The Quotient of an Abstract Interpretation for Comparing Static Analyses

A. Cortesi
Univ. di Venezia
cortesi@moo.unive.it

G. Filé
Univ. di Padova
file@zenone.unipd.it

W. Winsborough
Pennsylvania St. Univ.
winsboro@cs.psu.edu

Abstract

Within the abstract interpretation approach, a data-flow analysis is specified by describing an *abstract interpretation* $\mathbf{D}=\langle D, \mu_1, .., \mu_k \rangle$, where D is a set of data-descriptions, called *abstract domain* and μ_i are the operations on D. In an abstract interpretation that expresses and thus computes several properties of the concrete values, we want to identify precisely which part is useful for computing each property. Given an abstract interpretation **D** that expresses a property P, we want to extract from the abstract domain **D** the subset $\mathcal{Q}_P(D)$ that expresses exactly the part of D that is useful for computing P. We call it the *quotient of D with respect to P*. In addition to giving more insight in the way complex abstract interpretations work, the main application of the notion of quotient is the comparison of abstract interpretations with respect to the precision with which they compute a given property.

1 Introduction

Abstract Interpretation is a general method for defining data-flow analyses [2]. In this framework, a data-flow analysis is specified by describing a domain of data-descriptions and operations on these data-descriptions that mimic the concrete operations of the language. A tuple $\mathbf{D}=\langle D, \mu_1, .., \mu_k \rangle$, where D is the set of data-descriptions (called *abstract domain*) and the μ_i are the operations on D (called *abstract operations*), is called an *abstract interpretation*. The correctness of the data-flow analysis induced by **D** is guaranteed when some safety condition holds between **D** and the *concrete interpretation* $\mathbf{C}=\langle C, o_1, .., o_k \rangle$. In this paper we will take as safety condition that:

(0) D and C are complete lattices,

(1) there is a Galois insertion between D and C (with γ and α concretization and abstraction functions, respectively), and

(2) for any $c \in C$ and for any $d \in D$, $c \sqsubseteq_C \gamma(d) \Rightarrow o_i(c) \sqsubseteq_C \gamma(\mu_i(d))$.

When conditions (0) and (1) are satisfied, we will say that D *abstracts* C, when also condition (2) holds then we say that **D** *abstracts* **C**. Weaker safety conditions are also sufficient for guaranteeing correctness [8]. However we need (0)-(2) for the purposes of the present paper.

The aim of this paper is to understand the internal workings of complex abstract interpretations. In an abstract interpretation that expresses and thus computes several properties, we want to identify which part is useful for computing each property. Let us be more precise. Consider an abstract interpretation **D** that expresses a property P. We want to extract from the abstract domain D the subset $\mathcal{Q}_P(D)$ that expresses exactly the part of D that is useful for computing P. We call it the *quotient* of D with respect to P.

In addition to giving more insight in the functionalities of complex abstract interpretations, the main application of the notion of quotient is the comparison of abstract interpretations. Assume we want to compare the relative precision of two abstract interpretations **D** and **L** in the computation of a given property P. This comparison may be impossible relying on the classical notion of Galois insertion (or connection). In fact, **D** and **L** may express information that is irrelevant for computing P and makes them incomparable using the classical formal tools. We show that the comparison can be done by considering the quotients $\mathcal{Q}_P(D)$ and $\mathcal{Q}_P(L)$. Being able to perform such sophisticated comparisons is obviously useful for choosing the best available abstract interpretation for a particular purpose. Such comparisons are also useful when combining different abstract interpretations in order to obtain a more powerful one, as is suggested in [8] and [1]. By appropriate comparisons one may check whether such combinations of **D** and **L** are worthwhile at all and also one can "tune" the combination by allowing it only for those properties P for which it may turn out to be profitable: those for which $\mathcal{Q}_P(D)$ and $\mathcal{Q}_P(L)$ are incomparable.

The main achievements of the paper are described below.

(a) For any abstract interpretation **D** we describe how to obtain the quotient $\mathcal{Q}_P(D)$. We also give a condition that guarantees that $\mathcal{Q}_P(D)$ is a complete lattice that abstracts D.

