2) the problem is split into *parallel subproblems* that can be solved by independent processes communicating (thanks to a common memory area) by only checking and updating the values of global variables;
3) a suitable *extension of the unification algorithm* is defined to deal also with global variables.

Split resolution may be regarded as a *complete extension of unit resolution* and coincides with it when applied to unsatisfiable sets of Horn's clauses. This fact partly explains the effectiveness of this new method, that we are experimentally verifying on the implementation we built at the Computer Science Department of the University of Milano. The effectiveness of split resolution can be easily evaluated at propositional level where, whatever inference is chosen among the possible ones, the satisfiability of a clause set is reduced to that of one or (more rarely) two less complex sets (containing less literal occurrences). This characteristic depends on the fact that each propositional inference rule subsumes one or both of its premisses.

To give a first intuitive idea of split resolution, at least at propositional level, we start by showing its origin as a tableaux specialization for clause refutation. Eventually we will consider the first order generalization of this method together with its formal proof of validity and completeness.

2 *Propositional level*

From the logical point of view clauses are disjunctions of literals, therefore the decomposition of disjunctions (rule "∨") is the only tableau rule applicable to them (besides the rule to close branches). Nevertheless, since a clause is a literal set, it is better to modify the "∨" rule in the sense of implicitly taking into account the disjunction associativity and commutativity. Moreover, to reduce the number of branches and therefore make a tableau smaller, it is useful to consider formulas in connection with other formulas containing complementary literals. For instance to refute S°∪{A∨L, A∨B∨¬L}, where A and B are disjunctions of (possibly 0) literals, we might follow Fig.1a and therefore build the subtree T_1 independently from T_2. But the closure of the first branch (the bold one) assures the possibility to close the whole tableau. In fact T_2 can be replaced by an application of the excluded middle rule. This creates two branches the first of which can be closed by reusing T_1 and the other by simply applying the "¬∨" rule (see Fig.1b). So we can introduce the

metarule MR_1 (see Fig.1c) which hangs the formula A∨B to a branch containing both A∨L and A∨B∨¬L. A∨B∨¬L is underlined to mean that it can be ignored in the remaining part of the refutation, since it is subsumed by A∨B. Of course this metarule is correct, because it corresponds to the application of the logical equivalence (A∨L)∧(A∨B∨¬L)↔(A∨L)∧(A∨B). Note that the rule to close branches, traditionally represented by a cross under the branch leaf, is the particular case of MR_1 arising when A and B coincide with the empty clause.

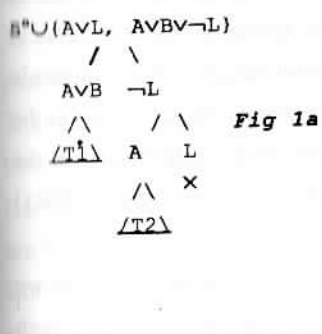


Fig 1a

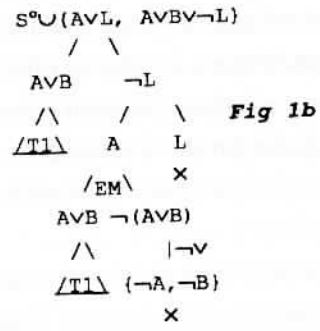


Fig 1b

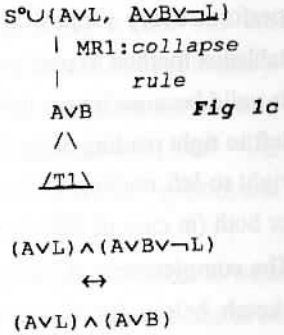


Fig 1c

On the other side to refute an unsatisfiable set of the form S°∪{A∨L, B∨¬L}, where A and B are disjunctions of one or more literals, it is convenient to define another metarule (MR_2) reducing the number of forks, as Fig 2 suggests.

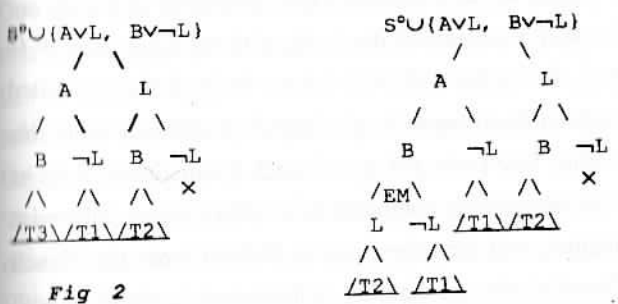
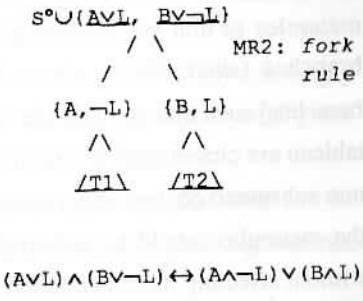


Fig 2



Also in this case the subtree T_3 might be independent from T_1 and T_2 but it is better to build it as an EM application reusing them, since they close the two branches in bold. Therefore, the metarule MR_2 forks a branch containing the formulas {A∨L, B∨¬L} and hangs the nodes {A,¬L} and {B,L} each to a branch of the fork. The subsumption of the premisses (represented by the underlining) follows from the logical equivalence:

$(A \vee L) \wedge (B \vee \neg L) \leftrightarrow (A \wedge \neg L) \vee (B \wedge L)$. Of course the signs of the different occurrences of the propositional letter L can be interchanged without losing the validity of the rules and of the corresponding logical equivalences.

3 *Validity and completeness at propositional level*

The metarules we have just introduced (MR_1, MR_2) are *valid*, in the sense that they allow to close only tableaux with unsatisfiable root, but they are also *complete* i.e. sufficient to perform every propositional refutation. So they define a specialization of the classical tableaux method to treat *ground clauses*, i.e. clauses without bound variables. Each metarule is valid because it only deduces a logical consequence from two premisses, according to the left to right reading of the logical equivalence associated to the metarule itself. Moreover the right to left reading of this equivalence justifies the subsumption of one (in case of MR_1) or both (in case of MR_2) premisses.

The completeness of these metarules is a trivial consequence of the decision procedure we sketch below for the satisfiability of finite sets. Because of the subsumption, each application of our two rules amounts to replacing the set S of the formulas occurring along a branch with one or two sets S' and S" containing less "$\vee$" connectives and such that S is unsatisfiable if and only if so are S' and S". Let be n the number of occurrences of the "$\vee$" connective in the tableau root S. Hence we can construct (by arbitrarily applying our metarules to non yet subsumed clauses) a tableau of depth $\leq n$ with no more than $2^n/2$ branches (since MR_2 reduces by 2 the number of connectives in both the generated branches) such that no metarule is applicable any more to any branch. If all branches of this tableau are closed then S is unsatisfiable. Else each still open branch is satisfiable, since its non subsumed clauses only contain literals with no complement in other clauses (otherwise the metarules would be still applicable), and then they can be deleted from the branch without affecting its satisfiability. Therefore the satisfiability of the branch (coinciding with that of the set of its non subsumed clauses) reduces to that of the empty set, which is trivially satisfiable. But clearly the detection of a satisfiable branch is enough to prove that also S is satisfiable.

Of corse the limit of $2^n/2$ for the number of the tableau branches is in practice exorbitant, since a branch is no more extended once a complementary pair is detected, and the fork rule can be used only when the collapse rule is not usable. Indeed our metarules provide very

efficient tableaux where the fork rule is rarely used and hence in most cases a little more than n inferences are enough to refute by split resolution every set containing n occurrences of "$\vee$".

4 *Lifting the propositional rules to the first order level*

Now we have to solve the problem of generalizing the inference rules of propositional split resolution to deal also with clauses containing universally quantified variables.

At propositional level the necessary and sufficient condition to apply a rule was the presence of two occurrences (with opposite sign) of the same literal in two distinct clauses. The corresponding condition for predicative level is weaker: to perform an inference we need two distinct clauses C_1 and C_2 *(parent clauses)* containing two literals L_1 and $\neg L_2$ *(complementary literals)* whose atoms are *unifiable*, i.e. have the same predicate symbol and are both reducible to the same atom by instantiating their universal variables.

Both propositional rules had the characteristic that all deduced clauses *(consequences)* were subclauses of at least one parent clause. But at predicative level each consequence has to be an *instance of a subclause* of at least one parent clause. A definition is in order before continuing:

DEFINITION: Given a clause C and a substitution σ, σ is a *strict simplifier* of C iff $C\sigma \subset C$ and for each substitution θ, $\sigma \subset \theta$ implies $C\theta \not\subset C$. With *simplifier* of a clause C we mean the empty substitution or a strict simplifier of C.

LEMMA: If σ is a *strict simplifier* of the clause C then C can be split into two non empty subclauses C_1 and C_2 (i.e. $C=C_1 \cup C_2$ and $C_1 \cap C_2 = \emptyset$) such that $C_2\sigma = C_2$, $C_1\sigma \subseteq C_2$ and $C\sigma = C_2$ (easy to prove).

Thanks to the lemma the literals of C_1 are "superfluous" in the sense that they can be deleted from C with no loss nor gain of information, because of the logical equivalence $C \leftrightarrow C_2$.

EXAMPLE: the substitution $\sigma = \{x/g(y)\}$ is a simplifier of $C = P(x) \vee P(f(x)) \vee P(g(y)) \vee P(f(g(y))) \vee Q(y)$, since $C\sigma = P(g(y)) \vee P(f(g(y))) \vee Q(y) \subset C$. Analogously the clause $C = P(x,y,g(v)) \vee P(x,f(z),g(w))$ has the simplifier $\sigma = \{y/f(z), v/w\}$, since $C\sigma = P(x,f(z),g(w)) \subset C$. In both cases the logical equivalence between C and $C\sigma$ is evident. On the contrary the clause $P(x) \vee P(f(x))$ has no strict simplifier: in fact $\forall x P(f(x))$ cannot be deduced from $\forall x (P(x) \vee P(f(x)))$.

Now we show the first order rules (of course the signs of the literals L_1 and L_2 are interchangeable):

1) Let be $A \vee L_1$ and $B \vee \neg L_2$ (more formally $A \cup \{L_1\}$ and $B \cup \{\neg L_2\}$) clauses with disjoint variables, where A and B are possibly empty subclauses, and the literals L_1 and L_2 unify for some most general unifier θ (shortly *m.g.u.* θ). We define the following inference rule (that will be called *linear extension*) where σ is a "simplifier" of $(A \vee B)\theta$ such that $(A \vee B)\theta\sigma \subseteq A\theta$.

$$\textit{linear extension}\qquad \frac{A \vee L_1,\ B \vee \neg L_2}{A\theta\sigma}$$

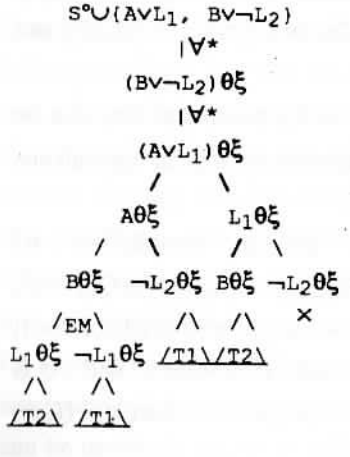
if θ is a most general unifier of L_1 and L_2 and there exists a simplifier σ of $(A \vee B)\theta$ such that $(A \vee B)\theta\sigma \subseteq A\theta$

2) If $A \vee L_1$ and $B \vee \neg L_2$ are clauses where L_1 and L_2 have a m.g.u. θ and A and B are non empty and the restrictions for the applicability of a linear extension are not satisfied, then the following *fork rule* is applicable, for any substitution ξ such that the literals $L_1\theta\xi$ and $L_2\theta\xi$ be ground.

$$\textit{fork rule}\qquad \frac{A \vee L_1,\ B \vee \neg L_2}{\{A\theta\xi, \neg L_2\theta\xi\}\quad \{B\theta\xi, L_1\theta\xi\}}$$

if θ is a most general unifier of L_1 and L_2 and for any ξ such that $L_1\theta\xi$ is ground, provided that there is no simplifier σ of $(A \vee B)\theta$ such that $(A \vee B)\theta\sigma \subseteq A\theta$

The fork rule can be considered a metarule for the method of first order tableaux. In the next figure we replace it with a sequence of applications of traditional tableaux rules to the set $\{A \vee L_1, B \vee \neg L_2\}$:



The closure of the bold branches assures also that of the others, which then can be ignored. The symbol "$\forall$*" stands for the repeated application of the rule of variable instantiation until the effect of the substitution $\theta \circ \xi$ is got. The rightmost branch is closed because θ is a unifier of L_1 and L_2 while the substitution ξ is required for two reasons:

1) instantiating the variables shared by the literal $L_1\theta$, which is equal to $L_2\theta$, with the subclauses $A\theta$ or $B\theta$ in order to allow the two applications of the "$\vee$" rule;

2) instantiating those variables of $L_1\theta$ that, although not present in $A\theta$ nor in $B\theta$, would make the formula $\neg L_1\theta$, introduced by the excluded middle rule, incompatible with the definition of clause in that it would be the negation of a universal formula.

