glb(\{I \mid T_P(I) \preceq I\})$;

— the least fix-point 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 fix-point.

Consider again our running example program. We recall that in this specific case the semiring is $S = \langle N \cup \{+\infty\}, min, +, +\infty, 0\rangle$ and $D = \{a, b, c\}$. Thus function T_P is:

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

In this semiring 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 interpretation I_i to each ground atom. Some of the atoms[2] 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 fix-point 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, $I_3(s(a)) = min\{I_2(p(a, a)), I_2(p(a, b)), I_2(p(a, c))\} = min\{+\infty, +\infty, 3\} = 3$. Instead, $I_4(s(a)) = min\{I_3(p(a, a)), I_3(p(a, b)), I_3(p(a, c))\} = min\{+\infty, 2, 3\} = 2$. 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 $I(s(a))$.

[2]Actually, an infinite number of them.

We will now prove that the least fix-point of function T_P coincides with the minimal model of program P, when restricted to ground atoms. To do that, we need an intermediate result which shows that the ground models of a given program P are the solutions of the equation $T_P(I) \preceq I$.

THEOREM 5.1. (MODELS AND T_P) *Given any interpretation I for a program P, I is a ground 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 $I(H) \geq_S I(\exists B_i)$. Now, function T_P assigns to H the sum of the values assigned by I to $\exists B_1$ and $\exists B_2$, thus $T_P(I)(H) = I(\exists B_1) + I(\exists B_2)$. But the $+$ operation coincides with the lub of the semiring, thus any value of the semiring which is greater than both $I(\exists B_1)$ and $I(\exists B_2)$ 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. $\Box$

THEOREM 5.2. (GROUND MODELS AND FIX-POINT SEMANTICS) *Given an SCLP program P, we have that $GA(M_P) = lfp(T_P)$*[3].

PROOF. By definition of ground minimal model, $GA(M_P) = glb(\{I \mid I \text{ is a model for } P\})$. By the previous theorem, we get $GA(M_P) = glb(\{I \mid T_P(I) \preceq I\})$. By the classical results cited above [Tarski 1955], this coincides with the least fix-point of T_P. $\Box$

Notice that the result of Theorem 4.1 (that is, the glb of two models is a model) can be proven also by using Theorem 5.1 (which states that a model is a prefixpoint of T_P) and by easily showing that the glb of two prefixpoints of a monotone operator (as T_P is) is a prefixpoint as well.

6. PROOF-THEORETIC SEMANTICS

We will define here a proof-theoretic semantics in the style of CLP [Jaffar and Lassez 1987]. However, we first rewrite the program into a form which allows us to make the semantics treatment more uniform and simpler.

First, we rewrite each clause so that the head is an atom whose arguments are different variables. This means that we must explicitly specify the substitution that was written in the head, by inserting it in the body. That is, given a clause of the form $p(t_1, \ldots, t_n) :- B$, 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 syntax: $B1 :: \langle B, \theta \rangle$. 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 $B2 :: \langle LAT, \theta, \mathbf{a} \rangle$. If we have a body belonging to the syntactic category $B1$ of the form $\langle \mathbf{a}, \theta \rangle$, we get $\langle \Box, \theta, \mathbf{a} \rangle$. 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 syntax $CL1 :: H :- B2$.

Initial goals need to be transformed as well: given a goal $G = (:- C)$, where C is a collection of atoms, we get the goal $G' = (:- \langle C, \varepsilon, \mathbf{1} \rangle)$. The reason why we

[3]We recall that, for any interpretation I, $GA(I)$ is the restriction of I to ground atoms.

write the value **1** of the semiring is that this element is the unit element w.r.t. the operation we want to perform on it, that is, constraint combination.

In summary, given a SCLP program, we get a program in an intermediate language, whose syntax is as follows:

$B2 :: \ \langle LAT, \theta, \mathbf{a} \rangle$
$CL1 :: H : -B2$
$P1 :: \ CL1 \mid CL1, P1$
$G1 :: \ B2$

Consider again our running example. 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).

$$\frac{\begin{array}{c} C = A, Cr \\ Cl = (A' : -\langle C_1, \theta_1, \mathbf{a_1} \rangle) \text{ is a variant of a clause or a fact} \\ \theta' = mgu(A\theta, A'\theta_1) \end{array}}{\langle C, \theta, \mathbf{a} \rangle \xrightarrow{Cl, \theta'} \langle (C_1, Cr), (\theta \circ \theta' \circ \theta_1), \mathbf{a} \times \mathbf{a_1} \rangle}$$

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 $\mathbf{a_1}$, then the value associated to the new goal is $\mathbf{a} \times \mathbf{a_1}$. 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.
Notice that we must use a *variant* of the clause involved in the rule, because we need fresh variables to avoid confusion between the variables of the clause and those of the current goal.

Definition 6.1. (DERIVATIONS AND REFUTATIONS) 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 $\langle \Box, \theta, \mathbf{a} \rangle$.

Note that in this paper we give a simplified view of the solver for soft constraints, where the solver is *implemented* by some clauses in the program (see the last two clauses of our running example). These clauses have a special shape, since they are ground facts. Also, they are in a finite number, since our constraints have finite domains. Therefore, when executing such a program, the soft constraints (like `t` and `r` in our example) are *solved* using these clauses and thus they do not appear in the resulting final goal. This allows us to simplify the usual CLP operational semantic rules (like those in [Jaffar and Maher 1994]), so that, instead of having computation states made by a current goal and a store, we can just have a goal, a substitution for the variables (this would be in the store in CLP) and a semiring value. Therefore, our operational semantic rule and states are consistent with those of CLP, in the case that we have 1) finite domain variables in the constraints, 2) solver specified by program clauses (this again can be done in this straightforward way only because we have finite domains), and 3) soft constraints. While the first two points are special cases, the third is an extension of CLP. Notice also that we can have just one rewriting rule (instead of several rules, as in [Jaffar and Maher 1994]) because we first rewrite the program into a syntax which makes clauses and facts uniform. A last point to notice is that we do not test at each step if the constraint store is consistent (that is, if the semiring value collected is greater than $\mathbf{0}$) because the notion of inconsistency is not so strong in soft constraints, and we perform this check only at the end.

Definition 6.2. (COMPACT REFUTATION SET) Given an SCLP program P, its compact refutation set $S(P)$ is defined as follows:

$$S(P) = \langle C, \theta_{|_{var(C)}}, \mathbf{a}\rangle \mid \langle C, \varepsilon, \mathbf{1}\rangle \rightarrow^* \langle \Box, \theta, \mathbf{a}\rangle\}.$$

$S(P)$ contains all triples representing all refutations for the given program. Note that we only record the part of θ which involves the variables in C. Notice also that a triple $\langle C, \theta, \mathbf{a}\rangle$ may represent more than one refutation, but all these refutations start from the same goal C, build the same substitution θ, and generate the same semiring value $\mathbf{a}$. Thus we do not mind making them indistinguishable.

