Towards a Set Oriented Calculus for Logic Programming\textsuperscript{1}

R. Caballero\textsuperscript{2} Y. García-Ruiz\textsuperscript{3} F. Sáenz-Pérez\textsuperscript{4}

Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid
Madrid, Spain

Abstract
This paper presents SOCLP (Set Oriented Calculus for Logic Programming), a proof calculus for pure Prolog programs with negation. The main difference of this calculus w.r.t. other related approaches is that it deals with the answer set of a goal as a whole, instead of considering each single answer separately. We prove that SOCLP is well defined, in the sense that, at most, one answer set can be derived from a goal, and that the derived set is correct and complete w.r.t. the logical meaning of the program. However, not all the answer sets admit a proof in SOCLP. For this reason, the paper also introduces another calculus, SOCLP\textsubscript{→}, which allows to prove finite approximations of (possibly infinite) answer sets. SOCLP\textsubscript{→} is coupled, through the inference rule for dealing with negation, with a third calculus SOCLP\textsubscript{←}, which proves whether a set is a superset of the answer set of a given goal. The paper also shows how the three calculi are related and their main properties.

Key words: Logic Programming, Semantics.

1 Introduction
The semantics of logic programs was firstly studied in the seminal paper [5], which introduced the ubiquitous immediate consequence operator $T_P$. In [4], the negation as failure rule was introduced as an effective means of deducing negative information for logic programs, and proposed the completion of a program as a description of its meaning. These and further approaches are based on SLD resolution for logic programming, as proposed by [7], a particular refinement of the resolution principle [10]. The drawback of this point

\textsuperscript{1}This work has been funded by the projects TIN2005-09207-C03-03 and S-0505/TIC/0407.
\textsuperscript{2}Email: rafa@sip.ucm.es
\textsuperscript{3}Email: ygruiz@gmail.com
\textsuperscript{4}Email: fernan@sip.ucm.es

This paper was generated using LATEX macros provided by
Electronic Notes in Theoretical Computer Science
of view is that one cannot directly reason about the meaning of a program in terms of answer sets. In particular, this is an inconvenience when using algorithmic debugging [11] for detecting missing answers, where the complete set of answers for any subgoal of the subcomputation is needed. Also, deductive databases, for which semantics is usually defined in the same way as logic programming [14], may be better described with a set oriented calculus, since the natural answer in this context is a set of tuples, as in relational databases.

This paper presents SOCLP (Set Oriented Calculus for Logic Programming), a proof calculus for pure Prolog programs with negation, which deals with the set of answers of a goal as a whole, instead of considering each single answer separately. However, we will see that the completeness of SOCLP can only be ensured for a restrictive class of logic programs, namely the hierarchical programs [8,6] over finite Herbrand universes. For this reason, the paper also introduces another calculus, SOCLP _, which establishes whether a given set is a subset of the (possibly infinite) answer set of a certain goal. Therefore, SOCLP _ allows to prove finite approximations of the answer set. SOCLP _ is coupled, through the inference rule for dealing with negation, with a third calculus SOCLP →, which proves whether a set is a superset of the answer set of a given goal.

We also include theoretical results proving properties of the three calculi. For the sake of clarity and brevity, proofs of the results are not included in this paper but can be consulted at [3].

Among these results we prove that SOCLP is well defined, in the sense that only one set can be derived from a goal, and that such set represents faithfully the answer set of a goal. We also show how the three calculi are related by proving that a set of answers can be inferred from a given goal in SOCLP iff the set can be inferred from both SOCLP _ and SOCLP →.

To the best of our knowledge, only few papers in the field of declarative debugging address calculi involving goal answer sets (e.g., [12]). However, these works do not consider programs with negation, and usually represent the sets as disjunctions of logic formulas with variables instead of dealing directly with set of ground terms. While their approach is more useful for representing the answers of Prolog systems, ours is more oriented to the case of deductive databases where the answers are assumed to be sets of ground terms.

The limitation of our approach is that, in general, infinite proofs should be needed for recursive goals with function symbols, and hence it seems no adequate for describing logic programs. But if we consider deductive databases, termination is guaranteed provided some conditions. These conditions can avoid infinite proof trees and restrict both function symbols and negation. This is the case of safe Datalog programs with stratified negation [13], where finiteness and termination is ensured.