5 *Strengthening the inference rules by means of a limited factorization*

Even though the above inference rules for the predicative split resolution are sufficient to perform every refutation, it is useful to strengthen them by means of the *factorization around the literals to be resolved*. That is, given the parent clauses $A \vee C_1$ and $B \vee C_2$ where the subclauses C_1 and C_2 contain the complementary literals L_1 and respectively $\neg L_2$, if the set $C_1 \cup \text{-}C_2$ is unifiable (defining "$\text{-}C_2$" as the set of the literals of C_2 changed in sign), instead of finding a m.g.u. λ of L_1 and L_2 we compute a m.g.u. θ of $C_1 \cup \text{-}C_2$. With this variation, the inference rules for first order logic assume the following form, where the symbols already introduced keep the same meaning as above:

$$\textit{linear extension}\qquad \frac{A \vee C_1,\ B \vee C_2}{A\theta\sigma}$$

if θ is a most general unifier of $C_1 \cup \text{-}C_2$ and there exists a simplifier σ of $(A \vee B)\theta$ such that $(A \vee B)\theta\sigma \subseteq A\theta$

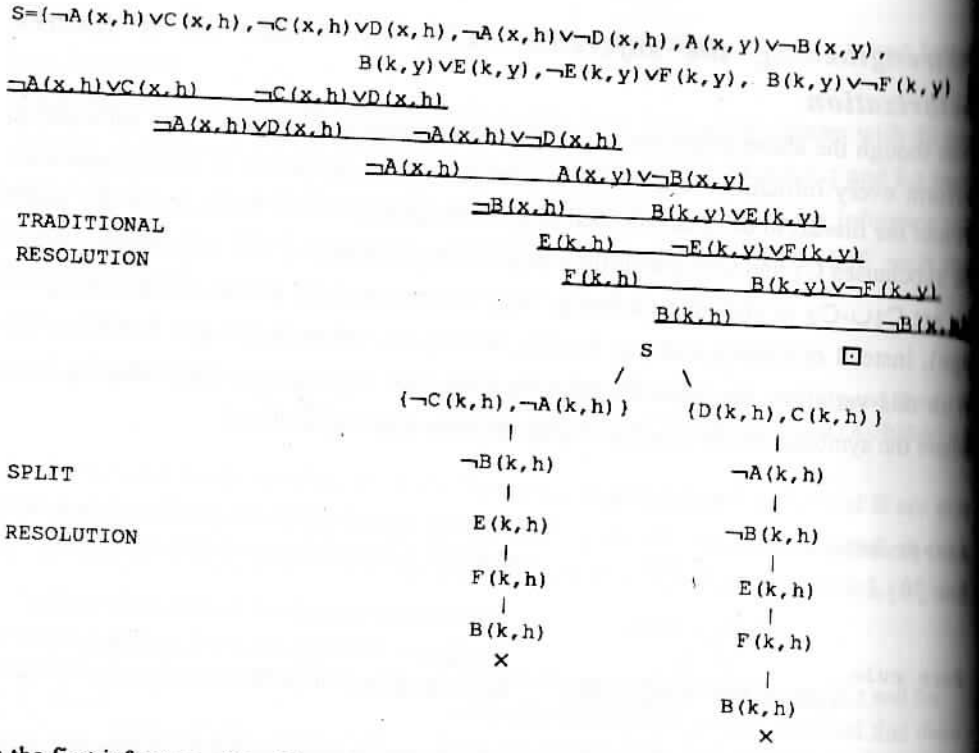
$$\textit{fork rule}\qquad \frac{A \vee C_1,\ B \vee C_2}{\{A\theta\xi, C_2\theta\xi\}\quad \{B\theta\xi, C_1\theta\xi\}}$$

if θ is a most general unifier of $C_1 \cup \text{-}C_2$ and for any ξ such that $C_1\theta\xi$ is ground, provided that there is no simplifier σ of $(A \vee B)\theta$ such that $(A \vee B)\theta\sigma \subseteq A\theta$

This strengthening of the rules, although unnecessary in theory is useful in practice, since it allows to summarize more applications of the preceding weaker rules by means of one strong inference.

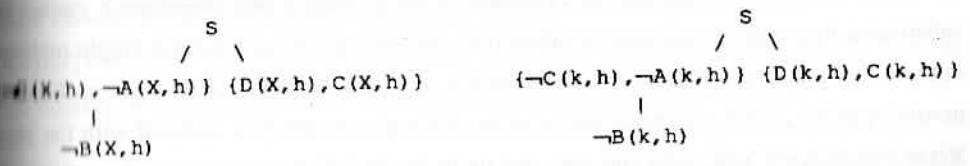
6 *Considerations about the implementation*

Split resolution, thanks to the fork rule, can always generate instances of subclauses of the parent clauses, even when the traditional resolution cannot. Although the fork rule is valid for each substitution ξ complying with the restriction of making ground the complementary literals, ξ must be chosen in a suitable way in order to close the refutation. But when the rule is applied, maybe it is not yet clear how to perform this choice, which is generally determined by the course of the following inferences. Nevertheless we will see that the introduction of "global variables" and the related extension of the unification algorithm will allow us to solve this problem by virtually advancing the determination of ξ. This fact is illustrated in the following example that compares two refutations of the same set S, which are made with traditional and respectively split resolution.



In the first inference of traditional resolution it is not necessary to instantiate the variable x because we create a new clause which is not subclause of any parent clause, while in split resolution it is necessary to instantiate at once the x with k to usefully apply the fork rule, even though at this point of the refutation it is difficult to foresee that the right value for x is just k. To obviate this drawback it is useful to introduce a new syntactical category of terms which will be called *temporary parameters* and will be written with the capital version of the characters that are usually used for variables. In essence a temporary parameter represents an unknown term whose structure will be better specified in the continuation of the refutation, according to the need for unifying pairs of complementary literals. But, although it is used in two distinct branches of a split resolution, a temporary parameter represents a unique term in all the refutation. Therefore the assignment computed during the extension of a branch generated by an application of fork rule, must be compatible with the assignment computed in the other branch. In this sense a temporary parameter is a *global variable*. Hence, after closing the first branch of a fork in which a temporary parameter was introduced, it is necessary to keep track of the final structure attributed to the parameter so

that this be consistently used in the second branch. Using the expedient of instantiating with new temporary parameters the variables remaining in the complementary literals after their unification, the preceding refutation can be started as in the left side of the figure below (X is the new parameter replacing the variable x):



Now we note that, by giving to X the value k, we can perform a linear extension on the pair of formulas {¬B(X,h), B(k,y)∨E(k,y)} and it is clear that the more suitable value for x had to be k. Since no particular structure for X is yet required, we are allowed to suppose that X just coincides with k and we can substitute all the occurrences of X in the whole refutation with k (see right side of the preceding figure). Then we can complete the refutation by means of linear extensions as indicated in the first example.

In the example the final determination of the parameter X was made by the single ground substitution X/k, but in general the structure of a temporary parameter might grow through a sequence of substitutions with terms containing other parameters. Therefore it would be an expensive operation to substitute, each time in the whole refutation, all the occurrences of a parameter with a term that might change again. So it is convenient to associate to each temporary parameter X an *instantiation state* that is a substitution X/t, where t is a term built with function symbols, constants and other temporary parameters but without variables. When X is created, its state consists of the empty substitution that becomes X/t (for some term t) when a unification requires the value t for X. Before comparing two literals we substitute the occurrences of each parameter with the term associated to it in its instantiation state, then we do the comparison by means of an *extended unification algorithm*, obtained from the classical one through the addition of the following rules, where X is a temporary parameter :

1) X unifies with each term t without variables (except when X is a proper subterm of t) through the substitution X/t ;
2) X unifies with any variable y through the substitution y/X;
3) X unifies with any term t containing variables through the substitution $\xi \cup \{X/t\xi\}$, for each substitution ξ assigning new parameters to all the variables of t.

For example X unifies with f(k,Z), where Z is a parameter, through the substitution X/f(k,Z) and X unifies with f(k,y,Z) through {y/Y, X/f(k,Y,Z)} provided that Y is a new parameter.

The need for the substitution ξ of the point 3, stems from the fact that X represents a single term still unknown throughout the branches of the refutation and therefore X cannot be substituted by a term containing variables that, for their universal meaning, might undergo many incompatible substitutions even inside a single branch. After each unification involving rules 1 or 3, the instantiation state of the parameter X is updated with the state X/t or respectively X/tξ, while no state update is required if the unification is based on rule 2 only.

It is worth-while to note that the recourse to temporary parameters is in practice rare, especially if we use the strategy to perform forks only when linear extensions are not possible and, in such case, we give the preference to the forks in which the unifier θ makes the complementary literals ground. Therefore the increased complexity of the unification algorithm, required by split resolution in presence of temporary parameters, is in practice negligible and surely balanced by the corresponding advantage of eliminating the production of long clauses, which is on the contrary an unavoidable drawback in traditional resolution.

7 *Validity and completeness at first order level*

Let us prove that each finite unsatisfiable set of quantified clauses is refutable by split resolution. We will use the already proved completeness of this method for the propositional logic and the second Herbrand's theorem. But at first it is necessary to give some definitions that are not used in the domain of traditional resolution.

DEFINITION: Given two clauses A and B, A is *stronger than* B (A>>B) if and only if there exists a substitution θ such that $A\theta \subseteq B$. Clearly if A>>B then B is a logical consequence of A. This relation extends in natural way to sets, that is, given two clause sets S and S', S is *stronger than* S' (S>>S') if and only if for each clause $C' \in S'$ there exists $C \in S$ such that C>>C'.

EXAMPLE: $P(f(x), y) \vee P(z, g(v)) >> P(f(k), g(k))$ through the substitution θ={x/k, y/g(k), z/f(k), v/k}; $P(x) >> P(f(k)) \vee Q$ for θ={x/f(k)}.

DEFINITION: The literal set C_1 is *superfluous* in the clause $C=C_1 \cup C_2$ if and only if there exists a simplifier σ such that $C_2\sigma=C_2$ and $C_1\sigma \subseteq C_2$.

DEFINITION: The expression "-C" denotes the *complement of the clause* C that is the clause obtained by changing in sign the literals of C. Moreover two clauses C_1 and C_2 are called *complementary* if there is a substitution θ that unifies the set $C_1 \cup -C_2$.

DEFINITION: The *complexity* of a finite set S' of ground clauses, denoted by $\|S'\|$, is the total number of occurrences of the logical connective "$\vee$" in the clauses of S'.

We will suppose to perform, in case of need, a factorization around the resolved literals that is, once selected two parent clauses C_1 and C_2 and two complementary literals $L_1 \in C_1^\circ \subseteq C_1$ and $\neg L_2 \in C_2^\circ \subseteq C_2$, where C_1° and C_2° are complementary clauses, we will suppose to compute the m.g.u. θ of $C_1^\circ \cup -C_2^\circ$, instead of a simple m.g.u. of L_1 and L_2. Of course all the following discussion remains valid if we interchange the signs of the resolved literals L_1 and $\neg L_2$.

Then let be S a quantified clause set with no variable shared by distinct clauses and S' a finite unsatisfiable set of ground clauses such that S>>S'. If $\|S'\|=0$, that is if the complexity of S' is null, then S' contains the empty clause ($\boxdot$) or contains a pair of complementary unit clauses {L},{$\neg$L}. If $\boxdot \in S'$ then $\boxdot \in S$ because S>>S' and the unique clause having an instance that is also subset of $\boxdot$ is the empty clause itself. If on the contrary $\{\{L\},\{\neg L\}\} \subseteq S'$ and $\boxdot \notin S$ (else S is trivially refutable) then being S>>S' there exists a pair of non empty clauses $\{C_1,C_2\} \subseteq S$ such that C_1>>L and C_2>>$\neg$L and therefore for the definition of the relation ">>" there exist two substitutions λ_1 and λ_2 such that $C_1\lambda_1 \subseteq L$ and $C_2\lambda_2 \subseteq \neg L$, that is λ_1 is a unifier of all the literals of C_1 and λ_2 a unifier of all the literals of C_2. Since C_1 and C_2 have no common variables, the union λ of λ_1 and λ_2 is a unifier of the literals of C_1 and of those of $-C_2$, hence there exists a m.g.u. θ such that $C_1\theta = -C_2\theta$. By applying the rule of linear extension to the factors $C_1\theta$ and $C_2\theta$ of C_1 and C_2, we obtain the empty clause, because these factors are complementary unit clauses.

With this the property P(0) is proved, being the predicate P defined as in the following formula, where S stands for a quantified clause set and S' for a finite set of ground clauses:
$P(n)=\forall S \forall S'(\|S'\|=n \wedge S>>S' \wedge S'$ unsatisfiable $\Rightarrow$ S refutable by split resolution).

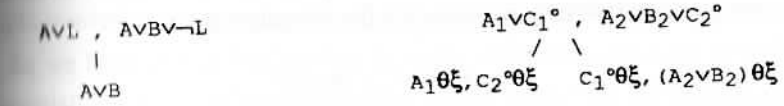
Reasoning by complete induction on the complexity of the ground unsatisfiable set, let [illegible] assume that the property P holds for all natural numbers less than n, and let us prove it [illegible] n.