A property of the set $S(P)$ is that for any triple $\langle C, \theta, \mathbf{a}\rangle$ in this set, there is also the triple $\langle C\theta, \varepsilon, \mathbf{a}\rangle$. Notice that this is a generalized version of the lifting lemma of [Lloyd 1987].

THEOREM 6.1. (GOAL SPECIALIZATION) *Given an SCLP program P and the corresponding set $S(P)$, let us consider any triple $\langle C, \theta, \mathbf{a}\rangle$ in $S(P)$. Then also the triple $\langle C\theta, \varepsilon, \mathbf{a}\rangle$ is in $S(P)$.*

PROOF. Triple $\langle C, \theta, \mathbf{a}\rangle$ says that there is a refutation which starts from C, builds θ, and obtains the semiring value $\mathbf{a}$. If we start from $C\theta$, we can construct a refutation which follows exactly the same steps as the previous one, and thus obtains the same semiring value, since θ is compatible with all these steps. $\Box$

Our goal is now to study the correspondence between the operational semantics of a goal and its model-theoretic meaning. In particular, given a goal C, we will define the operational semantics of C in two different ways, to model the meaning of both $\forall C$ and $\exists C$.

6.1 Universal Closure

Among all refutations represented by the triples in $S(P)$, there are some which have the same first element, say C, and which build the empty substitution, ε, during the computation. These refutations have to be merged by the operational semantics since they represent different alternative branches for goal C which lead to possibly different semiring values and that hold for any value of the variables of C. Thus they naturally correspond to the intuitive meaning of $\forall C$.

When merging such elements of $S(P)$, the respective semiring values must be merged as well. This is done by performing the $+$ operation. Actually, we should also merge those refutations which start from different goals, say C and C', such that C' is more instantiated than C. That is, such that there exists a substitution θ such that $C' = C\theta$. In fact, the semiring value obtained from C holds also for all goals which are more instantiated than C. For example, if we have the triples $\langle p(x), \varepsilon, \mathbf{v_1} \rangle$ and $\langle p(a), \varepsilon, \mathbf{v_2} \rangle$, the first refutation says that for all values of x, $p(x)$ gets the semiring value $\mathbf{v_1}$. Thus also $p(a)$ will get this value. On the other hand, the other refutation says that $p(a)$ gets the value $\mathbf{v_2}$. Therefore the value to be assigned to $p(a)$ is the lub between $\mathbf{v_1}$ and $\mathbf{v_2}$, which is $\mathbf{v_1} + \mathbf{v_2}$ by definition of $+$. However, it is possible to show that if there is the triple $\langle C, \varepsilon, \mathbf{v} \rangle$ in $S(P)$, then there is also the triple $\langle C\theta, \varepsilon, \mathbf{v} \rangle$ for any θ. In other words, if we have a refutation starting from C, building the empty substitution and obtaining the semiring value $\mathbf{v}$, then there is also a refutation starting from any goal more instantiated than C (that is, $C\theta$), which follows the same steps as the refutation for C and thus obtains the same semiring value. Therefore, we just need to merge those refutations which start from the same goal.

THEOREM 6.2. (MORE ABOUT GOAL SPECIALIZATIONS) *Given an SCLP program P and the corresponding set $S(P)$, let us consider any triple $\langle C, \varepsilon, \mathbf{a} \rangle$ in $S(P)$. Then, for any θ, also the triple $\langle C\theta, \varepsilon, \mathbf{a} \rangle$ is in $S(P)$.*

PROOF. The refutation represented by the triple $\langle C, \varepsilon, \mathbf{a} \rangle$ does not bind any variable present in C. By starting from a goal $C\theta$, which is more instantiated than C, we can follow exactly the same steps as in the refutation for C. In fact, at each step we can use the same clause as before, since we know that such a step did not bind the variables in C. $\square$

Let us now define a function $OS1_P$ (for Operational Semantics), which, given a goal, returns a value of the semiring by looking at all refutations for that goal which build the empty substitution.

Definition 6.3. (FUNCTION $OS1_P$) Given an SCLP program P, function $OS1_P : LAT \rightarrow A$, where LAT is the set of conjunctions of atoms and A is the semiring set, is defined as follows: $OS1_P(C) = \sum_{\langle C, \varepsilon, \mathbf{a} \rangle \in S(P)} \mathbf{a}$.

Note that, when there is no triple $\langle C, \varepsilon, \mathbf{a} \rangle \in S(P)$, function $OS1_P(C)$ returns the unit element of the $+$ operation, that is, $\mathbf{0}$. This is reasonable, since the absence of such triples means that there is no refutation starting from C and builds the empty substitution.

Notice that, if $OS1_P(C) = \mathbf{a}$, not necessarily there is a refutation starting from C and obtaining the semiring value $\mathbf{a}$. In fact, as noted above, the value $\mathbf{a}$ may have been obtained by combining several refutations, which may be incomparable with respect to $\leq_S$.

Let us now consider some examples of goal refutations and their operational semantics via function $OS1$. By considering the goal $\langle s(a), \{\varepsilon\}, 0 \rangle$ in our running example, we get two refutations, which end respectively with the goals $\langle \square, \{\varepsilon\}, 2 \rangle$ and $\langle \square, \{\varepsilon\}, 3 \rangle$. Thus $S(P)$ contains the triples $\langle s(a), \{\varepsilon\}, 2 \rangle$ and $\langle s(a), \{\varepsilon\}, 3 \rangle$. Theferore $OS1_P(s(a)) = min(2, 3) = 2$. Instead, $OS1_P(s(b)) = +\infty$ (which is the bottom of the semiring), since there is no refutation for $\langle s(b), \{\varepsilon\}, 0 \rangle$. Consider now the goal $\langle s(x), \varepsilon, 0\rangle$. In this case, we get two refutations with the same final goals as above, and thus $S(P)$ contains the same elements as above. More precisely, $S(P)$ does not contain any triple of the form $\langle s(x), \varepsilon, \mathbf{a}\rangle$, and therefore $OS1_P(s(x)) = +\infty$.