(b) We show that if **D** and **L** are optimal for the computation of a property P and if $\mathcal{Q}_P(D)$ abstracts $\mathcal{Q}_P(L)$, then **L** is at least as precise as **D** for computing P. Moreover, if $\mathcal{Q}_P(D)$ abstracts $\mathcal{Q}_P(L)$ strictly, (i.e., $\mathcal{Q}_P(L)$ does not abstract $\mathcal{Q}_P(D)$), then **L** is strictly more precise than **D** for computing P.

(c) As an application of this theory, we compare two abstract interpretations that are well-known for the analysis of Prolog programs: **Prop**, [11] and [4], and **Sharing**, [10]. Although both these analyses compute the property of variable groundness, they are not directly comparable (i.e. neither one abstracts the other) because **Prop** computes also possible equivalence through disjunctions and **Sharing** computes also variable sharing. We compare the (relative) precision of these two analyses in the computation of variable groundness (GR) and we show that

- both **Prop** and **Sharing** are optimal and compute GR optimally. The former result is already shown in [7], whereas the latter is new.
- $\mathcal{Q}_{GR}$(Prop) is Prop itself, whereas $\mathcal{Q}_{GR}$(Sharing) is the set Def [9] of (the equivalence classes of) definite formulas , i.e., conjunctions of formulas of

the form: $\wedge S \rightarrow \wedge T$, where S and T are sets of variables. From this it is immediate to see that Def strictly abstracts Prop. Thus, from (b) above it follows that Prop is strictly more precise than Sharing for the computation of variable groundness.

The present work continues and improves on the theory we developed in [5]. In that paper, the problem of isolating the part of an abstract domain that is useful for computing a property P was solved in a less satisfactory way. There, we observed that, in order to compare D and L, it was necessary to define a reference domain R and then compare the workings of D and L projected on R. Clearly, R plays the same role as $\mathcal{Q}_P(D)$ and $\mathcal{Q}_P(L)$ in the present approach, but in [5] we did not have any systematic way to obtain it. Its definition was only based on the insight one had over D and L. In addition to this, the results we obtain with the present approach are stronger than those in [5] in the sense that the notion of "being more precise for computing P" we use here is strictly stronger than that used in [5]. Finally, it is important to remark that, even though we have applied it to the comparison of two abstract interpretations for the data-flow analysis of Prolog programs, the approach presented in this paper is completely general as it does not depend on the programming language considered.

The paper is organized as follows. Section 2 contains the preliminary definitions. The definition of $\mathcal{Q}_P$ is given in Section 3 together with the general result mentioned in (b) above. The application of the theory to **Prop** and **Sharing**, mentioned in (c) above, is described in Section 4.

2 Preliminaries

This section consists of three parts. The first introduces some classical notions of abstract interpretation theory and some known results. The second part contains the definitions of two types of optimality. The final part introduces the criterion for comparing abstract interpretations that will be used in the rest of the paper.

2.1 Composition of Galois Insertions

Definition 2.1 (Galois insertion) *Let C and D be complete lattices and $\gamma_{DC} : D \longrightarrow C$ and $\alpha_{CD} : C \longrightarrow D$ be monotonic functions. The 4-tuple $G_{CD} = (\gamma_{DC}, C, D, \alpha_{DC})$ is a* Galois insertion *if $\forall c \in C : c \sqsubseteq_C \gamma_{DC}(\alpha_{CD}(c))$ and $\forall d \in D : d = \alpha_{CD}(\gamma_{DC}(d))$.*

The last condition implies that γ_{DC} is one-to-one. The functions γ_{DC} and α_{CD} are called the concretization and the abstraction function, respectively. The domain D is said to *abstract* C.

A function's domain and range are indicated by subscripts: ε_{XY} is a function from X to Y. The ordering and the least upper bound operator defined in X are denoted by $\sqsubseteq_X$ and $\sqcup_X$, respectively.

Definition 2.2 (Interpretation) *An interpretation is a tuple $\mathbf{D} = \langle D, \mu_D \rangle$ where D is a complete lattice, μ_D is a monotone, continuous function of type $D \longrightarrow D$.*

In general, an interpretation contains more than one operation, and the operations may have arity greater than one and may take some other arguments besides elements of D. For the sake of clarity we consider the simplest setting, as the generalization of definitions and results is immediate.