Our paper will be organized as follows: First, a motivating section introduces our setting and highlights their advantages. Second, we pose some
preliminaries about the logic language we adhere to. The third section presents
the set oriented calculus SOCLP intended to represent goal meanings as tuple
sets. Section four introduces the two calculi SOCLP – and SOCLP – and their
properties. Finally, some conclusions and future work are pointed out.

2 Preliminaries

In this section, we present the notation and definitions used throughout the
paper which somehow differ from other approaches to logic programming. For
other basic definitions about this paradigm, we refer the reader to [1].

We consider programs with the syntax of pure Prolog programs with nega-
tion but without ”impure” features. A program \( P \) is therefore a set of normal
clauses \[ \{ \](. In order to distinguish the different clauses defining the same
predicate \( p \), we use subindices following any arbitrary criterium (such as the
textual order in the program). Thus, if a predicate \( p \) is defined through \( r \) clauses we will denote them as \( p_1, \ldots, p_r \). Each normal clause \( p_i \) must have
the form:

\[
\begin{array}{c}
\text{head}:
\hline
p_i(t_n) :- l_1(a_{k_1}^1), \ldots, l_m(a_{k_m}^m),
\end{array}
\]

\[
\text{body}
\]
corresponding to the first order logic formula \( p_i(t_n) \iff l_1(a_{k_1}^1) \land \ldots \land l_m(a_{k_m}^m), \)
where the variables whose first occurrence is on the head of the clause have
implicit universal quantifiers and the variables whose first appearance is in
the body of the clause have implicit existential quantifiers. The notation \((t_n)\)
denotes the \( n \)-ary tuple \((t_1, \ldots, t_n)\). In particular, we represent by () the 0-ary
tuple, commonly called the unit tuple. The symbols \( t_i \) (with \( 1 \leq i \leq n \)) and
\( a_u^v \) with \( 1 \leq u \leq m, 1 \leq v \leq k_u \) represent terms, defined as usual in logic
programming: any variable and constant is a term, and any expression \( f(t_n) \)
(with a function symbol \( f \) of arity \( n \) and terms \( t_i \) (\( 1 \leq i \leq n \))) is a term as
well. The set of all the tuples of \( n \) terms that can be built using the constants
and functions of a program \( P \) is denoted in the rest of the paper as \( U_n \). Notice
that \( U_n \) is infinite if the set of terms (the Herbrand universe) is infinite. The
symbols \( l_i(a_{k_i}^i) \) with \( 1 \leq i \leq m \) stand for literals which can be either positive
atoms of the form \( p(a_{k_i}^i) \) or negated atoms \( \neg p(a_{k_i}^i) \). Sometimes, we will also
be interested in extended literals which can be of the form \( p(a_{k_i}^i), p_j(a_{k_j}^j) \) and
\( \neg p(a_{k_i}^i) \). Including names of clauses as possible (extended) literals is consistent
with considering each clause \( p_i \) as a predicate defined by just one clause, and
any predicate \( p \) defined by the implicit formula:

\[
\forall X_1, \ldots, X_n \ (p(X_n) \iff p_1(X_n) \lor \cdots \lor p_m(X_n))
\]

where \( n \) is the predicate arity and \( X_j \) is a variable for every \( 1 \leq j \leq n \).

Goals will be literals \( l(t_n) \) with \((t_n) \in U_n \) and with all the variables in
the goal assumed to be existentially quantified. Although the usual defini-
tion of goals in logic programming considers conjunctions of literals, ours
Relating topics to areas of knowledge has no lack of generality; whenever we need to use a conjunction of literals $l_1(a_{k_1}^1), \ldots, l_m(a_{k_m}^m)$ as a goal, we replace it by the goal $\text{main}(\bar{X}_n)$, assuming the existence of a new predicate $\text{main}$ defined by only one clause of the form:

$$\text{main}_1(\bar{X}_n) : - l_1(a_{k_1}^1), \ldots, l_m(a_{k_m}^m)$$

where $\{X_1, \ldots, X_n\}$ is the set of variables in $l_1(a_{k_1}^1), \ldots, l_m(a_{k_m}^m)$.

**Example 2.1** Figure 1 presents a small program written following the conventions described so far. The predicate $\text{topicArea}$ relates topics to their area of knowledge. The three clauses of this predicate are facts, which are displayed in our setting by including the special propositional constant $\text{true}$ as the body of each clause. The $\text{main}$ predicate of the example holds for those values $X$ such that are topics of the field of mathematics but not of the field of computers.