A more complex example is related to the goal $\langle p(a,y), \{\varepsilon\}, 0\rangle$, which has two refutations ending with the goals $\langle \Box, \{y = b\}, 2\rangle$ and $\langle \Box, \{y = c\}, 3\rangle$, which are therefore represented in $S(P)$ by the triples $\langle p(a,y), \{y = b\}, 2\rangle$ and $\langle p(a,y), \{y = c\}, 3\rangle$. Therefore, the operational semantics of $p(a,y)$ is $OS1_P(p(a,y)) = +\infty$, even if there are refutations starting from $p(a,y)$ and obtaining a semiring value smaller than $+\infty$. Formally, this is due to the fact that such refutations all build a substitution different from ε, and thus are not considered by function $OS1_P$. Intuitively, the fact that the operational semantics assigns to $p(a,y)$ the worst semiring value even if there are refutations which lead to better values can be explained by considering that by only looking at the refutations, as the operational semantics does, there will be other domain values, say d, for which $p(a,d)$ does not hold. We will come back to this issue in the following paragraph, when we will formally compare the operational and the model-theoretic semantics. However, we can already notice that this behavior leads to the same result (that is, the same semiring value) we computed via the model-theoretic tools (that is, function M_P) at the end of Section 4 for the formula $\forall y.p(a,y)$.

We now prove formally that in general the operational semantics of SCLP programs, as just defined via function $OS1_P$, for each goal C, computes the same semiring value as the model-theoretic semantics of $\forall C$, defined in Section 4, and thus also as the fix-point semantics of Section 5 when applied to any ground instantiation of C.

However, before stating and proving the main theorem it is useful to prove two lemmas, which will be used in the proof of the theorem. The first lemma basically proves that the operational semantics of a collection of ground atoms is the same as the multiplication of the operational semantics of the single atoms, while the second one uses the fact that we have an infinite domain, but a finite number of domain elements in the program, to prove that the operational semantics of a goal coincides with the glb of the operational semantics of all its ground instances.

LEMMA 6.1. (DISTRIBUTION) *Given an SCLP program P, consider two collections of atoms C_1 and C_2. Then we have that $OS1_P(C_1, C_2) = OS1_P(C_1) \times OS1_P(C_2)$.*

PROOF. By definition of $OS1_P$, we have that $OS1_P(C_1) \times OS1_P(C_2) = (\sum_{\langle C_1,\epsilon,\mathbf{a}\rangle \in S(P)} \mathbf{a}) \times (\sum_{\langle C_2,\epsilon,\mathbf{b}\rangle \in S(P)} \mathbf{b})$. Let us assume that we have n elements of the form $\langle C_1, \epsilon, \mathbf{a}\rangle$ and m elements of the form $\langle C_2, \epsilon, \mathbf{b}\rangle$ in $S'(P)$. Therefore we will consider the semiring values $\mathbf{a_1}, \ldots, \mathbf{a_n}$ and $\mathbf{b_1}, \ldots, \mathbf{b_m}$. Notice that n and m may be infinite; however, since we work with complete lattices, the lub ($+$) and glb are defined also in the infinite case. Thus we have $OS1_P(C_1) \times OS1_P(C_2) = (\sum_{i=1,\ldots,n} \mathbf{a_i}) \times (\sum_{j=1,\ldots,m} \mathbf{b_j})$. Now, since $\times$ distributes over $+$ (by the definition of semiring), this can be rewritten as $\sum_{i=1,\ldots,n, j=1,\ldots,m} (\mathbf{a_i} \times \mathbf{b_j})$. Consider now any refutation for the goal (C_1, C_2) which does not instantiate any variable, and assume, without loss of generality, that its selection strategy first "consumes" C_1 and then C_2. Then, after C_1 has been "consumed", we have a goal of the form $\langle C_2, \epsilon, \mathbf{a_i}\rangle$. Then the refutation continues by "consuming" C_2, leading to a final goal of the form $\langle \Box, \epsilon, \mathbf{a_i} \times \mathbf{b_j} \rangle$. Now, $OS1_P(C_1, C_2)$ just sums all the semiring values of all such simple refutations, therefore we have that $OS1_P(C_1, C_2) = \sum_{i,j}(\mathbf{a_i} \times \mathbf{b_j})$, which is the same as $OS1_P(C_1) \times OS1_P(C_2)$ for what we have said before. $\Box$

LEMMA 6.2. (INFINITE DOMAIN) *Given an SCLP program P, consider any collection of atoms C. Then $OS1_P(C) = glb\{OS1_P(C\theta)$, for all θ such that $C\theta$ is ground$\}$.*

PROOF. Given a goal C, function $OS1_P$ returns the sum of all the semiring values obtained by all those refutations of C which build the empty substitution. Let us call $E(C)$ the set of such refutations. Since $C\theta$ is more instantiated than C, then the refutations of $C\theta$ will be able to use more clauses than those for C. In fact, if the head of a clause is as instantiated as a subgoal of $C\theta$ but more instantiated than the corresponding subgoal of C, then a refutation of $C\theta$ will be able to use it without building a non-empty substitution, while any refutation of C using such a clause will have to add a non-empty substitution to the substitution of the current goal. Therefore we have that $E(C) \subseteq E(C\theta)$, which, by definition of $OS1_P$ and by the fact that the $+$ operation is the lub, means that $OS1_P(C) \leq_S OS1_P(C\theta)$. Now, this holds for any θ, but it is possible to show that for some θ we actually have $OS1_P(C) = OS1_P(C\theta)$. To do that, it is enough to consider a θ which binds all variables of C to domain elements not present in P. In this way, the refutations of $C\theta$ will not be able to use more clauses than those of C. Therefore, the glb of all $OS1_P(C\theta)$ for all θ is actually equal to $OS1_P(C)$, since we know that $OS1_P(C)$ is smaller than (or equal to) all elements $OS1_P(C\theta)$ and that it is actually equal to one of them. $\Box$

THEOREM 6.3. (OPERATIONAL MEANING OF $\forall C$) *Given a SCLP program P, consider a collection of atoms C. Then we have that $M_P(\forall C) = OS1_P(C)$.*

PROOF. If $OS1_P(C) = \mathbf{a}$, it means that $S(P)$ contains triples of the form $\langle C, \varepsilon, \mathbf{a_i} \rangle$, for $i = 1, \ldots, n$, such that $\sum_{i=1,\ldots,n} \mathbf{a_i} = \mathbf{a}$. Again, we recall that n may be infinite. However, we work with complete lattices, and thus both infinite lub (that is, sum) and infinite glb are defined.

We will prove the statement of the theorem by induction on the length of the longest of the refutations corresponding to such triples.