Then let S' be a finite set of ground clauses such that ||S'||=n, S>>S' and S' [illegible] unsatisfiable. For the completeness of split resolution at propositional level S' is refutable with such method and there are two possible cases in the first inference:

1) The first inference in the refutation of S' is a linear extension with parent clauses A∨L, A∨B∨¬L and conclusion A∨B that makes useless the re-use of the weaker formula A∨B∨¬L (A and B are possibly empty subclauses while L is a literal). The unsatisfiability of S' is so reduced to that of S"=S'\{A∨B∨¬L}∪{A∨B} and obviously ||S"||<n.
Being S>>S', S contains two clauses C_1 and C_2 such that C_1>>A∨L, C_2>>A∨B∨¬L that is, since no variable can occur simultaneously in two clauses of S, there exists a substitution λ such that $C_1\lambda \subseteq A \vee L$ and $C_2\lambda \subseteq A \vee B \vee \neg L$. C_1 can be split into two subclauses A_1 and $C_1°$ such that A_1>>A and $C_1°$>>L and analogously C_2 can be split into the three subclauses A_2>>A, B_2>>B, $C_2°$>>¬L that is briefly $C_1=A_1 \vee C_1°$, $C_2=A_2 \vee B_2 \vee C_2°$. If one of the two subclauses $C_1°$ and $C_2°$ is empty then C_1>>A or C_2>>A∨B, that is in each case not only S>>S' but also S>>S'∪{A∨B}, so clearly S>>S" and, being ||S"||<n, for the inductive hypothesis S is refutable by split resolution.
Supposing on the contrary that $C_1° \neq \square$ and $C_2° \neq \square$, let θ be a m.g.u. of $C_1° \cup -C_2°$. The substitution θ exists because $C_1°\lambda = -C_2°\lambda = \{L\}$ and moreover λ can be written as θ∘σ for some substitution σ. Let be R the clause $(A_1 \vee A_2 \vee B_2)\theta$ and be C the result of eliminating from R the superfluous literals, therefore C⊆R. Now it is better to distinguish two subcases according to the relation between C and the parent clauses C_1 and C_2:

1.a) If $C \subseteq A_1\theta$ or $C \subseteq (A_2 \vee B_2)\theta$ the linear extension can be applied also in the predicative refutation obtaining as result C. Obviously $C\sigma \subseteq A_1\theta\sigma = A_1\lambda \subseteq A$ or $C\sigma \subseteq (A_2 \vee B_2)\theta\sigma = (A_2 \vee B_2)\lambda \subseteq A \vee B$ and therefore C>>A or C>>A∨B that is in each case C>>A∨B. So the refutation of S is reduced to that of S∪{C}, which is stronger than S", having complexity <n, and hence for the inductive hypothesis S∪{C} is refutable through split resolution. But this means that also S is refutable because C is obtained from S through the above described application of the linear extension rule, which is logically valid. The situation is sketched in the following figure where the inference used at ground [illegible] is represented beside the corresponding inference that can be done in the predicative [illegible]

1.b) If, unlike the preceding subcase, C is not an instance of a subset of a parent clause, then a fork is required at predicative level. To this purpose the parent clauses $C_1=A_1 \vee C_1°$ and $C_2=A_2 \vee B_2 \vee C_2°$ must be instantiated by a substitution ζ such that $C_1°\zeta$ and $C_2°\zeta$ are unit clauses without variables and differ only for the sign of their unique literal. Even when ζ does not coincide with the unifier θ defined above, the set $C_1° \cup -C_2° \cup \{L\}$ has the unifier θ and therefore a m.g.u. ζ such that $C_1°\zeta$ and $C_2°\zeta$ do not contain variables. In fact they must be equal to {L} that, being a ground unit clause, is not affected by any substitution. Since the substitution ζ is a unifier of $C_1°$ and $-C_2°$, there exists a substitution ξ such that ζ=θ∘ξ, being θ a m.g.u. for $C_1°$ and $-C_2°$. The determination of ξ is performed by using the temporary parameters as described in the algorithm of split resolution. The application of the fork rule is sketched in the following figure, together with the corresponding linear extension performed in the refutation of S'.



Each of the two formula sets generated by the fork, that is $S_1=S \cup \{A_1\theta\xi, C_2°\theta\xi\}$ and $S_2=S \cup \{C_1°\theta\xi, (A_2 \vee B_2)\theta\xi\}$, is stronger than S" because, being θ∘ξ more general than λ, there exists a substitution χ such that λ=θ∘ξ∘χ. Hence $A_1\theta\xi\chi = A_1\lambda \subseteq A$ and $(A_2 \vee B_2)\theta\xi\chi = (A_2 \vee B_2)\lambda \subseteq A \vee B$, that is (for the definition of ">>") $A_1\theta\xi$>>A∨B and $(A_2 \vee B_2)\theta\xi$>>A∨B. Being ||S"||<n, S_1 and S_2 are refutable by split resolution, for the inductive hypothesis, and consequently so is S too.

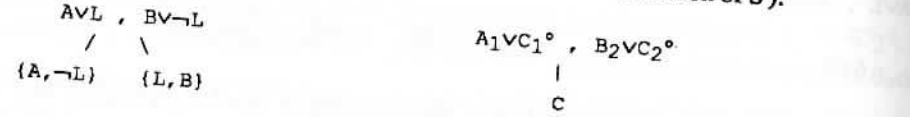
2)Let us suppose the first step of the refutation of S' is a fork from the pair of clauses {A∨L, B∨¬L}⊆S', as sketched in the left hand side of the figure below. By means of this inference rule the refutation of S' is reduced to the equivalent problem of proving the unsatisfiability of the sets $S_1'=S' \setminus \{A \vee L, B \vee \neg L\} \cup \{A, \neg L\}$ and $S_2'=S' \setminus \{A \vee L, B \vee \neg L\} \cup \{B, L\}$. In fact S' is unsatisfiable if and only if so are also the sets S_1' and S_2'.

Since S>>S', there exist in S two clauses C_1 and C_2 such that C_1>>A∨L and C_2>>B∨
Therefore $C_1=A_1 \vee C_1^\circ$ and $C_2=B_2 \vee C_2^\circ$ and these relations also hold: A_1>>A, B_2>>
C_1°>>{L}and C_2°>>{¬L}.

If at least one of the subclauses C_1° and C_2° coincides with the empty clause, then C_1>>A
or C_2>>B and therefore S>>S\{A∨L}∪{A} or S>>S\{B∨¬L}∪{B}. Obviously these two
sets are unsatisfiable, because they are obtained from S' by throwing away a formula and
adding a stronger one that allows to deduce the other. Therefore in each case S is stronger
than a refutable ground set having complexity <n and then, for the inductive hypothesis, S
is refutable by split resolution.

Thus, let us suppose the conditions $C_1^\circ \neq \square$ and $C_2^\circ \neq \square$ hold in the remaining part of the
proof. For the definition of ">>" and the fact that C_1 and C_2 have no common variables, we
can state that there exists a substitution λ such that $C_1^\circ\lambda = -C_2^\circ\lambda = L$, $A_1\lambda \subseteq A$ and $B_2\lambda \subseteq \neg L$.
Let be $R=(A_1 \vee B_2)\theta$, where θ is a m.g.u. of $C_1^\circ \cup -C_2^\circ$ (θ exists because $C_1^\circ \cup -C_2^\circ$ has the
unifier λ) and let C be the subclause of R obtained by deleting the superfluous literals in R.
Now we have to distinguish two subcases :

2.a) $C \subseteq A_1\theta$ or $C \subseteq B_2\theta$ and the predicative refutation of S begins with a linear extension having the conclusion C (see the right hand side of the following figure, on which left hand side we sketch the corresponding inference performed in the refutation of S').



Being θ more general than λ, there exists a substitution σ such that $\lambda = \theta \circ \sigma$. Therefore $C\sigma \subseteq A_1\theta\sigma = A_1\lambda \subseteq A$ or $C\sigma \subseteq B_2\theta\sigma = B_2\lambda \subseteq B$ and consequently C>>A or C>>B. Moreover since S' is refutable and contains the formulas A∨L and B∨¬L, all the more reason for the sets S\{A∨L}∪{A} and S\{B∨¬L}∪{B} being refutable. In fact they both have complexity <n and then, for the inductive hypothesis, the set S∪{C}, which is stronger of at least one of them, is refutable by split resolution. But C is a logical consequence of S and then also S is refutable.

2.b) If C is not a subset of any of the formulas $A_1\theta$ and $B_2\theta$ also the refutation of S starts with a fork on condition to apply, to the parent clauses C_1 and C_2, a substitution making ground and unitary their subclauses C_1° and C_2°. For this purpose the m.g.u. θ of $C_1^\circ \cup$-

C_2° is not necessarily sufficient but, since the substitution λ is a unifier of $C_1^\circ \cup$-$C_2^\circ \cup \{L\}$, there exists a m.g.u. ζ (of the latter literal set) that meets the goal. Since ζ is more general than λ but less general than θ there exist two substitutions ξ and χ such that $\lambda = \theta \circ \zeta \circ \chi$. Moreover $\zeta = \theta \circ \xi$. The determination of ξ is made by using new temporary parameters that are assigned to the variables remaining in $C_1^\circ\theta$. The values of these parameters are specified in the continuation of the refutation according to the requirements of the next unifications, according to the algorithm of split resolution. Then the predicative refutation of S starts with the fork rule that is sketched in the following figure on the right hand side of the inference starting the refutation of S'.

The continuation of the refutation of S consists in proving the unsatisfiability of the two sets $S_1 = S \cup \{A_1\theta\xi, C_2^\circ\theta\xi\}$ and $S_2 = S \cup \{C_1^\circ\theta\xi, B_2\theta\xi\}$. Being $C_1^\circ\theta\xi = \{L\}$, $C_2^\circ\theta x = \{\neg L\}$ and moreover $A_1\theta\xi\chi = A_1\lambda \subseteq A$ and $B_2\theta\xi\chi = B_2\lambda \subseteq B$, then $A_1\theta\xi$>>A and $B_2\theta\xi$>>B. Obviously $C_1^\circ\theta\xi$>>{L} and $C_2^\circ\theta\xi$>>{¬L}. Therefore S_1>>S_1', S_2>>S_2' and, being $\|S_1'\| < n$ and $\|S_2'\| < n$, we can conclude that, for the inductive hypothesis, S_1 and S_2 are refutable by split resolution and consequently also S is refutable by such method.

So we have proved $\forall n P(n)$ that is, each quantified clause set S stronger than a finite unsatisfiable set S' of ground clauses, can be refuted by split resolution. For the second Herbrand's theorem, each finite unsatisfiable clause set has a finite unsatisfiable set of ground instances. Moreover, for each instance C' of a clause C, the relation C>>C' holds. Then we can conclude that each finite unsatisfiable clause set is refutable through split resolution.

References

[BL76] J. Bell, M. Machover. *A course in mathematical logic* , North-Holland (1976)

[CH73] C. Chang, R. Lee. *Symbolic logic and mechanical theorem proving* , Academic Press (1973)

[HD86] W. Hodges. *Logica* , Garzanti (1986)

[RB65] J.A. Robinson. *A machine-oriented logic based on the resolution principle* , in Journal of the A.C.M (1965)

[SM68] R. M. Smullyan. *First-order logic*, Springer-Verlag (1968)

An Algebraic Theory of Observables

Marco Comini Giorgio Levi

Dipartimento di Informatica, Università di Pisa,
Corso Italia 40, 56125 Pisa, Italy,
{comini, levi}@di.unipi.it

Abstract

We give an algebraic formalization of *SLD*-trees and their abstractions (observables). We can state and prove in the framework several useful theorems (lifting, *AND*-compositionality, correctness and full abstraction of the denotation, equivalent top-down and bottom-up constructions) about semantic properties of various observables. Observables are represented by Galois co-insertions and can be used to model abstract interpretation. The constructions and the theorems are inherited by all the observables which can be formalized in the framework. The power of the framework is shown by reconstructing some known examples (answer constraints, call patterns and ground dependencies call patterns).

1 Introduction

SLD-trees are structures used to describe the operational semantics of logic programs. From an *SLD*-tree we can derive several operational properties which are useful for reasoning about programs. Examples are *SLD*-derivations, resultants, call patterns, partial answers, computed answers. All these properties, that we call *observables*, can be obtained as abstractions of the *SLD*-tree. Let for example T be an *SLD*-tree with initial goal g_0. Then, an *SLD*-derivation is any path of T; a resultant is the formula $g_n \rightarrow g_0\vartheta_n$, if g_n and ϑ_n are the goal and the substitution associated to any node n in T; a call pattern is any atom selected in T; a partial answer is the substitution associated to any node n in T (restricted to g_0) while a computed answer is the substitution associated to any node n, such that g_n is the empty clause.

The behavior of a program p (via selection rule r) with respect to a given observable can be understood by observing the corresponding properties for all possible goals. We know from recent results on the semantics [11, 15, 2] that we can characterize this behavior just by observing the property for some specific atomic goals, namely the "most general" atomic goals. This result was first obtained for computed answers in the $\mathcal{S}$-semantics framework [9, 10] and then proved for other less abstract observables, such as resultants, call patterns and partial answers in [11, 15]. The behaviors for most general atomic goals can then be considered a program denotation. In [15] this approach is used to define a semantic framework, where one can

define denotations modeling various observables, by inheriting from the framework the basic constructions and theorems. Some of the denotations enjoy additional properties, such as full abstraction. The technical tool used to define the abstractions is the definition of suitable equivalence relations. We approach here the same problem with a different emphasis and a different technical tool. Our main objective is in fact the formalization of the observable, while our abstractions will be based on abstract interpretation techniques [8]. This will allow us to model, within the same framework, the approximation which is involved in the abstractions used for program analysis. A similar approach can be found in [18].

Our main results are the definition of an algebraic framework for reasoning about *SLD*-trees and their abstractions (observables) in the case of Constraint Logic Programs. The framework is provided with several general theorems (lifting, *AND*-compositionality, correctness and full abstraction, equivalent top-down and bottom-up constructions), which are valid for any abstraction, possibly in a weaker form in the case of abstract interpretation ($\mathcal{I}$-observables). We give the reconstruction of several existing constructions to show the expressive power of the framework.