An answer of a positive goal $p(\bar{t}_n)$ w.r.t. a program $\mathcal{P}$ is a tuple of terms $(\bar{a}_n)$ such that:

i) $(\bar{a}_n) \in U_n$.

ii) There exists a substitution $\theta$ verifying $(\bar{t}_n)\theta = \bar{a}_n$.

iii) $p(\bar{a}_n)$ is a logical consequence of the set of logic formulas represented by the program.

We say that $\theta$ is the associated substitution to the answer $(\bar{a}_n)$. The first condition limits our possible answers to ground terms. For instance, the only answer of the goal $\neg\text{main}(Y)$ w.r.t. the program of Figure 1 is $(\text{topology})$.

An answer for a negative goal $\neg p(\bar{t}_n)$ w.r.t. a program $\mathcal{P}$ is a tuple of terms $(\bar{a}_n)$ such that $(\bar{a}_n)$ is not an answer of $p(\bar{t}_n)$. We use the expression answer set to indicate the set containing all the answers of a given goal. For instance, the answer set of the goal $\neg\text{main}(X)$ is the set $\{(\text{logic}), (\text{maths}), (\text{computers})\}$, since these are the elements of $U_1$ that are not answers of $\text{main}(X)$.

Finally, we define a special kind of programs: we say that a program is hierarchical [8,6] if there exists some level mapping such that every clause $p_i(\bar{t}_n) : - l_1(a_{k_1}^1), \ldots, l_m(a_{k_m}^m)$ verifies that the level of every predicate occurring in the body is less than the level of $p$. A level mapping of a program is a mapping from its set of predicate symbols to the natural numbers. We call level of a predicate to the value of predicate under such mapping. For instance, the program in Figure 1 is a hierarchical program because we can define the
### SOCLP

1. **(TR)** \[ \text{true} \leftrightarrow \{()\} \]

2. **(EMP)** \[ \begin{array}{c} p_i(\overline{t}_n) \leftrightarrow \emptyset \\ \text{if } (p_i(\overline{a}_n) \leftarrow l_1(\overline{b}_{k_1}^1), \ldots, l_m(\overline{b}_{k_m}^m)) \in P, \text{ and} \\ m.g.u.((\overline{a}_n), (\overline{t}_n)) \text{ does not exist.} \end{array} \]

3. **(NEG)** \[ \begin{array}{c} p(\overline{t}_n) \leftrightarrow S_2 \\ \neg p(\overline{t}_n) \leftrightarrow S_1 \\ \text{if } S_1 = U_n \setminus S_2 \end{array} \]

4. **(PR)** \[ \begin{array}{c} p(\overline{t}_n) \leftarrow S_1 \ldots p_k(\overline{t}_n) \leftarrow S_k \\ p(\overline{t}_n) \leftrightarrow S \\ \text{if } S = \bigcup_{i=1}^k S_i, \text{ and } p_1, \ldots, p_k \text{ are all the clauses of } p \text{ in } P \end{array} \]

5. **(CL)** \[ \begin{array}{c} l_1(\overline{b}_{k_1}^1)\theta \leftarrow S_1 \ldots l_m(\overline{b}_{k_m}^m)\theta \leftarrow S_m \\ p_i(\overline{t}_n) \leftrightarrow S \\ \text{if:} \\ - S \subseteq U_n \\ - (p_i(\overline{a}_n) \leftarrow l_1(\overline{b}_{k_1}^1), \ldots, l_m(\overline{b}_{k_m}^m)) \in P \\ - \theta = m.g.u.((\overline{a}_n), (\overline{t}_n)) \\ - (\overline{t}_n)\theta \theta' \in S \text{ for all } \theta' \text{ s.t. } (\overline{b}_{k_i}^i)\theta \theta' \in S_i \text{ with } i = 1 \ldots m \\ - \text{for all } (\overline{t}_n) \in S \text{ exists } \theta' \text{ s.t. } (\overline{t}_n) = (\overline{t}_n)\theta \theta' \\ \text{and } (\overline{b}_{k_i}^i)\theta \theta' \in S_i \text{ for each } i = 1 \ldots m \end{array} \]

**Fig. 2.** The SOCLP calculus

Given the following mapping: \{ \text{true} \mapsto 0, \text{topicArea} \mapsto 1, \text{main} \mapsto 2 \}.