— **base case:** If all refutations $\langle C, \varepsilon, \mathbf{a_i} \rangle$ have length 1, it means that there are n facts in the program of the form $C' : -\langle \Box, \theta_i, \mathbf{a_i} \rangle$, where C' contains only variables and $C'\theta_i$ is equally or less instantiated than C. In fact, if the head of such facts would be more instantiated than C, then we would have a substitution different from ε in the refutation.
By definition of model-theoretic semantics, $M_P(\forall C)$ is the greatest lower bound of the values given to each ground instantiation of C by function M_P. Now, consider any of such ground instantiations, say $C\theta$. Since $C'\theta_i$ is equally or less instantiated than C, it is also equally or less instantiated than $C\theta$. Therefore $M_P(C\theta) = \sum_{i=1,\ldots,n} \mathbf{a_i} + \sum_{j=1,\ldots,m} \mathbf{b_j}$, where the semiring values $\mathbf{b_j}$ are present in other m facts whose head is less instantiated than $C\theta$ but more instantiated than C. Now we must compute the glb of all such $M_P(C\theta)$ for all θ. But it is possible to see that there is a θ such that $M_P(C\theta)$ involves only the n facts with semiring values $\mathbf{a_i}$:

just choose a substitution, say θ', which binds all variables in C to domain elements not present in P. Therefore $M_P(C\theta') = \sum_{i=1,\ldots,n} \mathbf{a_i}$. This means that the glb of all $M_P(C\theta)$ is exactly $M_P(C\theta')$, that is, $\mathbf{a}$.

— **inductive case:** Let us assume that the statement of the theorem holds when the longest of the refutations of the form $\langle C, \varepsilon, \mathbf{a_i}\rangle$ has length n. Let us now consider a collection of atoms C such that the longest refutations for C have length $n+1$. We have two cases to consider: one occurs when C contains just one atom, and the other one when C has two or more atoms.

— **1st case (one atom):** The reasoning we will use in this case is very similar to that of the base case. To consider the refutation $\langle A, \varepsilon, \mathbf{a}\rangle$, let us assume to start from the goal $\langle A, \varepsilon, \mathbf{1}\rangle$, where A is an atom. Consider now the set of n clauses contained in the program P of the form $A' : -\langle B_i, \theta_i, \mathbf{0}\rangle$, for $i = 1, \ldots, n$, where B_i is a collection of atoms and A' contains only variables and the same predicate symbol as A. To be used to expand our goal, it must be that $A'\theta_i$ is equally or less instantiated than A. In fact, if the head of such clauses would be more instantiated than A, then we would have a substitution different from ε in the refutation. By definition of $OS1_P$, we have that $OS1_P(A) = \sum_{i=1,\ldots,n} OS1_P(B_i\theta_i)$.
By definition of model-theoretic semantics, $M_P(\forall A)$ is the greatest lower bound of the values given to each ground instantiation of A by function M_P. Now, consider any of such ground instantiations, say $A\theta$. Since $A'\theta_i$ is equally or less instantiated than A, it is also equally or less instantiated than $A\theta$. Therefore $M_P(A\theta) = \sum_{i=1,\ldots,n} \mathbf{a_i} + \sum_{j=1,\ldots,m} \mathbf{b_j}$, where $\mathbf{a_i} = M_P(\forall B_i\theta_i)$ for $i = 1, \ldots, n$, and $\mathbf{b_j} = M_P(\forall B_j\theta_j)$ for $j = 1, \ldots, m$, with $A' : -\langle B_j, \theta_j, \mathbf{0}\rangle$, for $j = 1, \ldots, m$ other clauses where $A'\theta_j$ is more instantiated than A but less instantiated than $A\theta$. Now we must compute the glb of all such $M_P(A\theta)$ for all θ. But it is possible to see that there is a θ such that $M_P(A\theta)$ involves only the n clauses with semiring values $\mathbf{a_i}$: just choose a substitution, say θ', which binds all variables in A to domain elements not present in P. Therefore $M_P(A\theta') = \sum_{i=1,\ldots,n} \mathbf{a_i} = \mathbf{a}$. This means that the glb of all $M_P(A\theta)$ is exactly $M_P(A\theta')$, that is, $\sum_{i=1,\ldots,n} M_P(\forall B_i\theta_i)$. Now, by inductive hypothesis, this coincides with $\sum_{i=1,\ldots,n} OS1_P(B_i\theta_i)$, which is $OS1_P(A)$ as we stated before.

— **2nd case (two or more atoms):** Let us assume that we start from the goal $\langle C, \varepsilon, \mathbf{1}\rangle$, and that $C = A, C'$. Then we have that:$M_P(\forall C) = \{$by definition of C $\}M_P(\forall(A, C')) = \{$by definition of interpretation over a universally quantified formula$\}glb_\theta\{M_P((A, C')\theta)\} = glb_\theta\{M_P((A\theta, C'\theta)\} = \{$by definition of interpretation over a conjunction of formulas$\}glb_\theta\{M_P(A\theta) \times M_P(C'\theta)\} = \{$by inductive hypothesis and by the assumption that C' is not empty$\}glb_\theta\{OS1_P(A\theta) \times OS1_P(C'\theta)\}$ $= \{$by 6.1$\}glb_\theta\{OS1_P(A\theta, C'\theta)\}$ $=glb_\theta\{OS1_P((A, C')\theta)\} = \{$by 6.2$\}OS1_P(A, C')$ $= \{$by definition of $C\}OS1_P(C)$.

□

6.2 Existential Closure

Theorem 6.3 relates the operational semantics of a goal C to the model-theoretic meaning of its universal quantification, that is, $\forall C$. We will now show that also the declarative meaning of the existential quantification of C, that is, $\exists C$, can be computed operationally. First we need to define a function $OS2_P(C)$ which combines all the refutations for C by summing all the corresponding semiring values.

Definition 6.4. (FUNCTION $OS2_P$) Given an SCLP program P, function $OS2_P : LAT \rightarrow A$, where LAT is the set of conjunctions of atoms and A is the semiring set, is defined as follows: $OS2_P(C) = \sum_{\langle C,\theta,\mathbf{a}\rangle \in S(P)} \mathbf{a}$.

An intuitive explanation of why $OS2_P$ is defined this way is the following: since we are interested in describing the operational meaning of $\exists C$, we do not care which substitution is built during a refutation starting from C. Therefore we take all refutations starting from C. Then, we choose the best among all such refutations by using the $+$ operator.

It is worth noting at this point that this definition of $OS2_P$ coincides with the following one: $\sum_{\langle C\theta,\varepsilon,\mathbf{a}\rangle \in S(P)} \mathbf{a}$. First we need the following lemma.