The paper is organized as follows. Section 2 characterizes the general theory in the case of *SLD*-trees. Section 3 formalizes the main class of observables, i.e. the $\mathcal{A}$-observables for which all the general results are valid. Finally section 4 considers the $\mathcal{I}$-observables, where the results are weaker yet meaningful from the abstract interpretation theory viewpoint. All the proofs can be found in [6].

For a comprehensive description of the semantics of (positive) logic programs see [31, 1]. Σ, Π and V denote a set of function symbols, a set of predicate symbols and a denumerable set of variables respectively. The term algebra over Σ and V is denoted by *Term*. Tuples of variables and terms are sometimes denoted by $\tilde{x}, \tilde{y}, \ldots$ and $\tilde{t}, \tilde{s}, \ldots$. $\tilde{x}_i$ denotes $\tilde{x}_{i1}, \ldots, \tilde{x}_{in_i}$. $\tilde{b}$ denotes a (possibly empty) conjunction of atoms $b_1, \ldots, b_n$ and $\tilde{b}, \tilde{b}'$ denotes the conjunction $b_1, \ldots, b_m, b'_1, \ldots, b'_n$. We denote by $\tilde{t}$ both the tuple and the set of corresponding syntactic objects. *Atom* denotes the set of atoms of the form $q(\tilde{t})$ where $q \in \Pi$ and $\tilde{t} \in Term$.

For a comprehensive description of abstract interpretation see [8]. For a comprehensive description of term systems, closed semirings and constraints systems see [17].

2 The basic framework

When we want to formalize program execution we must take into account, in addition to the inference rules which specify how derivations are made, the properties we observe in a computation (*observables*). In the following we give a rigorous definition of observable in terms of *SLD*-trees. We consider a version of *CLP*, where any goal ($\mathcal{A}$-goal) contains all the information on the derivation. Namely, in addition to the constraint and to the conjunction of atoms, it contains a sequence of clauses (the sequence of clauses used to derive it). Let $\mathcal{A}$ be a semi-distributive constraint system (based on the term system $\mathcal{T}$). An *$\mathcal{A}$-goal* is a formula $c \,\square\, b_1, \ldots, b_n \diamond ks$ with $n \geq 0$, where c is an $\mathcal{A}$-constraint, $b_1, \ldots, b_n$ is a sequence of $(\mathcal{T}, \Pi)$-atoms, and ks is a sequence of clauses of $\mathcal{K}_{\mathcal{A}}$ (the set of $CLP(\mathcal{A})$ clauses). $G_{\mathcal{A}}$ is the set of $\mathcal{A}$-goals and

$P_{\mathcal{A}}$ is the set of programs, i.e. $\mathcal{P}(\mathcal{K}_{\mathcal{A}})$. A *selection rule* r is a function $r : G_{\mathcal{A}} \to \mathbb{N}$ such that $r(c \square b_1, \ldots, b_n \diamond ks) \in \{1, \ldots, n\}$. Given a standard goal $c \square \tilde{b}$, its $\mathcal{A}$-goal version is $g = c \square \tilde{b} \diamond \varepsilon$. When clear from the context, we drop the $\mathcal{A}$ prefix or index.

We must redefine the concepts of resolvent and derivation to take into account the additional information contained in goals.

Definition 2.1 (resolvent) *Let p be a logic program, $g = c \square a_1, \ldots, a_n \diamond ks$ be a goal and $k = h :- c' \square b_1, \ldots, b_m$ be a clause. A* resolvent *of g and k (with selected atom a_i) is the goal $g' = c \otimes a_i{=}h \otimes c' \square a_1, \ldots, a_{i-1}, b_1, \ldots, b_m, a_{i+1}, \ldots, a_n \diamond ks :: [k]$ if the constraint $c \otimes a_i{=}h \otimes c'$ is satisfiable.*

The function $= : Atom \times Atom \to \mathcal{A}$ is defined as $q(t_1, \ldots, t_n){=}r(s_1, \ldots, s_m) = t_1{=}s_1 \otimes \ldots \otimes t_n{=}s_n$ if $q = r$ and $n = m$, 0 otherwise.

Definition 2.2 (*SLD*-derivation) *An* SLD-derivation *of g in p via r is a maximal sequence of goals $g_0, \ldots, g_n, \ldots$ such that $g = g_0$ and, for each i, $g_i = c_i \square \tilde{b}_i \diamond ks_i$; $c \square \tilde{b} \diamond ks_i :: [k]$ is a resolvent of g_i and a variant k of a clause in p not sharing any variable with g_i and ks_i. Moreover the atom in g_i is selected according to r; $g_{i+1} = \exists(c)_{g_0,\tilde{b}} \square \tilde{b} \diamond ks_i :: [k]$. A derivation is* successful *if the last goal has an empty body and is called SLD-refutation.*

If g is a goal, $g \rightsquigarrow c \square \tilde{b} \diamond ks$ denotes an *SLD*-derivation of the goal $c \square \tilde{b} \diamond ks$ from g in p via r. $g \rightsquigarrow c \square \diamond ks$ denotes the refutation of g and c is the computed answer constraint.

Definition 2.3 (*SLD*-tree) *The SLD-tree of g in p via r is the prefix tree built from all the SLD-derivations of g in p via r.*

An *SLD*-tree of a goal g and a program p via a selection rule r is a compact notation for representing a set of derivations of g and p via r. The set of *SLD*-trees (denoted by $ST_p^{r,g}$) is ordered by tree inclusion. This partial order formalizes the evolution of the computation process.

2.1 The domain of *SLD*-trees

SLD-trees are technically hard to handle. Therefore we will introduce another type of notation for representing sets of derivations, which is suitable for our needs and is more compact. We can consider the *resultants*, first introduced by [22] in the theory of partial evaluation, and later used in [1] to discuss the correctness of *SLD*-resolution.

Definition 2.4 (resultant) *Let $g_0, g_1, \ldots$ be a derivation of g_0 in p via r. A* resultant *(of level i), is the expression $g_0 \rightsquigarrow g_i$. $Res_p^{r,g}$ is the set of all the resultants of g in p via r.*

We can define on the set of resultants the following partial order: $g \rightsquigarrow c \square \tilde{b} \diamond ks \leq g \rightsquigarrow c' \square \tilde{b}' \diamond ks'$ iff $ks \leq ks'$, where clause sequences are ordered "lexicografically".

Definition 2.5 (well-formed) *A set A of resultants is* well-formed *if $\rho \in A$ implies $\forall \rho' \leq \rho$, $\rho' \in A$.*

It is easy to see that a well-formed set of resultants contains all the information of an *SLD*-tree, because we can identify each resultant with a node of the tree and vice versa. Thus the set of all the well-formed resultants of g and p via r, $R_p^{r,g} = \{A \in Res_p^{r,g} \mid A \text{ is well-formed}\}$ has the following property.

Proposition 2.6 *$R_p^{r,g}$ is isomorphic to $ST_p^{r,g}$.*

We can define the domain of all the well-formed sets of resultants $R = \cup_{r,p,g} R_p^{r,g}$. We have now an alternative representation of *SLD*-trees. Since we are interested in all the *SLD*-trees of a program p, we define $W^r(p) : P \longrightarrow R$ as $W^r(p) = \cup_{g \in G} R_p^{r,g}$. The set R inherits the set inclusion partial ordering from $ST_p^{r,g}$. It is easy to prove that

Proposition 2.7 *$(R, \subseteq)$ is a complete lattice.*

2.2 The observables

The aim of this paper is to define a precise notion of observable. Roughly speaking we want to express an observable as an abstraction function defined from R into a suitable domain of observable properties.

An observable property domain is a set of properties of the derivation with an ordering relation which can be viewed as an approximation structure. An observation consists in looking at an *SLD*-tree, and then extracting some property (abstraction). Therefore the observable is a function from $ST_p^{r,g}$ to a suitable property domain D, which preserves the approximation structure. Such a function must be a Galois connection between $ST_p^{r,g}$ and D. Because of proposition 2.6 we can give the following definition.

Definition 2.8 (observable) *Let R be the domain of SLD-trees and D be an observable domain. $\alpha : R \longrightarrow D$ is an* observable *when there exists γ s.t. $(\alpha, \gamma) : R \longrightarrow D$ is a Galois co-insertion.*

We denote by the same symbol an observable and the Galois connection it can be extended to.

A choice of the observable α induces an *observational equivalence* $=_\alpha$ on programs. Namely $p_1 =_\alpha p_2$ iff p_1 and p_2 are observationally indistinguishable according to α, i.e. iff the observable properties of p_1 are exactly those of p_2. Namely $p_1 =_\alpha p_2 \iff \alpha(W^r(p_1)) = \alpha(W^r(p_2))$.

Example 2.9 If ξ denotes *computed answer constraints* we can take $D = (\mathcal{P}(G \times \mathcal{A}), \subseteq)$ as properties domain and extend to a connection the function $\xi : R \longrightarrow D$, $\xi(A) = \{(g, c) \mid g \rightsquigarrow c \square \diamond ks \in A\}$. It is easy to see that $p_1 =_\xi p_2$ iff for any goal g, g has the same answer constraints in p_1 and in p_2. ∎

Let us now define what it means for "an observable α to be stronger than another observable α'". The intuition is that $p_1 =_\alpha p_2$ should imply $p_1 =_{\alpha'} p_2$ (i.e. $=_\alpha$ is finer than $=_{\alpha'}$). It is easy to see that this means that all the objects of the weak observable can be expressed by the stronger, by preserving all the properties of the former. This in turn means that there exists a Galois connection between the domains of the observables. If $(f_\alpha, f_\gamma) : A \longrightarrow B, (g_\alpha, g_\gamma) : A \longrightarrow C$ are co-insertions,

then $(\forall x, y \in A\ f_\alpha(x) = f_\alpha(y) \Rightarrow g_\alpha(x) = g_\alpha(y))$ iff $(\exists h : B \longrightarrow C$ co-insertion s.t. $g = h \circ f)$. This result allows us to understand the following

Definition 2.10 (observables approximation) *An observable $\alpha : R \longrightarrow D$ approximates $\alpha' : R \longrightarrow D'$ if there exists a co-insertion $\beta : D \longrightarrow D'$ s.t. $\alpha' = \beta \circ \alpha$.*

We can define an ordering on observables, using closure of Galois co-insertion composition, simply by defining $\alpha \leq \alpha'$ iff α' approximates α.

Lemma 2.11 *Let $(\mathbb{O}, \leq)$ be the set of observables ordered by approximation. $(\mathbb{O}, \leq)$ is a complete lattice.*

2.3 Semantic properties of *SLD*-trees

The goal we want to achieve is to develop a denotation modeling *SLD*-trees. We follow the approach in [14, 2], by first defining a "syntactic" semantic domain (π-interpretation). Our modeling of *SLD*-trees is essentially the basic denotation defined in terms of clauses in [11, 15], extended to handle constraint systems in the style of [17, 19, 20]. In the following for the sake of simplicity we consider the PROLOG leftmost selection rule (denoted by *lm*). All our results can be generalized to local selection rules [23].

Let us consider the equivalence relation of variance extended to resultants $\equiv$.

Definition 2.12 (π-interpretation) *A π-interpretation $\mathcal{I}$ is an element of $R_{/\equiv}$. We denote by $\mathcal{R}$ the set of π-interpretations (that is $\mathcal{R} = R_{/\equiv}$).*

Lemma 2.13 *$(\mathcal{R}, \subseteq)$ is a complete lattice.*

A denotation of the program characterizing its *SLD*-trees computed by using the rule *lm* might be the set of the *SLD*-trees for all the possible goals modulo variance, i.e. $W^{lm}(p)_{/\equiv}$. Because of the *AND*-compositionality theorem 2.16 below, this set can be obtained from the top-down *SLD*-trees denotation, which is the set of *SLD*-trees for the most general atomic goals.

Definition 2.14 (top-down *SLD*-trees denotation $\mathcal{O}$) *Let p be a program. The* top-down *SLD*-trees denotation *of p according to lm is the π-interpretation*

$$\mathcal{O}(p) = \left\{ q(\tilde{x}) \rightsquigarrow c \,\square\, \tilde{b} \diamond ks \in Res_p^{lm, q(\tilde{x})} \mid q \in \Pi_n,\ \tilde{x} \in V^n \right\}_{/\equiv}.$$

It is easy to see that $\mathcal{O}(p)$ is well-formed. Now we prove that this denotation fully characterizes all the *SLD*-trees of p. This is obtained by first proving a lifting lemma which relates the *SLD*-trees of an atomic goal to the *SLD*-trees of the corresponding most general atomic goal. The second step, i.e. the *AND*-compositionality theorem, relates the *SLD*-trees of a conjunctive goal to the *SLD*-trees of the atomic goals.