Definition 2.3 (Abstraction) *An interpretation $\mathbf{D} = \langle D, \mu_D \rangle$ abstracts an interpretation $\mathbf{C} = \langle C, \mu_C \rangle$ if*

1. *there is a Galois insertion $(\gamma_{DC}, C, D, \alpha_{CD})$ and*
2. *$\forall c \in C : \forall d \in D : c \sqsubseteq_C \gamma_{DC}(d) \Rightarrow \mu_C(c) \sqsubseteq_C \gamma_{DC}(\mu_D(d))$.*

We say that **D** *properly abstracts* **C** *if* **D** *abstracts* **C** *but* **C** *does not abstract* **D**

An abstract interpretation is intended to report information about a program's execution behavior. When **L** abstracts **D** we know that the analysis induced by **D** is at least as informative as the analysis induced by **L**. In the rest of the paper we denote domains by capital letters C, D, L, R possibly subscripted, and we denote interpretations by boldface capital letters **C**,**D**,**L**,**R**.

It is well known that if **R** abstracts **D** and **D** abstracts **C**, then **R** abstracts **C**. The proofs of the propositions listed below can be found in [5, 6].

Proposition 2.4 (Same-Order Composition) *Let G_{DR} and G_{CD} be Galois insertions. Then $(\gamma_{RC}, C, R, \alpha_{CR})$ is a Galois insertion, where $\gamma_{RC} = \gamma_{DC} \circ \gamma_{RD}$ and $\alpha_{CR} = \alpha_{DR} \circ \alpha_{CD}$.*

Remark 2.5 *Observe that from the previous Proposition, we obtain in particular that*

i) $\forall c \in C \quad \alpha_{CR}(c) = \alpha_{DR}(\alpha_{CD}(c))$;
ii) $\forall d \in D \quad \gamma_{DC}(d) \sqsubseteq_C \gamma_{RC}(\alpha_{DR}(d))$;
iii) $\forall d \in D \quad d = \gamma_{RD}(r) \Leftrightarrow \gamma_{DC}(d) = \gamma_{RC}(r)$.

Proposition 2.6 (Opposite-Order Composition) *Assume that G_{CL} and G_{CD} are Galois insertions. Let $\varepsilon_{DL} = \alpha_{CL} \circ \gamma_{DC}$ and $\varepsilon_{LD} = \alpha_{CD} \circ \gamma_{LC}$. The following holds:*

i) *ε_{DL} and ε_{LD} are monotone;*

ii) $\forall d \in D, \forall l \in L \quad \varepsilon_{LD}(\varepsilon_{DL}(d)) \sqsupseteq_D d$ *and* $\varepsilon_{DL}(\varepsilon_{LD}(l)) \sqsupseteq_L l$.

Proposition 2.7 *Assume that G_{CL} and G_{CD} are Galois insertions. The following conditions are equivalent.*

i) *$G_{DL} = (\alpha_{CD} \circ \gamma_{LC}, D, L, \alpha_{CL} \circ \gamma_{DC})$ is a Galois insertion;*

ii) $\gamma_{LC}(L) \subseteq \gamma_{DC}(D)$;

iii) $\forall c_1, c_2 \in C. \, \alpha_{CD}(c_1) = \alpha_{CD}(c_2) \Rightarrow \alpha_{CL}(c_1) = \alpha_{CL}(c_2)$.

Remark 2.8 *Let G_{CD} and G_{CL} be Galois insertions. If G_{DL} of Proposition 2.7 is a Galois insertion, it is said to be coherent with respect to C. In this case, the following facts hold:*

i) $\forall \ell \in L\ \exists d \in D : \gamma_{LC}(\ell) = \gamma_{DC}(d)$.

It is easy to prove that for such ℓ and d it holds that $\gamma_{LD}(\ell) = d$.

ii) $\gamma_{LC} = \gamma_{DC} \circ \gamma_{LD}$ *and* $\alpha_{CL} = \alpha_{DL} \circ \alpha_{CD}$.