LEMMA 6.3. *Given an SCLP program P and a conjunction of atoms C, we have that, if $\langle C\theta, \varepsilon, \mathbf{a}\rangle$ is in $S(P)$, then there exists θ' such that $\langle C, \theta', \mathbf{a}\rangle$, with $\theta' \leq \theta$ is in $S(P)$ as well.*

PROOF. If we start a refutation from C instead of $C\theta$, we can follow the same steps as in refutation $\langle C\theta, \varepsilon, \mathbf{a}\rangle$, because we have a more general goal. The only difference is that we may need to build a substitution, say θ', different from ε. However, such a substitution will be compatible with θ since we followed the same steps. Moreover, θ' cannot be more specific than θ, otherwise the refutation starting from $C\theta$ would have built a substitution different from ε. □

THEOREM 6.4. *Given an SCLP program P and a conjunction of atoms C, we have that $\sum_{\langle C,\theta,\mathbf{a}\rangle \in S(P)} \mathbf{a} = \sum_{\langle C\theta,\varepsilon,\mathbf{a}\rangle \in S(P)} \mathbf{a}$.*

PROOF. We will prove this theorem by showing that the set S_1 of semiring values reached by refutations of the form $\langle C, \theta, \mathbf{a}\rangle$ coincides with the set S_2 of values reached by refutations of the form $\langle C\theta, \varepsilon, \mathbf{a}\rangle$. In fact, if we show this, then, since $+$ is idempotent, the two sums coincide as well.

One direction of the proof has already been proven in Theorem 6.1: if $\langle C, \theta, \mathbf{a}\rangle$ is a refutation, then $\langle C\theta, \varepsilon, \mathbf{a}\rangle$ is a refutation as well. Therefore we have that $S_1 \subseteq S_2$. Now we have to prove that $S_2 \subseteq S_1$. In general, it is not true that, if $\langle C\theta, \varepsilon, \mathbf{a}\rangle$ is a refutation, then $\langle C, \theta, \mathbf{a}\rangle$ is a refutation as well. However, by Lemma 6.3, we have that, if $\langle C\theta, \varepsilon, \mathbf{a}\rangle$ is a refutation, then there exists θ' such that $\langle C, \theta', \mathbf{a}\rangle$, with $\theta' \leq \theta$, is a refutation as well. This is enough to prove that $S_2 \subseteq S_1$, since we are proving the equality between sets of semiring values and not sets of refutations. □

Another alternative way to define $OS2_P$ is by using the definition of $OS1_P$, as follows.

THEOREM 6.5. *Given an SCLP program P and a conjunction of atoms C, we have that $OS2_P(C) = \sum_{C\theta\ ground} OS1_P(C\theta)$.*

PROOF. The statement of the theorem comes from the definition of $OS1_P$, the associativity and idempotence of $+$, and Theorem 6.2. □

Now we formally show that the result of the application of function $OS2_P$ over any goal C is a semiring value which coincides with the model-theoretic meaning of $\exists C$.

THEOREM 6.6. (OPERATIONAL MEANING OF $\exists C$) *Given an SCLP program P, consider a collection of atoms C. Then we have that $M_P(\exists C) = OS2_P(C)$.*

PROOF. By definition of interpretation, $M_P(\exists C) = lub\{M_P(C\theta)$ for all θ s.t. $C\theta$ is ground$\} = \sum_{C\theta \text{ ground}} M_P(C\theta)$. Since $C\theta$ is ground, Theorem 6.3 says that $M_P(C\theta) = OS1_P(C\theta)$. Therefore, we have that $M_P(\exists C) = \sum_{C\theta \text{ ground}} OS1_P(C\theta)$, which is exactly $OS2_P(C)$ by the Theorem 6.5. □

7. A SEMI-DECIDABILITY RESULT

In logic programming, looking for an answer of a given goal is a semi-decidable problem: if the goal is satisfiable, then a refutation for such a goal can be found in finite time; but if the goal is not satisfiable, then there are cases in which we can go on forever without detecting such an unsatisfiability.

We will now show that a similar semi-decidability result holds also for SCLP programs. More precisely, if the semantics of an SCLP goal is a semiring value greater than (or greater than or equal to) a certain semiring value $\mathbf{k}$, then we can discover this in finite time.

The main idea, as in the logic programming case, is to visit the derivation tree of the given goal in a breadth-first way. We recall that the derivation tree of a goal is the tree whose root is the given goal, each node is a state in a derivation, and each path from the root to a leaf represents a derivation for that goal. If we visit this tree in a breadth-first way, it means that after k steps (that is, after examining k levels) we have seen all refutations of length k or less, and the first k steps of all derivations/refutations longer than k.

During this visit, we construct a sum of semiring values: starting from the $\mathbf{0}$ of the semiring at the root level, when we are at level k we add to the current sum the sum of all the semiring values associated to all refutations of length k. By the fact that the sum always leads to better values, as we go on we get new values which are better than or equal to the old ones.

Now, the crucial point is that, given any semiring value representing a partial sum, even though the values better than it are infinite (because a semiring can be infinite), the number of times that this value can be improved is finite. Thus, after examining at most a finite number of refutations, we will have computed the semiring value of the goal.

To formally prove this result, we will basically define an ordering among partial sums, and we will show that such an ordering is well-founded. Any partial sum will be represented by a suitable set of semiring values, obtained from the refutations examined so far.

We first need to give some formal definitions.

Definition 7.1. (C-PRODUCTS) Let $C = \{c_i\}_{i=1,\ldots,k}$ be a finite set of elements of a semiring S. A (symbolic) product $p = \prod_{i=1,\ldots,k} c_i^{n_i}$ is called a C-product and its value in S is denoted by $[\![p]\!]$.

Informally, a C-product represents the semiring value obtained by a refutation. In fact, such a value is obtained by multiplying all the semiring values of the various clauses used by the refutation. Such values belong to a specific set: the set of all semiring values that appear in the given program, which we call C here. The exponents in the product are needed because each semiring value in C may be used several times during a refutation, and all these occurrences have to be considered, since in general the multiplicative operation is not idempotent.

Definition 7.2. (PARTIAL ORDER OF C-PRODUCTS) We define a partial ordering $\sqsubseteq$ on C-products as $\prod_{i=1,\ldots,k} c_i^{n_i} \sqsubseteq \prod_{i=1,\ldots,k} c_i^{n'_i}$ iff $n_i \geq n'_i$ for all $i = 1, \ldots, k$.

Given two C-products p and p', if $p \sqsubseteq p'$ it means that all the semiring values of C have more (or the same number of) occurrences in p than in p'. Thus, the value of p is worse than that of p' in the semiring: multiplying more items leads to worse results, by the intensivity of $\times$. Formally: $p \sqsubseteq p'$ implies $[\![p]\!] \leq_S [\![p']\!]$. Thus we can say that p is "dominated" by p'.

Definition 7.3. (SATURATION OF A SET OF C-PRODUCTS) Given a finite or countably infinite set P of C-products, its saturation $\overline{P}$ is defined as $\overline{P} = \{p \mid \exists p' \in P.p \sqsubseteq p'\}$.

By saturating a set of C-products P, we basically add to P all those other C-products which are dominated by some element in P. Notice that $[\![P]\!] = [\![\overline{P}]\!]$, where we extended the use of the semantic parenthesis $[\![]\!]$ from C-products to sets of C-products: $[\![Q]\!] = \sum_{p\in Q}[\![p]\!]$.