### 3 The SOCLP Calculus

In this section, we present the proof calculus SOCLP, which allows to prove that a set \( S \) is the answer set of a goal \( G \) w.r.t. a program \( P \). We will use the notation \( P \models_{\text{SOCLP}} G \leftrightarrow S \), with \( G \) a goal and \( S \) a set of ground terms, to indicate that the statement \( G \leftrightarrow S \) can be deduced in SOCLP using a finite number of steps. In this case, we will say that \( G \leftrightarrow S \) has a proof in SOCLP, and that \( S \) is the SOCLP-set of \( G \). The five rules of SOCLP can be seen in Figure 2. The first two inference rules correspond to the trivial cases of a goal.
being either true or \( p_i(\overline{a}_n) \) with \((\overline{a}_n)\) not unifiable with the head of any clause for \( p_i \). In the first case, the SOCLP-set only contains the unit tuple \( () \), since we assume that this 0-ary predicate always holds. In the second case, if the most general unifier of \((\overline{a}_n)\) and the tuple at the head of \( p_i \) does not exist, it is easy to check that the goal has no answer. Thus, the SOCLP-set is \( \emptyset \). The inference rule \((\text{NEG} \leftrightarrow)\) says that the SOCLP-set of a negated atom \( \neg p(\overline{t}_n) \) is the complementary of the SOCLP-set of the corresponding positive atom w.r.t. \( U_n \). That is, we use the closed world assumption \([9]\), which assumes that all atoms not entailed by a program are false. This point of view is convenient for working with answer sets, which makes it widely used in database logic languages (see for instance \([14]\), chapter 10). \((\text{PR} \leftrightarrow)\) defines the SOCLP-set of a positive atom as the union of the SOCLP-sets obtained by using its defining clauses. Finally, \((\text{CL} \leftrightarrow)\) explains how to obtain the SOCLP-set \( S \) of a clause \((p_i \overline{t}_n) : l_1(\overline{b}_{k_1}) , \ldots , l_m(\overline{b}_{k_m}) \in P \) is assumed to have new variables, different from those in \((\overline{t}_n)\). This rule says that all the premises must be affected by the most general unifier of \((\overline{a}_n)\) and \((\overline{t}_n)\). Then, each substitution \( \theta' \) that generalizes the associated substitutions of one element of the SOCLP-set of each body literal produces an element in \( S \). Conversely, each substitution associated to an element of \( S \) must correspond to the restriction of a more general \( \theta' \) that generalizes the associated substitutions of one element of the SOCLP-set of each body literal.

**Example 3.1** As an example of inference using SOCLP, Figure 3 includes a SOCLP proof tree for \( \mathcal{P} \vdash_{\text{SOCLP}} \text{main}(Y) \leftrightarrow \{(\text{topology})\} \). The root contains the initial statement, and the children of any node correspond to the premises of the SOCLP inference rule applied at the node. Below the tree are listed the renaming of the clauses and the m.g.u. associated to each application of the \((\text{CL} \leftrightarrow)\) inference rule. For instance, the first application of \((\text{CL} \leftrightarrow)\) corresponds to the clause for \( \text{main} \). The children correspond to the body of the clause after applying the m.g.u.:

\[
\text{topicArea}(Y,\text{maths}), \neg \text{topicArea}(Y,\text{computers})
\]

The SOCLP-set for the first literal \( \text{topicArea}(Y,\text{maths}) \) is \( \{\text{logic, topology}\} \), corresponding to the substitutions \( \theta' = \{Y \mapsto \text{logic}\} \) and \( \theta' = \{Y \mapsto \text{topology}\} \), respectively. The SOCLP-set of the second literal is \( \{\text{maths, computers, topology}\} \), with associated substitutions \( \theta' = \{Y \mapsto \text{maths}\}, \theta' = \{Y \mapsto \text{computers}\}, \text{and} \theta' = \{Y \mapsto \text{topology}\} \). Only the substitution \( \theta' = \{Y \mapsto \text{topology}\} \) corresponds to the SOCLP-sets of both literals, and for that reason \( \text{topology} \) (underlined in the tree) is the only element in the SOCLP-set for \( \text{main} \), following the requirements of the fourth and fifth conditions of \((\text{CL} \leftrightarrow)\).