Lemma 2.15 (lifting) *Let p be a program, $g = c_g \,\square\, q_1(\tilde{t}_1), \ldots, q_n(\tilde{t}_n) \diamond ks_g$ be a goal and $g_{\tilde{x}} = 1 \,\square\, q_1(\tilde{x}_1), \ldots, q_n(\tilde{x}_n) \diamond ks_g$ be its lifted version with $\tilde{x}_1, \ldots, \tilde{x}_n$ containing fresh distinct variables. Then $g \rightsquigarrow c \,\square\, \tilde{b} \diamond ks$ if and only if $g_{\tilde{x}} \rightsquigarrow c_{\tilde{x}} \,\square\, \tilde{b}_{\tilde{x}} \diamond ks$, $c = \exists (c_g \otimes \tilde{x}_1 = \tilde{t}_1 \otimes \ldots \otimes \tilde{x}_n = \tilde{t}_n \otimes c_{\tilde{x}})_{g, \tilde{b}}$ and $\tilde{b} \equiv \tilde{b}_{\tilde{x}} \vartheta^{\tilde{t}}_{\tilde{x}}$, where $\vartheta^{\tilde{t}}_{\tilde{x}} = \{ \tilde{x}_1 / \tilde{t}_1, \ldots, \tilde{x}_n / \tilde{t}_n \}$.*

Theorem 2.16 (*AND*-compositionality) *Let $g = c_g \,\square\, q_1(\tilde{t}_1), \ldots, q_n(\tilde{t}_n) \diamond ks_g$ be a goal and p be a program. Then $g \rightsquigarrow c \,\square\, \tilde{b} \diamond ks$ if and only if $\exists\ r_j = q_j(\tilde{x}_j) \rightsquigarrow c_j \,\square\, \diamond ks_j \in \mathcal{O}(p)$, $1 \leq j < m$, $\exists\ r_m = q_m(\tilde{x}_m) \rightsquigarrow c_m \,\square\, \tilde{b}_m \diamond ks_m \in \mathcal{O}(p)$ s.t. $c = \exists (c_g \otimes \tilde{x}_1 = \tilde{t}_1 \otimes \ldots \otimes \tilde{x}_m = \tilde{t}_m \otimes c_1 \otimes \ldots \otimes c_m)_{g, \tilde{b}}$, $\tilde{b} = \tilde{b}_m, q_{m+1}(\tilde{t}_{m+1}), \ldots, q_n(\tilde{t}_n)$ and $ks = ks_g :: ks_1 :: \ldots :: ks_m$.*

The above closure property allows us to show that the semantics $\mathcal{O}(p)$ is correct and fully abstract for the identical observable.

Corollary 2.17 (correctness and full abstraction) *Let p_1, p_2 be two programs. Then $p_1 =_{id} p_2 \iff \mathcal{O}(p_1) = \mathcal{O}(p_2)$.*

The restriction to local rules plays a fundamental role in the definition of the bottom-up denotation. By using local rules we are able of reconstruct "from the bottom" a derivation, because the local rule chooses only among the atoms introduced in the last derivation step and then "forgets" about the previous steps, which, in a bottom-up construction, are not available yet. The following definition specifies an immediate consequences operator for the *lm* case.

Definition 2.18 (immediate consequences operator T_p) *Let $\mathcal{I}$ be a π-interpretation, p be a program. The* immediate consequences operator *of p via lm $T_p : \mathcal{R} \longrightarrow \mathcal{R}$ is:*

$$\begin{aligned} T_p(\mathcal{I}) = \{\, & q(\tilde{x}) \rightsquigarrow c \,\square\, \tilde{b} \diamond ks \mid \\ & ks = [k] :: ks_1 :: \ldots :: ks_m,\ \tilde{b} = \tilde{b}_m, a_{m+1}, \ldots, a_n, \\ & k = q(\tilde{t}) :- c_k \,\square\, q_1(\tilde{t}_1), \ldots, q_m(\tilde{t}_m), a_{m+1}, \ldots, a_n \in p, \\ & q_m(\tilde{x}_m) \rightsquigarrow c_m \,\square\, \tilde{b}_m \diamond ks_m \in \mathcal{I},\ q_j(\tilde{x}_j) \rightsquigarrow c_j \,\square\, \diamond ks_j \in \mathcal{I},\ 1 \leq j < m, \\ & c = \exists \left(\tilde{x} = \tilde{t} \otimes c_k \otimes \tilde{x}_1 = \tilde{t}_1 \otimes c_1 \otimes \ldots \otimes \tilde{x}_m = \tilde{t}_m \otimes c_m \right)_{\tilde{x}, \tilde{b}} \}. \end{aligned}$$

Lemma 2.19 *$T_p : \mathcal{R} \longrightarrow \mathcal{R}$ is continuous on $\mathcal{R}$.*

Definition 2.20 (fixpoint denotation $\mathcal{F}$) *Let p be a program. The* fixpoint denotation *of p according to lm is the π-interpretation $\mathcal{F}(p) = T_p \uparrow \omega$.*

The following theorem states the equivalence between the top-down and the bottom-up constructions, and shows that $\mathcal{F}(p)$ is also correct and fully-abstract w.r.t. the identity observable.

Theorem 2.21 *Let p be a program. Then $\mathcal{F}(p) = \mathcal{O}(p)$.*

2.4 An algebraic formalization of *SLD*-trees semantic properties

The properties we found for $\mathcal{O}$ and $\mathcal{F}$ allow us to claim that we have a good denotation modeling *SLD*-trees. Our goal however is to find the same results for the denotations modeling more abstract observables. We want then to develop a theory according to which the semantic properties of *SLD*-trees shown in subsection 2.3 are inherited by the denotations which model abstractions of the *SLD*-trees.

In order to define the denotation as a function of the observable, we need a mathematical formalization where one can model the abstraction process and specify properties which have to be shared by the constructions associated to the various abstractions. The first interesting property is the lifting one, which can be modeled only if we can instantiate variables in the derivation by means of constraints. Then we can define an operation $\cdot$ which adds a constraint to a denotation:

$$c \cdot A = \{ c \otimes c' \square \tilde{b}' \diamond ks' \rightsquigarrow c \otimes c'' \square \tilde{b}'' \diamond ks'' \mid c' \square \tilde{b}' \diamond ks' \rightsquigarrow c'' \square \tilde{b}'' \diamond ks'' \in A \}$$

This operation is related to $\otimes$ by the following property: $\forall v \in \mathcal{R}$ and $c_1, c_2 \in \mathcal{A}$, $(c_1 \otimes c_2) \cdot v = c_1 \cdot (c_2 \cdot v)$.

The next relevant property is *AND*-compositionality. We assume that there exists an operation $\times$, defined over the set of denotations, which computes the *AND*-composition of two denotations. In the case of *SLD*-trees denotations, the properties of $\times$ are shown by the following equation.

$$A \times B = \{ c_1 \otimes c_2 \square \tilde{b}_1, \tilde{b}_2 \diamond ks_1 :: ks_2 \rightsquigarrow c_1' \otimes c_2' \square \tilde{b}_1', \tilde{b}_2' \diamond ks_1' :: ks_2' \mid c_1 \square \tilde{b}_1 \diamond ks_1 \rightsquigarrow c_1' \square \tilde{b}_1' \diamond ks_1' \in A, c_2 \square \tilde{b}_2 \diamond ks_2 \rightsquigarrow c_2' \square \tilde{b}_2' \diamond ks_2' \in B \}.$$

$\times$ is (clearly) related to the operation $\otimes$ defined over $\mathcal{A}$, to the conjunction of atom sequences and to the *AND*-compositionality property of theorem 2.16.

Another essential feature that we want to preserve is the *SLD*-trees branching structure. The operation which puts together two denotations nondeterministically is the union of well-formed sets of resultants. By using an algebraic notation, for each pair A, B of well-formed sets of resultants we write $A+B = A \cup B$. It is straightforward to realize that $+$ is well defined over $\mathcal{R}$. The operation $+$ is related to the operation $\oplus$ defined over the constraint system $\mathcal{A}$ by the properties: $(c_1 \oplus c_2)\, v = c_1 v + c_2 v$ and $c\,(v_1 + v_2) = cv_1 + cv_2$. In analogy to what happens in $\mathcal{A}$ for $\oplus$ and $\otimes$, the product $\times$ is (left and right) distributive w.r.t. $+$, i.e. $v_1 \times (v_2 + v_3) = v_1 \times v_2 + v_1 \times v_3$. This property shows that the answers of conjunctive goals are all the compositions of the answers of the conjuncts.

The last issue we must be concerned with is that all the properties must hold "modulo variance". This property is usually modeled by renamings. Therefore we define a "renaming operation" ∇ on the objects of the domain. ∇ commutes with $+$ and $\times$ and is defined as a family of renamings ∇_ϑ depending on the renaming ϑ. In order to satisfy the usual properties of renamings, $\nabla_\vartheta \circ \nabla_{\vartheta'} = \nabla_{\vartheta \circ \vartheta'}$ must hold. Furthermore ∇_ϑ must be an extension of the renaming operation ∂_ϑ of the constraint system $\mathcal{A}$, on which the domain is defined. For each well-formed set A we define

$$\nabla_\vartheta(A) = \{ \partial_\vartheta(c) \square \tilde{b}\vartheta \diamond ks\vartheta \rightsquigarrow \partial_\vartheta(c') \square \tilde{b}'\vartheta \diamond ks'\vartheta \mid c \square \tilde{b} \diamond ks \rightsquigarrow c' \square \tilde{b}' \diamond ks' \in A \}.$$

Definition 2.22 (renaming operator) *Let $(\Theta, \circ, id)$ be the group of renamings. A* renaming operator ∇ *on D is an injective groups homomorphism $(\Theta, \circ, id) \longrightarrow (D \rightarrow D, \circ, id)$ s.t. for each $c \in \mathcal{A}$, $v \in D$ and $\vartheta \in \Theta$ $\nabla_\vartheta(cv) = \partial_\vartheta(c) \nabla_\vartheta(v)$.*

A renaming operator induces a "variance" relation $=_\nabla$. Namely $x =_\nabla y \iff \exists \vartheta : \nabla_\vartheta x = y$. Note that the above renaming operator on R induces exactly the variance relation $\equiv$, i.e. $x =_\nabla y \Leftrightarrow x \equiv y$.

We will appreciate the power of the above algebraic construction in the following section where the operations $+$, $\times$ and ∇ will play a relevant role in the definition of abstract denotations. Right now the formalization allows us to give an alternative nice formulation of theorem 2.16. First of all note that $g \rightsquigarrow c \square \tilde{b} \diamond ks$ iff $g \rightsquigarrow c \square \tilde{b} \diamond ks \in \mathcal{W}'^{\infty}(p)$, where $\mathcal{W}'^{\infty}(p) = W'^{\infty}(p)_{/\equiv}$. Theorem 2.16 is equivalent to $(v \in \mathcal{W}'^{\infty}(p))$ if and only if $(\exists c \in \mathcal{A}, e_1, \ldots, e_n \in \mathcal{O}(p)$ s.t. $v = c \cdot (\{e_1\} \times \ldots \times \{e_n\}))$. Moreover, by defining an *AND* closure operator on $\mathcal{R}$ as

$$\chi_{\mathcal{R}}(x) = min\{ z \in \mathcal{R} \mid x \leq z \text{ and } y, y' \leq z \Longrightarrow \forall c \in \mathcal{A} : c \cdot (y \times y') \leq z \},$$

we can restate theorem 2.16 as $\mathcal{W}'^{\infty} = \chi_{\mathcal{R}} \circ \mathcal{O}$.

3 The abstraction framework

We consider two classes of observables, namely $\mathcal{S}$-observables and $\mathcal{I}$-observables. The $\mathcal{S}$-observables ($\mathcal{S}$emantic observables) are observables for which all the structural properties of *SLD*-trees are preserved. The corresponding denotations provide a correct and complete characterization of the (abstract) program behavior. We show how we can reconstruct within the framework some existing semantics, such as the answer constraint semantics [13] and the call patterns semantics [15, 16], thus obtaining all the relevant theorems simply by specializing the theorems which are valid in the framework.

The $\mathcal{I}$-observables (abstract $\mathcal{I}$nterpretation observables) are meant to capture the abstractions involved in abstract interpretation, where approximation is the rule of the game. Theorems valid for $\mathcal{I}$-observables are therefore weaker and denotations provide characterizations of semantic properties which are correct in the sense of abstract interpretation theory. We show how we can reconstruct the abstract semantics defined in [12], which allows us to derive groundness relations among the arguments of procedure calls.

3.1 An algebraic formalization of observables: the $\mathcal{S}$-observables

We consider observable domains which should have properties analogous to those of R. We want to be able to observe computations of conjunctive goals from the single conjuncts computations. Moreover we do not want to loose the non-deterministic structure and the independence of the results upon renaming. We enforce all these properties by using an $\mathcal{S}$-domain.

Definition 3.1 ($\mathcal{S}$-domain) *A nonempty set D is an* $\mathcal{S}$-domain *on a constraint system $\mathcal{A}(\otimes, \oplus, 1, 0)$, and is denoted by $D(+, \times, \nabla)$, if there exist two operations $+$, $\times$ on D, a renaming ∇ (on D), and two elements 0, 1 in D s.t. $(D, \times, +, 1, 0)$ is a closed semiring. Moreover for each $c \in \mathcal{A}$ and $v \in D$ there exists an element $c \cdot v$ in D s.t. for each $v_1, v_2 \in D$*

1) $c \cdot (v_1 + v_2) = c \cdot v_1 + c \cdot v_2$,
2) $(c_1 \oplus c_2) \cdot v = c_1 \cdot v + c_2 \cdot v$,
3) $0 \cdot v = 0$,
4) $c_1 \cdot (c_2 \cdot v) = (c_1 \otimes c_2) \cdot v$,
5) $1 \cdot v = v$.

Furthermore for each $\vartheta_1, \vartheta_2 \in \Theta$, there exists $\vartheta \in \Theta$ s.t.

a) $\nabla_{\vartheta_1} v_1 \times \nabla_{\vartheta_2} v_2 = \nabla_{\vartheta}(v_1 \times v_2)$, *b)* $\nabla_{\vartheta_1} v_1 + \nabla_{\vartheta_2} v_2 = \nabla_{\vartheta}(v_1 + v_2)$.