In our method to compute the semantics of a goal, every time we add an element to the current partial sum, such an element is a C-product. If the semiring value of this element is dominated by another one already in the partial sum, in reality the sum value does not change, because of the properties of the $+$ operation: if $a \leq_S b$ then $a + b = b$.

The main result of this section is that such a chain of partial sums has a finite number of distinct elements, and that thus after such a finite number of steps we have computed the semantics of a goal in an SCLP program. But before stating this results we need a lemma.

LEMMA 7.1. (WELL-FOUNDEDNESS) *Let $\mathcal{P}_C$ be the set of all saturated (i.e., $\overline{P} = P$) sets of C-products. This set if well-founded under the inverse proper inclusion relation $\supset$, i.e. all the chains $P_0 \subset P_1 \subset \ldots$ of elements of $\mathcal{P}_C$ are of finite length.*

PROOF. We first consider only chains where $P_j = \overline{R_j}$, with R_j finite. Also, without loss of generality, we assume $P_0 = R_0 = \emptyset$ and $R_j = R_{j-1} \cup \{p_j\}$, with of course $\forall p \in R_{j-1}$, $p_j \not\sqsubseteq p$.

We prove the property by mathematical induction on the number k of constants in C. If $k = 1$ the property is trivial, since $P_j = \{c^n \mid n \geq \overline{n_j}\}$ for some $\overline{n_j}$, and thus $P_j \subset P_{j+1}$ means $\overline{n_j} > \overline{n_{j+1}}$.

Let us now assume that the property holds for $k-1$ and prove it for k. We work by absurd, assuming that an infinite chain exists for k and constructing an infinite chain also for $k-1$. Let us now decompose p_j as $p_j = p'_j \times c_k^{n_k^j}$, with $p'_j = \prod_{i=1,\ldots,k-1} c_i^{n_i^j}$. Now, since we must have $p_h \not\sqsubseteq p_j$ when $j < h$, we either have $p'_h \not\sqsubseteq p'_j$ or $n_k^j > n_k^h$. We now use a colored graph method to help us in the proof. The nodes of the graph are the indexes of the chain, and we draw a red arc from i to h in the latter case (that is, $n_k^j > n_k^h$) or when both conditions hold, a black arc otherwise. We now construct an infinite subsequence with no red arcs, i.e. where $p'_h \not\sqsubseteq p'_j$ for $h > j$.

Notice first that there is no infinite red path: in fact our graph is acyclic and exponent n^j_k cannot be indefinitely decreased. Assume now that we have already examined a finite initial segment of the chain, and we have already constructed a set of indexes I, initially empty. We assume inductively that there is no red arc outgoing from the indexes in I to the indexes of the rest of the chain. To find a new index to add to I, let us consider the first index of the rest of the chain. If it has no outgoing red arc, we are done. Otherwise, we follow any outgoing red arc, and we repeat the above procedure until an index without outgoing red arcs is found. Since there is no infinite red path, the procedure must terminate.

Finally, it is easy to see that for every P in $\mathcal{P}_C$ we have $P = \overline{R}$, for some finite R. In fact, let $R = \{p \in P \mid \forall p' \in P.p \not\sqsubseteq p'\}$. If R were infinite (but of course countable: $R = \{p_j\}_{j=1,2,\ldots}$), the chain $\emptyset \subset \overline{\{p_1\}} \subset \overline{\{p_1, p_2\}} \subset \ldots \subset \overline{\{p_j \mid j \leq h\}} \subset \ldots$ would be infinite. □

THEOREM 7.1. (FINITE CHAINS) *Let $P = \{p_j\}_{j=1,2,\ldots}$ be a finite or countably infinite set of C-products. Then there is a natural number N such that $\sum_{j=1,\ldots,N}[\![p_j]\!] = \sum_{j=1,2,\ldots}[\![p_j]\!]$.*

PROOF. The statement is trivial if P is finite. Thus, let us assume that P is infinite. According to Lemma 7.1, the chain $Q_1 \subseteq Q_2 \subseteq \ldots \subseteq Q_n \subseteq \ldots$ with $Q_h = \overline{\{p_j \mid j \leq h\}}$, is finite, i.e. there is a natural number N such that $Q_r = Q_N$ for $r \geq N$. Since $[\![Q_h]\!] = \sum_{j=1,\ldots,h}[\![p_j]\!]$ by definition, this proves the theorem. □

Summarizing, the formal developments of this section show that, by examining the refutations of a given $SCLP$ goal while visiting the search tree in a breadth-first way, the semantics of the goal can be computed in a finite number of steps (that is, after examining a finite number of refutations). This result, together with the property that $+$ is the lub of the lattice, leads us to the following semi-decidability statement:

> It is semi-decidable to decide whether the semantics of a goal is in relation R with a certain semiring value $\mathbf{k}$, where $R \in \{>, \geq\}$.

In fact, if it is in such a relation, then the theorem of this section tells us that after a finite number of steps we have computed such semantics. Thus, after a smaller or equal number of steps the current partial sum will have value $\mathbf{k}$ or more, at which point we can stop and say that the goal has a semantics in relation R with $\mathbf{k}$. If instead it is not in such a relation, then we do not have a method to know this in finite time. However, if there were a method to semi-decide whether the semantics of a goal is in relation $not(R)$ with a certain semiring value $\mathbf{k}$, where $not(R) \in \{<, =, \leq, \not\equiv\}$, with $\not\equiv$ meaning "incompatible with", then by Post's theorem (if a property and its complement are semi-decidable, then the property is decidable) we would conclude that it is decidable to know whether the semantics of a goal is equal to a certain semiring value $\mathbf{k}$, which we know is not true for the special case of logic programming.

8. SCLP PROGRAMS WITH NO FUNCTIONS

In the previous section we have proved that, if a goal of an SCLP program has a semiring value greater than, or greater than or equal to, some $\mathbf{k}$, then it is possible to discover this in finite time. In this section we will show that, for the special class of SCLP programs with no functions, we have the additional result that once the program is fixed, the time for computing the value of any goal for this program is finite and bounded by a constant (see Theorem 8.2 later in this section). Thus the semantics of SCLP programs without functions is decidable.

This result is based on the observation that, in SCLP programs without functions, we just have to consider a finite subclass of refutations, called in the following *simple* refutations, with a bounded length. After having considered all these refutations up to that bounded length, we have finished computing the semiring value of the given goal.

Notice that, while the absence of functions in SCLP programs is obviously a restriction, the underlying semiring could in general contain an infinite number of elements. Thus this result is not so obvious as it may appear, since not all source of infiniteness are taken away.