This example shows that in SOCLP neither the order of the literals in the body of a clause nor the textual order of the clauses of the same predicate is important. The following proposition ensures that the calculus defines at most one SOCLP-set for any given goal.
Proposition 3.2 Let $\mathcal{P}$ be a program, $l(\bar{t}_n)$ a goal, and $S_a$, $S_b$ two sets such that $\mathcal{P} \vdash_{SOCLP} l(\bar{t}_n) \leftrightarrow S_a$ and $\mathcal{P} \vdash_{SOCLP} l(\bar{t}_n) \leftrightarrow S_b$. Then $S_a = S_b$.

The next theorem establishes the relationship between the SOCLP proofs and the logical meaning of the program, proving that the SOCLP-set of any goal is actually its answer set.

Theorem 3.3 (Soundness and weak completeness of SOCLP)

Let $\mathcal{P}$ be a program, $l(\bar{t}_n)$ a goal, and $S$ a set such that $\mathcal{P} \vdash_{SOCLP} l(\bar{t}_n) \leftrightarrow S$. Then $\bar{a}_n \in S$ iff $(\bar{a}_n)$ is an answer of $l(\bar{t}_n)$.

The theorem states that, whenever a SOCLP-proof for $l(\bar{t}_n) \leftrightarrow S$ exists, we can ensure that $S$ is precisely the answer set of $l(\bar{t}_n)$. Hence, we can trust the SOCLP proofs as suitable descriptions of the answer set of a goal. However, not all the answer sets admit a SOCLP proof tree.

Example 3.4 Consider, for instance, the small program:
\( \neg \text{main}(X) \leftrightarrow \{(\text{suc}(\text{zero})), (\text{suc}(\text{suc}(\text{zero}))), \ldots\} \)

\( \text{main}(X) \leftrightarrow \{(\text{zero})\} \)

\( \text{main}_1(X) \leftrightarrow \{(\text{zero})\} \)

\( \text{less}(X, \text{suc}(\text{zero})) \leftrightarrow \{(\text{zero}, \text{suc}(\text{zero}))\} \)

\( \text{less}_1(X, \text{suc}(\text{zero})) \leftrightarrow \{(\text{zero}, \text{suc}(\text{zero}))\} \)

\( \text{less}_2(X, \text{suc}(\text{zero})) \leftrightarrow \emptyset \)

\( \text{true} \leftrightarrow \{()\} \)

\( \text{less}_1(U, \text{zero}) \leftrightarrow \emptyset \)

\( \text{less}_2(U, \text{zero}) \leftrightarrow \emptyset \)

\( \text{true} \leftrightarrow \{()\} \)

\( \text{less}_1(U, \text{zero}) \leftrightarrow \emptyset \)

\( \text{less}_2(U, \text{zero}) \leftrightarrow \emptyset \)

(1) Clause \( \text{main}_1(Y) : \neg \text{less}(Y, \text{suc}(\text{zero})), \theta = \{Y \mapsto X\} \)

(2) Clause \( \text{less}_1(\text{zero}, \text{suc}(V)) : \neg \text{true}, \theta = \{X \mapsto \text{zero}, V \mapsto \text{zero}\} \)

(3) Clause \( \text{less}_2(\text{suc}(U), \text{suc}(V)) : \neg \text{less}(U, V), \theta = \{X \mapsto \text{suc}(U), V \mapsto \text{zero}\} \)

Fig. 4.

\[ p_1(a) : \neg \text{true}. \]

\[ p_2(X) : \neg p(X). \]

It is easy to check out that the answer set of the goal \( p(X) \) is \{\( (a) \}\}. However, the statement \( p(X) \leftrightarrow \{(a)\} \) cannot be proved in SOCLP.

The next proposition establishes that the answer set of a goal admit a SOCLP proof if we restrict our setting to hierarchical programs over finite Herbrand universes:

**Proposition 3.5** *(Completeness of hierarchical programs over finite Herbrand universes)*

Let \( \mathcal{P} \) be a hierarchical program over a finite Herbrand universe and \( l(\bar{t}_n) \) a goal. Then, there exists some set \( S \) such that \( \mathcal{P} \vdash_{\text{SOCLP}} \mathcal{P}(l(\bar{t}_n)) \leftrightarrow S \).