Note that the set R of *SLD*-trees of section 2.1 is an $\mathcal{S}$-domain.

We can define, for each $\mathcal{S}$-domain D, a canonical ordering as follows: $v_1 \leq v_2$ iff $v_1 + v_2 = v_2$. It is easy to see, by using proposition 2.7, that $(D, \leq)$ is a complete lattice.

Example 3.2 The set $D = \mathcal{P}(G \times \mathcal{A})$ (the domain of $\xi : R \longrightarrow D$ of example 2.9) is an $\mathcal{S}$-domain. In fact $c \cdot A = \{((c \otimes c' \,\square\, \tilde{b}), c \otimes s) \mid ((c' \,\square\, \tilde{b}), s) \in A, c \otimes s > 0\}$ and

$$A \times B = \{((c \otimes c' \,\square\, \tilde{b}, \tilde{b}'), s \otimes s') \mid ((c \,\square\, \tilde{b}), s) \in A, ((c' \,\square\, \tilde{b}'), s') \in B, s \otimes s' > 0\},$$

where in case of conflict variables are renamed. The sum operation is set union, while the renaming operation is the usual renaming of *CLP*, i.e. $\nabla_{\vartheta}(A) = \{((\partial_{\vartheta}(c) \,\square\, \tilde{b}\vartheta), \partial_{\vartheta}(s)) \mid ((c \,\square\, \tilde{b}), s) \in A\}$. ∎

We want now to define a notion of (forgetful) morphism between $\mathcal{S}$-domains.

Definition 3.3 ($\mathcal{S}$-morphism) *A* morphism between $\mathcal{S}$-domains *($\mathcal{S}$-morphism), $\alpha : D(+, \times, \nabla) \longrightarrow \bar{D}(\bar{+}, \bar{\times}, \bar{\nabla})$ is a surjective application $\alpha : |D| \longrightarrow |\bar{D}|$ s.t. $\forall x, y \in D$, $c \in \mathcal{A}$, $\vartheta \in \Theta$*

1) $\alpha(x + y) = \alpha(x) \bar{+} \alpha(y)$, $\alpha(0) = \bar{0}$, *3)* $\alpha(c \cdot x) = c \cdot \alpha(x)$,
2) $\alpha(x \times y) = \alpha(x) \bar{\times} \alpha(y)$, $\alpha(1) = \bar{1}$, *4)* $\alpha(\nabla_{\vartheta}(x)) = \bar{\nabla}_{\vartheta}(\alpha(x))$.

Example 3.4 The observable ξ of example 2.9 is an $\mathcal{S}$-morphism. ∎

Additivity and surjectivity allow the morphism to associate the right observation in $\bar{D}$ to any concrete object in D. This is because $\mathcal{S}$-morphisms are Galois co-insertions with respect to the canonical orderings.

Proposition 3.5 *Let D, $\bar{D}$ be $\mathcal{S}$-domains. If there exists an $\mathcal{S}$-morphism $\alpha : D(+, \times, \nabla) \longrightarrow \bar{D}(\bar{+}, \bar{\times}, \bar{\nabla})$, then there exists a mapping $\gamma : |\bar{D}| \longrightarrow |D|$ such that (α, γ) is a Galois co-insertion of $(D, \leq)$ onto $(\bar{D}, \bar{\leq})$.*

Definition 3.6 ($\mathcal{S}$-observable) *Let $\bar{D}(\bar{+}, \bar{\times}, \bar{\nabla})$ be an $\mathcal{S}$-domain. A morphism $\alpha : \mathcal{R}(+, \times, \nabla) \longrightarrow \bar{D}(\bar{+}, \bar{\times}, \bar{\nabla})$ is an observable and we call it $\mathcal{S}$-observable. $\mathbb{O}_{\mathcal{S}}$ denotes the set of $\mathcal{S}$-observables.*

Lemma 3.7 *If the $\mathcal{S}$-morphism $\alpha : R \longrightarrow D$ approximates $\alpha' : R \longrightarrow D'$ then there exists an $\mathcal{S}$-morphism $\beta : D \longrightarrow D'$ s.t. $\alpha = \beta \circ \alpha'$.*

As was the case for observables, we can order $\mathbb{O}_{\mathcal{S}}$ by approximation.

Proposition 3.8 *$(\mathbb{O}_{\mathcal{S}}, \leq)$ is a complete sublattice of $(\mathbb{O}, \leq)$.*

3.2 Semantic properties of $\mathcal{S}$-observables

$\mathcal{S}$-domains are strongly related to semantic domains. The operations $+, \times$ of $R(+, \times, \nabla)$ are similar to those of $\mathcal{R}$ defined in subsection 2.3 (that we will still denote by $+, \times$). We can map an $\mathcal{S}$-domain D in a semantic domain $\mathcal{D} = D_{/=_{\nabla}}$ by using the canonical equivalence induced by ∇: $[\cdot]_{\nabla} : D \longrightarrow \mathcal{D}$, $[x]_{\nabla} = [x]_{/=_{\nabla}}$. We have that $\forall [v_1]_{\nabla}, [v_2]_{\nabla} \in \mathcal{R}$ $[v_1]_{\nabla} + [v_2]_{\nabla} = [v_1 + v_2]_{\nabla}$ and $[v_1]_{\nabla} \times [v_2]_{\nabla} = [v_1 \times v_2]_{\nabla}$.

These properties can be generalized to each $\mathcal{S}$-domain $D(+, \times, \nabla)$ simply by defining for each $[x]_{\nabla}, [y]_{\nabla} \in \mathcal{D}$ $[x]_{\nabla} \bar{+} [y]_{\nabla} = [x + y]_{\nabla}$ and $[x]_{\nabla} \bar{\times} [y]_{\nabla} = [x \times y]_{\nabla}$. The set $\mathcal{D}(\bar{+}, \bar{\times})$ inherits from D all the properties we have discussed in section 2.3 for $\mathcal{R}$. From now on, we will call the structure $\mathcal{D}(\bar{+}, \bar{\times})$ *semantic domain*, and denote by $\mathcal{D}$ the set of all the semantic domains.

Each morphism $\alpha : R \longrightarrow D$ can be transformed into a "semantic domains morphism" (that we still denote by α) from $\mathcal{R}$ to the semantic domain $\mathcal{D} = D_{/=_{\nabla}}$. We only need to set $\alpha([x]_{\nabla}) = [\alpha(x)]_{\nabla}$. This morphism is well defined thanks to axiom 3.3 in definition 3.3 which states compatibility between α and $=_{\nabla}$. Thus we have a syntactic abstraction which *preserves the interesting semantic properties of $\mathcal{R}$.*

Now we show how the algebraic construction can be used to easily derive interesting properties.

Lemma 3.9 *Let $\alpha : \mathcal{D} \longrightarrow \mathcal{D}'$ and $\chi_{\mathcal{D}}, \chi'_{\mathcal{D}}$ be the closure operators of $\mathcal{D}, \mathcal{D}'$ respectively. Then $\alpha \circ \chi_{\mathcal{D}} = \chi'_{\mathcal{D}} \circ \alpha$.*

We can then define the α-top-down denotation $\mathcal{O}_{\alpha}(p)$ of a program p as $\alpha(\mathcal{O}(p))$. From lemma 3.9 we immediately derive the (abstract) *AND*-compositionality of $\mathcal{O}_{\alpha}(p)$.

Corollary 3.10 *Let $\alpha : \mathcal{R} \longrightarrow \mathcal{D}$ and $\chi_{\mathcal{D}}$ be the AND-closure operator on $\mathcal{D}$. Then $W^{\infty}_{\alpha} = \chi_{\mathcal{D}} \circ \mathcal{O}_{\alpha}$.*

Theorem 3.11 (abstract *AND*-compositionality) *Let p be a program, $\alpha : \mathcal{R}(+, \times) \longrightarrow \mathcal{D}(\bar{+}, \bar{\times})$ be an $\mathcal{S}$-observable. Then $\bar{v} \in W^{\infty}_{\alpha}(p) \iff \exists c \in \mathcal{A}, \bar{e}_1, \ldots, \bar{e}_n \in \mathcal{O}_{\alpha}(p)$ s.t. $\bar{v} = c \cdot (\{\bar{e}_1\} \bar{\times} \ldots \bar{\times} \{\bar{e}_n\})$.*

Corollary 3.12 (correctness and full abstraction) *Let p_1, p_2 be programs and $\alpha : \mathcal{R} \longrightarrow \mathcal{D}$ be an $\mathcal{S}$-observable. Then $p_1 =_{\alpha} p_2 \iff \mathcal{O}_{\alpha}(p_1) = \mathcal{O}_{\alpha}(p_2)$.*

In the bottom-up case the best approximation for the immediate consequences operator is $T_{p,\alpha} = \alpha \circ T_p \circ \gamma$. If $\leq$ is the canonical ordering on $\mathcal{D}$, $T_{p,\alpha}$ is continuous on the lattice $(D, \leq)$. Indeed $(\alpha \circ T_p \circ \gamma)(\sum x_i) = (\alpha \circ T_p)(\sum \gamma(x_i)) = \alpha(\sum (T_p \circ \gamma)(x_i)) = \sum (\alpha \circ T_p \circ \gamma)(x_i)$. We can then define the fixpoint semantics as $\mathcal{F}_{\alpha}(p) = T_{p,\alpha} \uparrow \omega$.

Definition 3.13 (compatibility between α and T_p) *α is* compatible with T_p *if $T_{p,\alpha} \circ \alpha = \alpha \circ T_p$.*

Theorem 3.14 (bottom-up vs top-down) *Let p be a program and α be an $\mathcal{S}$-observable. Then $\mathcal{O}_{\alpha}(p) \leq \mathcal{F}_{\alpha}(p)$. Moreover if α is compatible with T_p then $\mathcal{O}_{\alpha}(p) = \mathcal{F}_{\alpha}(p)$.*

Some remarks about the above results are necessary. The top-down abstract de
tation, which is defined simply as the abstraction of the top-down denotation,
exactly the same properties of the top-down *SLD*-trees denotation, namely *AN*
compositionality, correctness and full abstraction. The bottom-up abstract deno
tion is in general less precise. The loss of precision is due to the fact that the abstr
immediate consequences operator $T_{p,\alpha}$ is obtained by specializing the general imm
diate consequences operator T_p ($T_{p,\alpha} = \alpha \circ T_p \circ \gamma$). It is exactly this specializati
which may sometimes result in a loss of precision. However, if the observable α
compatible with T_p, the two constructions are equivalent. It is worth noting th
most reasonable observables are indeed compatible with T_p (see the examples belo
A similar relation between the top-down and bottom-up constructions was alre
noted for abstractions of the answer constraint in [17, 19, 20]. In that framework t
equivalence was guaranteed in the case of distributive constraint systems. In such
case, our compatibility condition is always satisfied.

Let us finally note that, when the two constructions are equivalent, the bottom-u
one is indeed more efficient, since abstraction is used at every step of the fixpoint co
struction, thus deriving only the minimal amount of information about the *SLD*-tre
which is needed to characterize the observable property. The top-down construction
on the contrary, is always forced to build complete *SLD*-trees which have later to b
abstracted to get the observation.

Example 3.15 We show now how to reconstruct the *CLP* version of the $\mathcal{S}$-semantic
[9, 10]. We have already shown in example 3.4 that ξ is an $\mathcal{S}$-observable. We can tha
apply theorem 3.11 and the definition of resultant, to obtain the following denotations

$$W^{\mathrm{in}}_{\xi}(p) = \{\, (g,c) \mid g \rightsquigarrow c \,\square \diamond ks \,\},\ \mathcal{O}_{\xi}(p) = \{\, (q(\tilde{x}),c)_{/\equiv} \mid 1 \,\square\, q(\tilde{x}) \diamond \varepsilon \rightsquigarrow c \,\square \diamond ks \,\}.$$

Note that $\mathcal{W}^{\mathrm{in}}_{\xi}(p)$ contains all the answer constraints of p, while $\mathcal{O}_{\xi}(p)$ is *exactly* the
CLP version of the top-down definition of the $\mathcal{S}$-semantics [13]. Corollary 3.12 tells
us that $\mathcal{O}_{\xi}$ is correct and fully abstract w.r.t. answer constraints. Moreover theorem
3.11 tells us that answer constraints for any goal can be derived from the answer
constraints of the most general atomic goals. For the bottom-up case, by applying
the construction, we obtain

$$T_{p,\xi}(X) = \{\, (q(\tilde{x}),c) \mid \exists q(\tilde{t}) :\!- \; c_p \,\square\, q_1(\tilde{t}_1),\ldots,q_n(\tilde{t}_n) \in p,\ (q_i(\tilde{x}_i),c_i) \in X,\\ c = \tilde{x}{=}\tilde{t} \otimes c_p \otimes \tilde{x}_1{=}\tilde{t}_1 \otimes c_1 \otimes \ldots \otimes \tilde{x}_n{=}\tilde{t}_n \otimes c_n \,\}.$$

Moreover $\xi \circ T_p = T_{p,\xi} \circ \xi$. Hence T_p is compatible with ξ, and, because of theorem
3.14, $\mathcal{O}_{\xi} = \mathcal{F}_{\xi}$. This result shows the power of our theory. In fact the proof of the
same result, using classical methods [13], needs much more effort. ∎