First we define an alternative representation for refutations, which is based on trees.

Definition 8.1. (REFUTATION TREE) Given a refutation r as follows:

$$\langle A, \varepsilon, \mathbf{1}\rangle \stackrel{Cl_1,\theta'_1}{\rightarrow} \langle C_1, \theta_1, \mathbf{a_1}\rangle \stackrel{Cl_2,\theta'_2}{\rightarrow} \ldots \stackrel{Cl_n,\theta'_n}{\rightarrow} \langle \Box, \theta_n, \mathbf{a_n}\rangle$$

its *refutation tree* is a labeled tree where each node is labeled by a clause instantiation of the form $\langle Cl_i, \sigma_i\rangle$, for $i \in \{1, \ldots, n\}$, where $\sigma_i = \theta'_i$ restricted onto the variables of the head of Cl_i. More precisely, starting from a single non-labeled node, the entire tree can be built as follows:

```
for k:=1 to n do
```

Select the first non-labeled node (which is a leaf) in a depth-first visit of the current tree, and label it with $\langle Cl_k, \sigma_k\rangle$.
If Cl_k is $H : -B_1, \ldots, B_m$, attach to this node m children.

```
end-for.
```

Definition 8.2. (SIMPLE REFUTATION) Given a refutation tree, a path from the root to a leaf is called *simple* if all its nodes have different labels up to variable renaming. A refutation is a simple refutation if all paths from the root to a leaf in its refutation tree are simple.

Thus in a simple refutation it is not possible to use the same clause, instantiated in the same way, more than once on atoms which depend on each other.

We will now show that, if we delete all non-simple refutations from $S(P)$, we do not change the value computed by $OS1_P(C)$ for any C.

THEOREM 8.1. (SIMPLE REFUTATIONS ONLY) *Consider an SCLP program P with no functions and its compact refutation set $S(P)$, and let us call $S'(P)$ the subset of $S(P)$ containing all its simple refutations. Then, for any goal C, we have that $\sum_{\langle C,\varepsilon,\mathbf{a}\rangle \in S'(P)} \mathbf{a} = OS1_P(C)$.*

PROOF. Given any non-simple refutation r_1 for a goal, we can obtain a simple refutation r_2 for the same goal: just take the refutation tree of the non-simple refutation and delete the part of the tree between any two nodes with the same label up to variable renaming. It is easy to see that this new tree still represents a refutation, and that such a refutation is simple.

Now we can notice that refutation r_2 has fewer steps than r_1, and all the steps in r_2 are present in r_1. Therefore, by the extensivity of $\times$, the semiring value computed by r_2 is better than or equal to the one computed by r_1.

Now, $OS1_P(C)$ sums all the semiring values computed by all refutations in $S(P)$ which start from C and build ε. Therefore, considering what we have said before, the sum of the semiring values associated to r_1 and r_2 gives the semiring value of r_2, that is, of the simple refutation. Thus, by "forgetting" all non-simple refutations we do not change the result of the sum of all the semiring values, which is exactly $OS1_P(C)$. $\square$

We will now prove that, given a goal C, there is only a finite number of simple refutations starting from C and building the empty substitution.

THEOREM 8.2. (FINITE SET OF SIMPLE REFUTATIONS) *Given an SCLP program P with no functions and a goal C, consider the set $SR(C)$ of simple refutations starting from C and building the empty substitution. Then $SR(C)$ is finite. Moreover, each refutation in $SR(C)$ has length at most N, where $N = \sum_{i=1}^{c} b^i$, c is the number of all clauses with head instantiated in all possible ways, and b is the greatest number of atoms in the body of a clause in program P.*

PROOF. Since a program contains a finite number of domain elements and of clauses, the number of different labels in a path of a simple refutation tree is finite. Thus the number of different refutation trees for simple refutations of C is finite. Therefore $SR(C)$ is finite.

Moreover, each simple refutation tree has the property that no two labels in a path from the root to a leaf are the same. Since such labels are clauses plus head instantiations, each path is long at most as the number of all possible clause head instantiations, say c. Furthermore, the branching factor of a simple refutation tree depends on the number of atoms in the bodies of the clauses used in the refutation. Thus the number of nodes of a simple refutation tree, and thus of steps in a simple refutation, is bounded by $\sum_{i=1}^{c} b^i$. $\square$

Therefore, computing $OS1_P(C)$ involves looking at a finite number of bounded-length simple refutations. Thus, $OS1_P(C)$ can be computed in a finite number of steps.

COROLLARY 8.1. (FINITE NUMBER OF STEPS FOR $OS1_P$) *Given an SCLP program P with no functions, consider a collection of atoms C. Then we have that $OS1_P(C)$ can be computed in a finite number of steps.*

PROOF. The statement follows directly from the results of Theorem 8.1 (we can forget about the non-simple refutations) and Theorem 8.2 (the number of simple refutations is finite, and the length of each simple refutation is bounded by a constant N). In fact, we can consider only those refutations with length at most N, and among these we take only the simple refutations and we sum their semiring values. $\square$

It is also easy to prove that $OS2_P(C)$ can be computed in a finite number of steps as well. Moreover, the T_P operator need to be applied only a finite number of times before reaching the fixpoint and thus computing the fixpoint semantics, as defined in Section 5.

9. RELATED WORK

In [Fitting 1981] a bilattice structure is used to model the presence of a family of truth values in logic programming, and a fixpoint semantics for this kind of programs is given. In this approach, the meet and join operators of the bilattice are used as extensions of classical *and* and *or*, while in our approach we use such operators to model the universal and existential quantification, but we adopt a different operator (the multiplicative operation of the semiring, possibly different from the meet operator), to extend the *logical and*. The bilattice structure he uses allows to always have a negation operator, which in general we do not have. By having less properties for our structure, we can model also situations where the multiplicative operator is not idempotent, like optimization and probabilistic problems.

The approach taken in [van Emden 1986] associates a value to each clause. From such values, taken from $[0, 1]$, a value is also associated to each atom. Then, atoms are combined by using the min and max operators. Thus this kind of logic programming is similar to what we have when using the fuzzy semiring. He also gives a model-theoretic semantics, a fixpoint semantics, and an operational semantics based on game theory.

A recent approach to multi-valued logic programming [Mobasher et al. 1997] uses bilattices with two orderings to model both truth and knowledge levels. The resulting logic programming semantics is just operational and fix-point, while no model-theoretic semantics is presented. Moreover, the presence in our approach of just one ordering (modeling truth levels) is not a restriction, since the vectorization of several semirings is still a semiring (see [Bistarelli et al. 1995; 1997a]) and thus optimization based on multiple criteria can be cast in our framework as well.