By Proposition 3.2, the set \( S \) is unique, and, by Theorem 3.3, this set is the answer set of the goal. Thus, the SOCLP calculus defines correctly the semantics of hierarchical programs [6] over finite Herbrand universes from the point of view of the answer sets. Although the proposition provides a rather restrictive set of programs for which SOCLP is complete, this does not mean that SOCLP cannot be applied to non-hierarchical programs.
Example 3.6 The predicate \texttt{main} of the following program defines the set of natural numbers which are less than one (i.e., only the number zero):
\[
\begin{align*}
\text{less}_1(\text{zero}, \text{suc}(Y)) & : - \text{true}. \\
\text{less}_2(\text{suc}(X), \text{suc}(Y)) & : - \text{less}(X, Y). \\
\text{main}_1(X) & : - \text{less}(X, \text{suc}(\text{zero})).
\end{align*}
\]
This program is not hierarchical, due to the second rule for \texttt{less}, and its Herbrand Universe is infinite. However, as Figure 4 shows, using SOCLP is possible to prove that \(\mathcal{P} \vdash_{\text{SOCLP}} \neg \text{main}(X) \iff \{(\text{suc}(\text{zero})), (\text{suc}(\text{suc}(\text{zero}))), \ldots \}\), i.e., that all the natural numbers different from \texttt{zero} are not less than \texttt{suc(zero)}.

In the previous example, it is also interesting to note that SOCLP proofs are not always limited to finite answer sets. This is due to the rule for negation, which can convert the proof of an infinite answer set in the proof of a finite answer set, and vice versa (as the first inference step of Figure 4 shows). This rule also makes the set of provable statements different from those of pure Prolog programs with negation. For instance, the previous goal \(\neg \text{main}(X)\) has no solution using negation as failure because \text{main}(X)\ does not fail (it succeeds with \(X \mapsto \text{zero}\)).

4 The \textsc{SOCLP}_\texttt{-} and \textsc{SOCLP}_\texttt{→} Calculi

The previous section presented some examples of SOCLP proofs for finite and, even, infinite answer sets. It also showed that it is very easy to find programs and goals whose answer set does not admit a SOCLP proof. Consider for instance the following program:

Example 4.1 A predicate \texttt{nat} defining the natural numbers using Peano’s axioms:
\[
\begin{align*}
\text{nat}_1(\text{zero}) & : - \text{true}. \\
\text{nat}_2(\text{suc}(X)) & : - \text{nat}(X).
\end{align*}
\]
The answer set of the goal \texttt{nat}(N)\ is the infinite set of program terms representing the set of the natural numbers \(\{(\text{zero}), (\text{suc}(\text{zero})), (\text{suc}(\text{suc}(\text{zero}))), \ldots \}\). The SOCLP calculus cannot build a proof for this set because of the recursive definition of \texttt{nat}_2.

Figure 5 introduces two new calculi, \textsc{SOCLP}_\texttt{-} and \textsc{SOCLP}_\texttt{→}, that can substitute SOCLP when we are interested in proving that a given set is included in the answer set of a given goal. The two calculi are coupled through the inference rule for negation. Given a goal \(G\) and a set of ground terms \(S\), we use the notation \(\mathcal{P} \vdash_{\text{SOCLP}_\texttt{-}} G \leftarrow S\) to indicate that there exists a finite inference for \(G \leftarrow S\) using the calculus \textsc{SOCLP}_\texttt{-}, and \(\mathcal{P} \vdash_{\text{SOCLP}_\texttt{→}} G \rightarrow S\) to indicate that there exists a finite inference for \(G \rightarrow S\) using the calculus \textsc{SOCLP}_\texttt{→}. 
\[
\begin{array}{ll}
\text{SOCLP}_- & \text{SOCLP}_-\\
\text{(TR}_-\text{)} & \text{(TR}_-\text{)} \\
\text{true} \leftarrow \{()\} & \text{true} \rightarrow \{()\} \\
\text{(EMP}_-\text{)} & \text{(EMP}_-\text{)} \\
p_i(t_n) \leftarrow \emptyset & p_i(t_n) \rightarrow S \\
\text{(NEG}_-\text{)} & \text{(NEG}_-\text{)} \\
p(t_n) \rightarrow S_2 & p(t_n) \leftarrow S \\
-p(t_n) \leftarrow S_1 & -p(t_n) \rightarrow S_1 \\
\text{(PR}_-\text{)} & \text{(PR}_-\text{)} \\
p_1(t_n) \leftarrow S_1 \ldots p_k(t_n) \leftarrow S_k & p_1(t_n) \rightarrow S \\
p(t_n) \rightarrow S & -p(t_n) \rightarrow S_1 \text{ if } S_1 \subseteq U_n \setminus S_2 \\
\text{if } S \subseteq \bigcup_{i=1 \ldots k} S_i & -p(t_n) \rightarrow S_1 \text{ if } S_1 = U_n \setminus S_2 \\
\text{(CL}_-\text{)} & \text{(CL}_-\text{)} \\
l_1(\tilde{b}_{k_1}^1)\theta \leftarrow S_1 \ldots l_m(\tilde{b}_{k_m}^m)\theta \leftarrow S_m & l_1(\tilde{b}_{k_1}^1)\theta \rightarrow S \\
p_i(t_n) \leftarrow S & p_i(t_n) \rightarrow S \\
\text{if:} & \text{if:} \\
- S \subseteq U_n & - S \subseteq U_n \\
- (p_i(\tilde{a}_n) :: l_1(\tilde{b}_{k_1}^1), \ldots, l_m(\tilde{b}_{k_m}^m)) \in P \text{ if } S \subseteq U_n \\
- \theta = \text{m.g.u.}(\tilde{a}_n), (t_n)) & - (p_i(\tilde{a}_n) :: l_1(\tilde{b}_{k_1}^1), \ldots, l_m(\tilde{b}_{k_m}^m)) \in P \text{ if } \theta = \text{m.g.u.}(\tilde{a}_n), (t_n)) \\
- \text{For all } (\tilde{t}_n) \in S, \text{ exists } \theta' \text{ s.t.} & - (\tilde{t}_n)\theta' \text{ is in } S \text{ for all } \theta' \text{ s.t.} \\
(\tilde{t}_n) = (t_n)\theta' \text{ and } & (\tilde{b}_{k_i}^i)\theta' \in S_i \text{ for } i = 1 \ldots m \\
(\tilde{b}_{k_i}^i)\theta' \in S_i & \\
\end{array}
\]