Example 3.16 The call patterns of a program p for a goal g and a selection rule r
are the atoms selected in any *SLD*-derivation of g in p via r. We define the $\mathcal{S}$-domain
$CP_{\mathcal{A}}$ made of objects of the form $(g, c \,\square\, a)$, where g is a goal and $c \,\square\, a$ is an atom.
The interpretation is "the execution of the goal g generates a procedure call a with
state (constraint) c". The operations can be defined as follows: $A + B = A \cup B$,

$$c \cdot A = \{\, ((c \otimes c' \,\square\, \tilde{b}),(c \otimes c'' \,\square\, a)) \mid ((c' \,\square\, \tilde{b}),c'' \,\square\, a) \in A, c \otimes c'' > 0 \,\},$$

$$A * B = \{\, (g,g',c \,\square\, a) \mid (g,c \,\square\, a) \in A \,\} \cup \{\, (g,g',c \otimes c' \,\square\, a') \mid \\ (g,c \,\square\,) \in A, (g',c' \,\square\, a') \in B \,\}$$

$$\Psi_{\vartheta}(A) = \{\, (\partial_{\vartheta}(c) \,\square\, \tilde{b}\vartheta, \partial_{\vartheta}(c') \,\square\, a\vartheta) \mid (c \,\square\, \tilde{b}, c' \,\square\, a) \in A \,\}.$$

The abstraction which allows us to obtain the call patterns is

$$\eta(A) = \{\, (g,c \,\square\, a) \mid g \rightsquigarrow c \,\square\, a, \tilde{b} \diamond ks \in A \,\}$$

By using the definition of resultant, we have that $\mathcal{W}^{\mathrm{in}}_{\eta}(p) = \{\, (g,c \,\square\, a) \mid g \rightsquigarrow c \,\square\, a, \tilde{b} \diamond$
$\}$, which is exactly the set of all the call patterns of p. The top-down denotation is
$\mathcal{O}_{\eta}(p) = \{\, (q(\tilde{x}),c \,\square\, a) \mid 1 \,\square\, q(\tilde{x}) \diamond \varepsilon \rightsquigarrow c \,\square\, a, \tilde{b} \diamond ks \,\}$, while the immediate consequences
operator turns out to be

$$T_{p,\eta}(X) = \{\, (q(\tilde{x}),c \,\square\, a) \mid \exists q(\tilde{t}) :\!- \; c_p \,\square\, q_1(\tilde{t}_1),\ldots,q_n(\tilde{t}_n) \in p, \text{ and} \\ c = \tilde{x}{=}\tilde{t} \otimes c_p,\ a = q_1(\tilde{t}_1) \text{ or} \\ c = \tilde{x}{=}\tilde{t} \otimes c_p \otimes \tilde{t}_1{=}\tilde{y} \otimes c',\ (q_1(\tilde{y}),c' \,\square\, a) \in X \,\}.$$

Since T_p is compatible with η, $\mathcal{F}_{\eta}(p) = \mathcal{O}_{\eta}(p)$. $\mathcal{F}_{\eta}$ is then correct and fully abstract
w.r.t. $=_{\eta}$. We obtained the call pattern semantics defined in [16, 11, 15]. ∎

3.8 Other observables

For a generic observable $\alpha : R \longrightarrow D$, the previous theorems state that if α is not an $\mathcal{S}$-
observable then we cannot have a correct and fully abstract denotation which is *AND*-
compositional, because in such a case D should have the structure of an $\mathcal{S}$-domain,
thus contradicting the hypothesis. Instead we can have a correct *AND*-compositional
denotation which is minimal w.r.t. the information content. In analogy to what we
did in the case of observables, we can consider the set $\mathscr{S}$ of *AND*-compositional
denotations $\mathcal{S} : \mathcal{P} \longrightarrow \mathcal{D}$ with $\mathcal{D} \in \mathscr{D}$. We can partially order $\mathscr{S}$ by approximation.

Definition 3.17 (denotation approximation) *A denotation $\mathcal{S} : \mathcal{P} \longrightarrow \mathcal{D} \in \mathscr{S}$ approximates $\mathcal{S}' : \mathcal{P} \longrightarrow \mathcal{D}'$ if there exists a Galois co-insertion $\alpha : \mathcal{D} \longrightarrow \mathcal{D}'$ s.t. $\mathcal{S}' = \alpha \circ \mathcal{S}$.*

Lemma 3.18 *Let $\mathcal{S} \leq \mathcal{S}'$ iff $\mathcal{S}'$ approximates $\mathcal{S}$. $(\mathscr{S}, \leq)$ is a complete lattice.*

Note that every denotation which approximates another denotation which is correct
w.r.t. an observable is still correct w.r.t. the same observable.

Since $\mathbb{O}$ is a complete lattice, we have that $\alpha \uparrow s = \{\, \beta \in \mathbb{O}_s \mid \beta \leq \alpha \,\}$ has
a minimum α'. Hence we can find a denotation (correct and fully abstract for α')
which models correctly α and is *minimal*.

Theorem 3.19 (the denotation) *For each observable α there exists a minimal AND-compositional denotation correct w.r.t. it, i.e. which is approximated by all the other AND-compositional correct denotations.*

4 Abstract interpretation

One more motivation for our algebraic construction can be found in abstract interpretation. The essence of abstract interpretation is to give a non–standard interpretation to the language. In the *CLP* case, as shown in [5, 17], we only need to give a non-standard interpretation to constraints (and terms).

In general a constraint system is an interpretation (in a semi-closed semiring) for constraint formulas. According to the approach of [19, 17, 20] constraint systems are related by means of constraint system semi-morphisms. The program interpretation process is expressed in terms of a set of algebraic operators which model how data objects are collected during the computation. An abstract interpretation for a given $CLP(\mathcal{A})$ program is then the semantics of an abstract program in $CLP(\mathcal{A}')$ where $\mathcal{A}'$ is a suitable constraint system correct w.r.t. $\mathcal{A}$.

In our framework, abstract interpretation can be viewed as the composition of a constraint system semi-morphism (the abstraction of the domain interpretation) and of an $\mathcal{S}$-observable, which chooses an adequate abstraction of *SLD*-trees. The construction is based on semi-morphisms on constraint systems. A constraint system semi-morphism $\mu : \mathcal{A} \longrightarrow \mathcal{A}'$ can be extended to an $\mathcal{S}$-domain semi-morphism $\mu^{\mathcal{D}} : D_{\mathcal{A}} \longrightarrow D_{\mathcal{A}'}$ in a natural way.

Definition 4.1 ($\mathcal{S}$-semi-morphism) *Let $D_{\mathcal{A}}$, $D'_{\mathcal{A}'}$ be $\mathcal{S}$-domains, with $\mathcal{A}'$ correct w.r.t. $\mathcal{A}$. An* $\mathcal{S}$-semi-morphism *$\alpha_\mu : D_{\mathcal{A}} \longrightarrow D'_{\mathcal{A}'}$, based on the constraint system semi-morphism $\mu : \mathcal{A} \longrightarrow \mathcal{A}'$, is the composition of $\mu^{\mathcal{D}}$ and an $\mathcal{S}$-morphism $\alpha : D_{\mathcal{A}'} \longrightarrow D'_{\mathcal{A}'}$, i.e. $\alpha_\mu = \alpha \circ \mu^{\mathcal{D}}$.*

Proposition 4.2 *Let $D_{\mathcal{A}}$, $D'_{\mathcal{A}'}$ be $\mathcal{S}$-domains, with $\mathcal{A}'$ correct w.r.t. $\mathcal{A}$. If there exists an $\mathcal{S}$-semi-morphism $\alpha_\mu : D_{\mathcal{A}} \longrightarrow \bar{D}_{\mathcal{A}'}$, then there exists a mapping $\gamma_\mu : |\bar{D}_{\mathcal{A}'}| \longrightarrow |D_{\mathcal{A}}|$ such that (α_μ, γ_μ) is a Galois co-insertion of $(D_{\mathcal{A}}, \leq)$ onto $(\bar{D}_{\mathcal{A}'}, \bar{\leq})$.*

Since $\mathcal{S}$-semi-morphisms from R are strongly related to abstract interpretation, we will call them *$\mathcal{I}$-observables*. Let $\mathbb{O}_i$ denote the set of $\mathcal{I}$-observables. $\mathbb{O}_i$ can be ordered by approximation, as it was the case for $\mathbb{O}_s$.

Proposition 4.3 *$(\mathbb{O}_i, \leq)$ is a complete sublattice of $(\mathbb{O}, \leq)$. Moreover $(\mathbb{O}_s, \leq)$ is a complete sublattice of $(\mathbb{O}_i, \leq)$.*

The following discussion on the abstract denotations relies on the notion of "abstracting the program". The idea is to replace the constraints (defined on the constraint system $\mathcal{A}$) in the original program with their abstract version (defined on a constraint system $\mathcal{A}'$ correct w.r.t. $\mathcal{A}$), thus obtaining a *CLP* program on a different constraint system. If $\mu : \mathcal{A} \longrightarrow \mathcal{A}'$ is the semi-morphism which formalizes the (correct) constraint system abstraction we denote by $\mu^{\mathcal{P}} : \mathcal{P}_{\mathcal{A}} \longrightarrow \mathcal{P}_{\mathcal{A}'}$ the homomorphism obtained by extending μ to programs in the natural way, i.e. by applying μ to the constraints and terms occurring in the program. The abstract denotation of p is simply the denotation $\mathcal{O}(\mu^{\mathcal{P}}(p))$ of the abstract program $\mu^{\mathcal{P}}(p)$. When clear from the context we denote $\mu^{\mathcal{P}}$ by μ. The following theorem generalizes a result in [17] and states that the semantics of the "abstract program" is a safe approximation of the abstraction of the semantics of the program.

Theorem 4.4 (abstract program) *Let $\mu : \mathcal{A} \longrightarrow \mathcal{A}'$ be a constraint system semi-morphism. Then $\mu^{\mathcal{R}}(\mathcal{O}(p)) \leq \mathcal{O}(\mu(p))$.*

Since an $\mathcal{I}$-observable on the constraint system $\mathcal{A}$ is obtained by an $\mathcal{S}$-observable on the constraint system $\mathcal{A}'$ we expect the semantics construction to be almost the same for the abstract interpretation case. Obviously the theorems are weaker, because of the lack of precision of the denotations.

Definition 4.5 (abstract denotations) *The ideal top-down denotation of an $\mathcal{I}$-observable $\alpha_\mu : \mathcal{R}_{\mathcal{A}} \longrightarrow \mathcal{D}_{\mathcal{A}'}$ is $\mathcal{O}_{\alpha_\mu}(p) = \alpha_\mu(\mathcal{O}(p))$, while the abstract top-down denotation is $\mathcal{O}_\alpha(\mu(p)) = \alpha(\mathcal{O}(\mu(p)))$. The ideal immediate consequences operator is $T_{p,\alpha_\mu} = \alpha_\mu \circ T_p \circ \gamma_\mu$, while the abstract immediate consequences operator is $T_{\mu(p),\alpha} = \alpha \circ T_{\mu(p)} \circ \gamma$. The ideal bottom-up denotation is $\mathcal{F}_{\alpha_\mu}(p) = T_{p,\alpha_\mu} \uparrow \omega$, while the abstract bottom-up denotation is $\mathcal{F}_\alpha(\mu(p)) = T_{\mu(p),\alpha} \uparrow \omega$.*

The following theorem relates different abstract interpretation mechanisms, i.e. the bottom-up execution of the abstract program $\mathcal{F}_\alpha(\mu(p))$, the top-down abstract execution of the abstract program $\mathcal{O}_\alpha(\mu(p))$, the abstraction of the top-down concrete execution $\mathcal{O}_{\alpha_\mu}(p)$ and the (specialized) bottom-up execution of the concrete program $\mathcal{F}_{\alpha_\mu}(p)$. As one could expect, the ideal top-down denotation is (safely) approximated by all the other denotations. $\mathcal{F}_\alpha(\mu(p))$ is the least precise and, in the case of compatibility, we have the usual equivalence between top-down and bottom-up executions. This is shown by the following theorem.

Theorem 4.6 *Let p be a program, $\alpha_\mu : \mathcal{R}_{\mathcal{A}} \longrightarrow \mathcal{D}_{\mathcal{A}'}$ be an $\mathcal{I}$-observable. Then $\mathcal{F}_\alpha(\mu(p)) \geq \mathcal{O}_\alpha(\mu(p))$, $\mathcal{F}_\alpha(\mu(p)) \geq \mathcal{F}_{\alpha_\mu}(p)$, $\mathcal{O}_\alpha(\mu(p)) \geq \mathcal{O}_{\alpha_\mu}(p)$, $\mathcal{F}_{\alpha_\mu}(p) \geq \mathcal{O}_{\alpha_\mu}(p)$. Moreover if α is compatible with $T_{\mu(p)}$ we have $\mathcal{F}_\alpha(\mu(p)) = \mathcal{O}_\alpha(\mu(p)) \geq \mathcal{F}_{\alpha_\mu}(p) = \mathcal{O}_{\alpha_\mu}(p)$.*

The next theorem shows that *AND*-compositionality holds in the denotation based on the abstract program.