2.2 Optimalities

As usual, an abstract interpretation is optimal if it mimics the concrete one in the best possible way. We also introduce a weaker notion in which we project the result of the operation on a more abstract domain.

Definition 2.9 *Consider the interpretations $\mathbf{D}=\langle D, \mu_D\rangle$ and $\mathbf{C}=\langle C, \mu_C\rangle$. Assume that $\mathbf{D}$ abstracts $\mathbf{C}$. $\mathbf{D}$ is optimal if* $\forall d \in D : \mu_D(d) = \alpha_{CD}(\mu_C(\gamma_{DC}(d)))$.
Let now R be a domain which abstracts D, and let $\alpha_{DR} = \alpha_{CR} \circ \gamma_{DC}$. $\mathbf{D}$ is R-optimal if $\forall d \in D : \alpha_{DR}(\mu_D(d)) = \alpha_{CR}(\mu_C(\gamma_{DC}(d)))$.

Lemma 2.10 *Consider the interpretations $\mathbf{D}=\langle D, \mu_D\rangle$, $\mathbf{C}=\langle C, \mu_C\rangle$. Assume that $\mathbf{D}$ abstracts $\mathbf{C}$, and that R abstracts D coherently with respect to C. If $\mathbf{D}$ is optimal then $\mathbf{D}$ is also R-optimal.*

proof: $\forall d \in D$

$$\begin{array}{rrcll} & \mu_D(d) & = & \alpha_{CD}(\mu_C(\gamma_{DC}(d))) & \text{as } \mathbf{D} \text{ is optimal} \\ \Rightarrow & \alpha_{DR}(\mu_D(d)) & = & \alpha_{DR}(\alpha_{CD}(\mu_C(\gamma_{DC}(d)))) & \text{applying } \alpha_{DR} \text{ to both sides} \\ \Rightarrow & \alpha_{DR}(\mu_D(d)) & = & \alpha_{CR}(\mu_C(\gamma_{DC}(d))) & \text{by Remark 2.8, as } R \text{ abstracts } D. \ \square \end{array}$$

Observe that the notion of R-optimality is a generalization of the notion of optimality. In fact $\mathbf{D}$ is optimal iff $\mathbf{D}$ is D-optimal.

2.3 General Comparison Criterion

We compare the precision of two interpretations with respect to a domain P according to the following intuitive idea. $\mathbf{L}$ isat least as precise as $\mathbf{D}$ with respect to P if every sequence of concrete operations is better or equally approximated by $\mathbf{L}$ than by $\mathbf{D}$ when considering only the information representable in P.

Definition 2.11 (comparison criterion) *Let $\mathbf{D} = \langle D, \mu_D\rangle$ and $\mathbf{L} = \langle L, \mu_L\rangle$ be interpretations abstracting $\mathbf{C} = \langle C, \mu_C\rangle$. Let P be a domain abstracting both D and L.*

- *$\mathbf{L}$ is at least as precise as $\mathbf{D}$ with respect to P if*

$$\forall c \in C, \forall i \geq 0 :\ \alpha_{LP}(\mu_L^i(\alpha_{CL}(c))) \sqsubseteq_P \alpha_{DP}(\mu_D^i(\alpha_{CD}(c))).$$

- *$\mathbf{L}$ is strictly more precise than $\mathbf{D}$ with respect to P if $\mathbf{L}$ is at least as precise as $\mathbf{D}$ but the viceversa does not hold.*

3 The Quotient of an Interpretation

In this section we show how to extract from an abstract domain exactly the part which is useful for computing some property. The obtained domain is called *quotient* of the original one. Two subsections follow the definition of quotient. In the first, some properties of the quotients are proven. In the second one, we show the relevance of this notion for comparing the precision with which two interpretations compute a given property.

Throughout this section, we always assume that G_{CP}, G_{CD}, G_{DP} are Galois insertions (with $\alpha_{DP} = \alpha_{CP} \circ \gamma_{DC}$ and $\gamma_{PD} = \alpha_{CD} \circ \gamma_{PC}$), and that the interpretation $\mathbf{D} = \langle D, \mu_D\rangle$ abstracts the interpretation $\mathbf{C} = \langle C, \mu_C\rangle$.