Fig. 5. The SOCLP\textunderscore and SOCLP\textunderscore Calculi

The purpose of the calculi is to prove partial approximations to the answer set of a given goal. Their inference rules are similar to the inference rules for SOCLP, but "relaxing" the associated conditions, which increase the set of provable statements. Considering again the Example 4.1, now it is possible to prove that \(P \vdash_{\text{SOCLP}_-} \text{nat}(N) \rightarrow S\), with \(S\) any finite approximation of the answer set \(\{\text{zero}, \text{suc(zero)}, \text{suc(suc(zero)),} \ldots\}\).

For instance, Figure 6 shows the proof tree proving that \(P \vdash_{\text{SOCLP}_-} \text{nat}(N) \leftarrow \{(\text{zero}), (\text{suc(suc(zero)))}\}\). The relationship between the two calculi and
SOCLP is stated in the next proposition:

Proposition 4.2 Let $P$ be a program, $l(\bar{t}_n)$ a literal, and $S$ a set of tuples of arity $n$. Then, $P \vdash_{SOCLP} l(\bar{t}_n) \leftrightarrow S$ implies $P \vdash_{SOCLP-} l(\bar{t}_n) \rightarrow S$ and $P \vdash_{SOCLP-} l(\bar{t}_n) \leftarrow S$.

That is, the existence of a SOCLP proof for $l(\bar{t}_n) \leftrightarrow S$ determines the existence of both a SOCLP- proof for $l(\bar{t}_n) \leftarrow S$, and a SOCLP- proof for $l(\bar{t}_n) \rightarrow S$. But the main goal of SOCLP- and SOCLP- is not proving the answer set of a goal, but approximations to this answer set. The next proposition states that the two calculi are also useful for this purpose:

Proposition 4.3 Let $l(\bar{t}_n)$ be a literal and $S$ a set such that $P \vdash_{SOCLP} l(\bar{t}_n) \leftrightarrow S$. Then, the two following statements hold:

i) $S' \subseteq S$ for every $S'$ such that $P \vdash_{SOCLP-} l(\bar{t}_n) \leftarrow S'$

ii) $S' \supseteq S$ for every $S'$ such that $P \vdash_{SOCLP-} l(\bar{t}_n) \rightarrow S'$

Therefore, given $P \vdash_{SOCLP-} l(\bar{t}_n) \leftarrow S_1$, $P \vdash_{SOCLP-} l(\bar{t}_n) \rightarrow S_2$ and $P \vdash_{SOCLP} l(\bar{t}_n) \leftrightarrow S$, the two sets $S_1$, $S_2$ provide, respectively, lower and upper
bounds on the set $S$ (considering $\subseteq$ as an ordering between sets). Thus, the set $S_1$ is correct in the sense that contains only answers of the goal, but not necessarily complete, while $S_2$ is always complete but may be incorrect, since it can include more tuples than those of the answer set. The situation of proposition 4.2 is, in this sense, the limit case, when the inclusions $S_1 \subseteq S \subseteq S_2$ become equalities.