Theorem 4.7 (*AND*-semi-compositionality) *Let p be a program, $\alpha_\mu : \mathcal{R}_{\mathcal{A}}(+, \times) \longrightarrow \mathcal{D}_{\mathcal{A}'}(\bar{+}, \bar{\times})$ an $\mathcal{I}$-observable. Then*

$$v' \in \mathcal{W}^{\mathrm{lm}}_{\alpha}(\mu(p)) \iff \exists c' \in \mathcal{A}', e'_1, \ldots, e'_n \in \mathcal{O}_\alpha(\mu(p)) \text{ s.t. } v' = c' \cdot (\{e'_1\} \bar{\times} \ldots \bar{\times} \{e'_n\}),$$

$$v' \in \mathcal{W}^{\mathrm{lm}}_{\alpha_\mu}(p) \iff \exists c' \in \mathcal{A}', e'_1, \ldots, e'_n \in \mathcal{O}_{\alpha_\mu}(p) \text{ s.t. } v' \leq c' \cdot (\{e'_1\} \bar{\times} \ldots \bar{\times} \{e'_n\}).$$

Example 4.8 Let $\mathcal{A}_{\mathcal{H}}$ be the Herbrand constraint system. We show how to reconstruct the ground dependency analysis for call patterns described in [12]. The abstract domain of computation is *Prop* [7], consisting of propositional formulas which provide a concise representation of abstract substitutions which describe ground dependency relations among arguments of a procedure call.

Prop can be formalized as a constraint system. Take a term system morphism μ which maps each term onto its set of free variables (the empty set for ground terms). $\mathcal{A}_{Prop}$ is the algebra of possibly existentially quantified disjunctions of formulas $(Prop, \wedge, \vee, true, false, \exists_x, \partial^t_x, \Lambda(t) \leftrightarrow \Lambda(t'))$, where each t belongs to V^n (for some n) and $\Lambda(t) = x_1 \wedge \ldots \wedge x_n$ for $t = \{x_1, \ldots, x_n\}$ $(\Lambda(\emptyset) = true)$.

We can define a constraint system semi-morphism $\mu : \mathcal{A}_{\mathcal{H}} \longrightarrow \mathcal{A}_{Prop}$ as

$$\mu(c) = \begin{cases} false & \text{if } c = 0 \\ \Lambda(var(c)) & \text{if } c \text{ is a simple constraint} \\ \Lambda(\mu(t)) \leftrightarrow \Lambda(\mu(t')) & \text{if } c = t{=}t' \\ \mu(c_1) \wedge \mu(c_2) & \text{if } c = c_1 \otimes c_2 \\ \mu(c_1) \vee \mu(c_2) & \text{if } c = c_1 \oplus c_2 \end{cases}$$

and extend it to well-formed sets of resultants as

$$\mu^{\mathcal{R}}(A) = \{\mu(c) \,\square\, q_1(\mu(\tilde{t}_1)), \ldots, q_n(\mu(\tilde{t}_n)) \diamond ks \rightsquigarrow \mu(c') \,\square\, q'_1(\mu(\tilde{t}'_1)), \ldots, q'_{n'}(\mu(\tilde{t}_{n'})) \diamond ks' \mid c \,\square\, q_1(\tilde{t}_1), \ldots, q_n(\tilde{t}_n) \diamond ks \rightsquigarrow c' \,\square\, q'_1(\tilde{t}'_1), \ldots, q'_{n'}(\tilde{t}'_{n'}) \diamond ks' \in A\}.$$

Since the call pattern observable η of example 3.16 was defined for any constraint system $\mathcal{A}$, we can compose it with $\mu^{\mathcal{R}}$, $\eta_\mu : R_{\mathcal{A}_{\mathcal{H}}} \longrightarrow CP_{\mathcal{A}_{Prop}}$:

$$\eta_\mu(A) = (\eta \circ \mu^{\mathcal{R}})(A) = \{(g, c \,\square\, a) \mid g \rightsquigarrow c \,\square\, a, \tilde{b} \diamond ks \in \mu^{\mathcal{R}}(A)\}$$

and obtain an abstract interpretation in *Prop*. The ground dependencies call patterns denotation are

$$\mathcal{O}_{\eta_\mu}(p) = \{(q(\tilde{x}), c_\mu \,\square\, q_1(\tilde{t}_\mu)) \mid 1 \,\square\, q(\tilde{x}) \diamond \varepsilon \rightsquigarrow c \,\square\, q_1(\tilde{t}), \tilde{b} \diamond ks,\ c_\mu = \mu(c),\ \tilde{t}_\mu = \mu(\tilde{t})\}$$

$$\mathcal{O}_{\eta}(\mu(p)) = \{(q(\tilde{x}), c \,\square\, a) \mid 1 \,\square\, q(\tilde{x}) \diamond \varepsilon \underset{\mu(p),\eta}{\rightsquigarrow} c \,\square\, a, \tilde{b} \diamond ks\},$$

$$T_{p,\eta_\mu}(X) = \{(q(\tilde{x}), c_\mu \,\square\, q_1(\tilde{t}_\mu)) \mid \exists q(\tilde{t}) :\!- c_p \,\square\, q_1(\tilde{t}_1), \ldots, q_n(\tilde{t}_n) \in p, \text{ and } c_\mu = \Lambda(\tilde{x}) \leftrightarrow \Lambda(\tilde{t}) \wedge \mu(c_p),\ \tilde{t}_\mu = \mu(\tilde{t}_1) \text{ or } c_\mu = \Lambda(\tilde{x}) \leftrightarrow \Lambda(\tilde{t}) \wedge \mu(c_p) \wedge \Lambda(\tilde{t}_1) \leftrightarrow \Lambda(\tilde{y}) \wedge c'_\mu,\ (q_1(\tilde{y}), c'_\mu \,\square\, q_1(\tilde{t}_\mu)) \in X\}.$$

$$T_{\mu(p),\eta}(X) = \{(q(\tilde{x}), c \,\square\, a) \mid \exists q(\tilde{t}) :\!- c_p \,\square\, q_1(\tilde{t}_1), \ldots, q_n(\tilde{t}_n) \in \mu(p), \text{ and } c = \Lambda(\tilde{x}) \leftrightarrow \Lambda(\tilde{t}) \wedge c_p,\ a = q_1(\tilde{t}_1) \text{ or } c = \Lambda(\tilde{x}) \leftrightarrow \Lambda(\tilde{t}) \wedge c_p \wedge \Lambda(\tilde{t}_1) \leftrightarrow \Lambda(\tilde{y}) \wedge c',\ (q_1(\tilde{y}), c' \,\square\, a) \in X\}.$$

Since $T_{\mu(p)}$ is compatible with η, $\mathcal{F}_\alpha(\mu(p)) = \mathcal{O}_\alpha(\mu(p))$. $\mathcal{O}_\alpha(\mu(p))$, which is a goal independent denotation computable either top-down or bottom-up, is the semantics obtained in [12] and, through a magic set like transformation, in [4]. ∎

5 Conclusions

We have defined an algebraic framework which allows us to prove several properties of concrete and abstract *SLD*-trees. The framework provides: a) a denotation consisting of all the *SLD*-trees obtained from most general atomic goals, with an equivalent alternative fixpoint construction; b) a set of important theorems (lifting, *AND*-compositionality) which show that the denotation characterizes the *SLD*-trees for any goal; c) a mechanism ($\mathcal{S}$-observable) for abstracting the semantics, which guarantees that

the general theorems do hold for any abstraction and always leads to the best (correct and fully abstract) denotation; d) a mechanism ($\mathcal{I}$-observable) to model abstraction by approximation, which guarantees that a weaker form of the general theorems is still valid and provides the semantic basis for abstract interpretation.

We have not considered yet two issues that could be discussed within the framework. The first one is related to *OR-compositionality* [3], which is a relevant property if we want to be able to reason about programs in a modular way. *SLD*-trees are indeed *OR*-compositional. However, the abstraction process can destroy this property. For example, two of the abstractions that we have considered, namely computed answer constraints and call patterns, are not *OR*-compositional. The theory should be extended with a characterization of *OR*-compositional observables. For non-*OR*-compositional observables there should be a result similar to the one of theorem 3.19. The second issue is related to the computation rule. The current results apply to the case of local selection rules, even if we have considered the leftmost selection rule only. The property that could be analyzed within the framework is the *independence from the selection rule*. *SLD*-trees do depend on the selection rule. However, more abstract observables, such as answer constraints, are independent from the selection rule. A second relevant extension of the framework might be the definition of conditions which guarantee that the abstractions are independent from the selection rule.

References

[1] K. R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 495–574. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[2] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: Theory and applications. *Journal of Logic Programming*, 1994. to appear.

[3] A. Bossi, M. Gabbrielli, G. Levi, and M. C. Meo. A Compositional Semantics for Logic Programs. *Theoretical Computer Science*, 122(1-2):3–47, 1994.

[4] M. Codish and B. Demoen. Analysing Logic Programs using "prop"-ositional Logic Programs and a Magic Wand. In D. Miller, editor, *Proc. 1993 Int'l Symposium on Logic Programming*, pages 114–129. The MIT Press, Cambridge, Mass., 1993.

[5] P. Codognet and G. Filè. Computations, Abstractions and Constraints. In *Proc. Fourth IEEE Int'l Conference on Computer Languages*. IEEE Press, 1992.

[6] M. Comini and G. Levi. An algebraic theory of observables. Technical report, Dipartimento di Informatica, Università di Pisa, 1994.

[7] A. Cortesi, G. Filè, and W. Winsborough. *Prop* revisited: Propositional Formula as Abstract Domain for Groundness Analysis. In *Proc. Sixth IEEE Symp. on Logic In Computer Science*, pages 322–327. IEEE Computer Society Press, 1991.

[8] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proc. Sixth ACM Symp. Principles of Programming Languages*, pages 269–282, 1979.

[9] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A new Declarative Semantics for Logic Languages. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 993–1005. The MIT Press, Cambridge, Mass., 1988.

[10] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative Modeling of the Operational Behavior of Logic Languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[11] M. Gabbrielli. *The Semantics of Logic Programming as a Programming Language.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 1992.

[12] M. Gabbrielli and R. Giacobazzi. Goal independency and call patterns in the analysis of logic programs. In *Proc. SAC'94*, 1994.

[13] M. Gabbrielli and G. Levi. Modeling Answer Constraints in Constraint Logic Programs. In K. Furukawa, editor, *Proc. Eighth Int'l Conf. on Logic Programming*, pages 238–252. The MIT Press, Cambridge, Mass., 1991.

[14] M. Gabbrielli and G. Levi. On the Semantics of Logic Programs. In J. Leach Albert, B. Monien, and M. Rodriguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium*, volume 510 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, Berlin, 1991.

[15] M. Gabbrielli, G. Levi, and M. C. Meo. Observational Equivalences for Logic Programs. In K. Apt, editor, *Proc. Joint Int'l Conf. and Symposium on Logic Programming*, pages 131–145. The MIT Press, Cambridge, Mass., 1992.

[16] M. Gabbrielli and M. C. Meo. Fixpoint Semantics for Partial Computed Answer Substitutions and Call Patterns. In H. Kirchner and G. Levi, editors, *Algebraic and Logic Programming, Proceedings of the Third International Conference*, volume 632 of *Lecture Notes in Computer Science*, pages 84–99. Springer-Verlag, Berlin, 1992.

[17] R. Giacobazzi. *Semantic Aspects of Logic Program Analysis.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 1992.

[18] R. Giacobazzi. On the Collecting Semantics of Logic Programs. In F. S. de Boer and M. Gabbrielli, editors, *Verification and Analysis of Logic Languages, Proc. of the Post-Conference ICLP Workshop*, pages 159–174, 1994.

[19] R. Giacobazzi, S. K. Debray, and G. Levi. A Generalized Semantics for Constraint Logic Programs. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pages 581–591, 1992.

[20] R. Giacobazzi, G. Levi, and S. K. Debray. Joining Abstract and Concrete Computations in Constraint Logic Programming. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Algebraic Methodology and Software Technology (AMAST'93), Proceedings of the Third International Conference on Algebraic Methodology and Software Technology*, Workshops in Computing, pages 111–127. Springer-Verlag, Berlin, 1993.

[21] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. Second edition.

[22] J. W. Lloyd and J. C. Shepherdson. Partial Evaluation in Logic Programming. *Journal of Logic Programming*, 11:217–242, 1991.

[23] L. Vieille. Recursive query processing: the power of logic. *Theoretical Computer Science*, 69:1–53, 1989.

FIXPOINT SEMANTICS OF L_λ

Maurizio Martelli*, Alessandro Messora†, Catuscia Palamidessi‡
D.I.S.I. Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, Viale Benedetto XV 3, 16132 Genova, Italy
fax: +39 – 10 – 3538028

Abstract

The language L_λ is a subset of λ-Prolog [14] which is based on a restricted β-rule, in order to simplify the unification problem. Like λ-Prolog, L_λ supports a notion of modularity and it presents higher-order features.

In this paper we propose a fixpoint semantics for L_λ which captures the notion of computed answer substitution, in the style of the S-semantics [3]. The purpose is to give a simple description of the meaning of a goal in terms of a specific interpretation, which should be a structure as elementary as possible (in [3] it is a simply set of atoms). The main difficulty here is to cope with the notion of module. The truth of a goal in a specific program will depend not only on the program, but also on the module in which the goal will be executed. The solution we propose is to define an interpretation to be a tuple of sets, each of them being associated to a particular module specified in the program.

1 Introduction

The language λ-Prolog [14] is a very rich extension of Prolog which provides higher-order programming, metaprogramming, modules, abstract data types and other useful abstractions.

Despite of embodying all these features, the language is simple and elegant, because it is based on the powerful and concise formalism of λ-calculus. Furthermore λ-Prolog supports unification, therefore it provides the possibility to solve queries.

*phone: +39 – 10 – 3538033, e_mail: *martelli@disi.unige.it*
†phone: +39 – 10 – 3538033, e_mail: *messora@disi.unige.it*
‡phone: +39 – 10 – 3538026, e_mail: *catuscia@di.unipi.it*