From another point of view, where classical constraints are extended to have several degrees of satisfaction, a related approach is HCLP (Hierarchical CLP) [Borning et al. 1989], where each constraint has a level of importance (like *strong, weak, required*), and these levels are used to decide which constraints to satisfy. However, a constraint can only be satisfied or not, and thus HCLP is a *crisp* formalism. Moreover, their treatment is only algorithmic, and they do not provide their language with a fix-point or a model-theoretic semantics.

10. CONCLUSIONS AND FUTURE WORK

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 also more sophisticated features like uncertainty, probability, fuzziness, and optimization. We have then given this class of languages three equivalent semantics: model-theoretic, fix-point, and proof-theoretic, in the style of classical CLP programs. Finally, we have obtaind interesting decidability results for general SCLP programs and also for those SCLP programs without functions.

To make SCLP a practical programming paradigm, we plan to investigate efficient techniques to implement their operational semantics, so that one optimal solution can be found efficiently. In this respect, we plan to collaborate with the LOCO research group at INRIA-Roquencourt, which has implemented a version of our framework on top of the existing constraint language clp(fd) [Codognet and Diaz 1996]. At the URL `http://pauillac.inria.fr/~georget/clp_fds.html` there is the latest version of this system, and in [Georget and Codognet 1998] there is a description of their implementation. In this implementation, full semiring-based arc-consistency (as in [Bistarelli et al. 1997a]) is used to keep the computation states compact and to check them for inconsistencies. In general, each domain element, as well as each constraint tuple, can be assigned a semiring value. However, they also provide useful built-ins which assign just two values to each arithmetic constraint: one to the tuples which satisfy it, and the other one to those which do not satisfy it. In this way, the modeling is easier and the computation faster, and in some cases it is expressive enough. We plan to collaborate with them to reach a reasonable trade-off between efficiency, semantic elegance of the language, and user needs.

The HCLP (Hierarchical CLP) framework [Borning et al. 1989] is the closest to what we propose in this paper. We will investigate in more detail the relationship between the two frameworks and also the possibility of implementing HCLP in the SCLP framework developed over clp(fd) [Georget and Codognet 1998].

We have used SCLP programs to model and solve framework and some classical operation research problems, like shortest path problems [Bistarelli et al. 2001]. The aim is to show, on one hand, that SCLP can be a good programming and specification environment for such problems and also generalizations of them, and, on the other hand, that some classical algorithms for these problems can be used to provide an efficient alternative semantics to some classes of SCLP programs.

ACKNOWLEDGMENTS

This research has been partially supported by MURST project Tecniche Formali per Sistemi Software and MURST project TOSCA. We would also like to sincerely thank the anonymous referees, who gave many valuable suggestions that helped improve the paper.

REFERENCES

BISTARELLI, S., FARGIER, H., MONTANARI, U., ROSSI, F., SCHIEX, T., AND VERFAILLIE, G. 1999. Semiring-based CSPs and Valued CSPs: Frameworks, properties, and comparison. *CONSTRAINTS: An international journal. Kluwer 4,* 3.

BISTARELLI, S., MONTANARI, U., AND ROSSI, F. 1995. Constraint Solving over Semirings. In *Proc. IJCAI95*. Morgan Kaufman, San Francisco, CA, USA.

BISTARELLI, S., MONTANARI, U., AND ROSSI, F. 1997b. Semiring-based Constraint Logic Programming. In *Proc. IJCAI97*. Morgan Kaufman, San Francisco, CA, USA.

BISTARELLI, S., MONTANARI, U., AND ROSSI, F. 1997a. Semiring-based Constraint Solving and Optimization. *Journal of the ACM 44,* 2 (Mar), 201–236.

BISTARELLI, S., MONTANARI, U., AND ROSSI, F. 2001. Soft constraint logic programming and generalized shortest path problems. *Journal of Heuristics. Kluwer*. to appear.

BORNING, A., MAHER, M., MARTINDALE, A., AND WILSON, M. 1989. Constraint hierarchies and logic programming. In *Proc. 6th International Conference on Logic Programming*, G. Levi and M. Martelli, Eds. MIT Press, Cambridge, MA, USA, 149–164.

CODOGNET, P. AND DIAZ, D. 1996. Compiling constraints in `clp(fd)`. *The Journal of Logic Programming. Elsevier Science 27,* 3.

DUBOIS, D., FARGIER, H., AND PRADE, H. 1993. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. IEEE International Conference on Fuzzy Systems.* IEEE, Piscataway, NJ, U.S.A., 1131–1136.

FARGIER, H. AND LANG, J. 1993. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proc. European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU).* Number 747 in LNCS. Springer-Verlag, Heidelberg, Germany, 97–104.

FITTING, M. 1981. Bilattices and the semantics of logic programming. *The Journal of Logic Programming. Elsevier Science 11,* 1–2, 91–116.

FREUDER, E. AND WALLACE, R. 1992. Partial constraint satisfaction. *AI Journal. Morgan Kaufmann 58.*

GEORGET, Y. AND CODOGNET, P. 1998. Compiling semiring-based constraints with clp(fd,s). In *Proc. CP98.* Number 1520 in LNCS. Springer-Verlag, Heidelberg, Germany.

HENTENRYCK, P. V. 1989. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, MA, USA.

JAFFAR, J. AND LASSEZ, J. L. 1987. Constraint logic programming. In *Proc. POPL.* ACM, New York, NY, USA.

JAFFAR, J. AND MAHER, M. 1994. Constraint logic programming: A survey. *The Journal of Logic Programming. Elsevier Science 19 and 20.*

LLOYD, J. W. 1987. *Foundations of Logic Programming.* Springer-Verlag, Heidelberg, Germany.

MACKWORTH, A. 1992. Constraint satisfaction. In *Encyclopedia of AI (second edition)*, S. Shapiro, Ed. John Wiley & Sons, New York, NY, USA, 285–293.

MOBASHER, B., PIGOZZI, D., AND SLUTZKI, G. 1997. Multi-valued logic programming semantics: An algebraic approach. *Theoretical Computer Science. Elsevier Science 171*, 77–109.

SCHIEX, T., FARGIER, H., AND VERFAILLE, G. 1995. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. IJCAI95.* Morgan Kaufmann, San Francisco, CA, USA, 631–637.

TARSKI, A. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics. International Press 5*, 285–309.

TSANG, E. P. 1993. *Foundations of Constraint Satisfaction.* Academic Press, San Diego, CA, USA.

VAN EMDEN, M. 1986. Quantitative deduction and its fixpoint theory. *The Journal of Logic Programming. Elsevier Science 3,* 1.

WALLACE., M. 1996. Practical applications of constraint programming. *Constraints: An International Journal. Kluwer 1,* 1 and 2.

Received February 1999; July 2000; accepted December 2000