A straightforward corollary of Proposition 4.3 is that for every goal $l(\bar{t}_n)$ and sets $S$, $S_1$ and $S_2$ such that $P \vdash_{\text{SOCLP}} l(\bar{t}_n) \leftarrow S$, $P \vdash_{\text{SOCLP}} l(\bar{t}_n) \leftarrow S_1$, and $P \vdash_{\text{SOCLP}} l(\bar{t}_n) \rightarrow S_2$, $S_1$ is less or equal than $S_2$, i.e., $S_1 \subseteq S_2$. The following proposition shows that this happens even after removing the premise of the existence of a proof for $l(\bar{t}_n) \leftarrow S$:

**Proposition 4.4** Let $P$ be a program, $l(\bar{t}_n)$ a literal with arity $n$, and $S_1, S_2$ two sets such that $P \vdash_{\text{SOCLP}} l(\bar{t}_n) \leftarrow S_1$ and $P \vdash_{\text{SOCLP}} l(\bar{t}_n) \rightarrow S_2$. Then, $S_1 \subseteq S_2$.

The following and last proposition defines the relationship between positive and negative atoms in both calculi

**Proposition 4.5** Let $P$ be a program, $p$ a predicate with arity $n$, and $S_1, S_2$ two sets. The two following statements hold:

i) If $P \vdash_{\text{SOCLP}} p(\bar{t}_n) \leftarrow S_1$ and $P \vdash_{\text{SOCLP}} \neg p(\bar{t}_n) \leftarrow S_2$, then $S_1 \cap S_2 = \emptyset$.

ii) If $P \vdash_{\text{SOCLP}} p(\bar{t}_n) \rightarrow S_1$ and $P \vdash_{\text{SOCLP}} \neg p(\bar{t}_n) \rightarrow S_2$, then $S_1 \cup S_2 = U_n$.

The two statements of the proposition point out the different and complementary nature of the two calculi: in SOCLP _, a term cannot belong simultaneously to the sets associated by the calculus to a literal and to its negation, while this is otherwise possible in SOCLP _. Conversely, SOCLP _ does not ensure that every term is either in the set associated to an atom or in the set of its negation, while SOCLP _ always satisfies this completeness property.

### 5 Conclusions and Future Work

The usual operational mechanisms used in logic programming implementations successively yield the answers for a given goal. While this is a good approach in practical implementations, from the point of view of semantics it is worth considering the set of answers of a goal as a whole. The SOCLP calculus presented in this paper represents this point of view. The calculus derivations can be seen as proof trees proving that a set includes all the possible ground answers for a given goal with respect to a logic program. In Theorem 3.3, we have proved that SOCLP proofs represent faithfully the answer set of a given goal. The main limitation of this calculus is that SOCLP proofs do not always exist; we have only proved completeness (Proposition 3.5) with respect to the class of hierarchical programs over finite Herbrand universes. Nevertheless, SOCLP proofs are often possible in more general programs and
it is part of our future work to extend the completeness result to a broader class of programs. In order to (partially) overcome this limitation, the paper also introduces a calculus SOCLP\(_{-}\) intended for proving subsets of the answer set of a goal. SOCLP\(_{-}\) is coupled through the rule for negation with SOCLP\(_{+}\), a third calculus which can be used for proving that a given set is a superset of the goal’s answer set. The paper proves some basic properties of the three calculi, highlighting that, together, they constitute an useful tool for working with answer sets in logic programming.

As future work, we plan to define a calculus based on SOCLP for defining the semantics of deductive databases \cite{2} and, in particular, of Datalog programs. Notice that SOCLP has already several features that already fit in the usual framework of Datalog, such as the notion of sets of ground tuples as natural answers, and the closed world assumption as a basis for the treatment of negation \cite{2}. Also, the requirement of a finite Herbrand universe for completeness is a condition for Datalog programs, in which no function symbols are allowed. Thus, the future extension of SOCLP should keep these features while extending the class of complete programs to include non-hierarchical programs, which is the case of Datalog.

Acknowledgement

The authors are grateful to F.J. López-Fraguas for his interesting and helpful comments and suggestions.

References