Definition 3.1 (associated relation) *The equivalence relation $\mathbf{r}_P$ on D associated to P is defined by*

$$(d_1, d_2) \in \mathbf{r}_P \quad \Leftrightarrow \quad \forall i \geq 0 : \alpha_{DP}(\mu_D^i(d_1)) = \alpha_{DP}(\mu_D^i(d_2))$$

The relation $\mathbf{r}_P$ is additive if given a set of pairs $S = \{(a,b) \mid (a,b) \in \mathbf{r}_P\}$, and let $S_1 = \{a \mid (a,b) \in S\}$ and $S_2 = \{b \mid (a,b) \in S\}$, also $(\sqcup_D S_1, \sqcup_D S_2) \in \mathbf{r}_P$.

Definition 3.2 (Quotient) *We denote by $[d]_P$ the set $\{d' \in D \mid (d,d') \in \mathbf{r}_P\}$. The quotient of D with respect to P is the set $\mathcal{Q}_P(D)$ defined by:*

$$\mathcal{Q}_P(D) = \{[d]_P \mid d \in D\}$$

The quotient $\mathcal{Q}_P(D)$ is ordered by: $[d_1]_P \sqsubseteq_Q [d_2]_P \Leftrightarrow \sqcup_D [d_1]_P \sqsubseteq_D \sqcup_D [d_2]_P$.

Lemma 3.3 *If the relation $\mathbf{r}_P$ on D associated to P is additive, then the set $\mathcal{Q}_P(D)$ enjoys the following properties:*

(1) $\forall d \in D : (\sqcup_D [d]_P, d) \in \mathbf{r}_P$;

(2) $\forall d_1, d_2 \in D : d_1 \sqsubseteq_D d_2 \Rightarrow [d_1]_P \sqsubseteq_Q [d_2]_P$.

Theorem 3.4 *If the associated relation $\mathbf{r}_P$ is additive, the following facts hold.*

i) *$\mathcal{Q}_P(D)$ is a complete lattice;*

ii) *$\mathcal{Q}_P(D)$ abstracts D;*

iii) *P abstracts $\mathcal{Q}_P(D)$;*

iv) *The abstractions of points ii) and iii) are coherent with respect to C.*

3.1 Comparison of Quotients

The results of this subsection show the important role that the notion of quotient plays in the comparison of two abstract interpretations.

Let in what follows **D** and **L** be such that their quotients $R_1 = \mathcal{Q}_P(D)$ and $R_2 = \mathcal{Q}_P(L)$ satisfy all the properties stated in Theorem 3.4. Assume also that **D** and **L** are, respectively, R_1- and R_2-optimal. Under these assumptions, we prove the following two results.

Theorem 3.5 *If R_1 abstracts R_2 coherently with respect to C, then* **L** *is at least as precise as* **D** *for computing P.*

proof: Consider any sequence π_D of operations of **D**. From the fact that **D** is R_1-optimal and from the construction of the quotient R_1, it is true that for any $d \in D$, the computation $\pi_D(d)$ can be "read" over R_1, as far as the P-information is concerned. More precisely, assume that d_i is the result after the first $i \geq 0$ steps of the computation $\pi_D(d)$, and let $t_i = \alpha_{DR_1}(d_i)$, then, for each i, $\alpha_{DP}(d_i) = \alpha_{R_1P}(t_i)$. A similar fact holds for any computation π_L on **L**. For such a π_L, let $l_0, l_1, .., l_i$ be the intermediate results and $k_0, .., k_i$ be the corresponding values in R_2.

We will use the above fact and the notation introduced for explaining it in the sequel of the proof. Let for any concrete value $c \in C$, $d_0 = \alpha_{CD}(c)$ and $l_0 = \alpha_{CL}(c)$. Let also π_D and π_L be sequences of corresponding operations of **D** and **L**. In what follows the d_i, t_i, l_i, and k_i are in the relation explained above. In order to prove the result, it suffices to show the following fact:

$$(\star) \qquad \forall i \geq 0,\ \gamma_{R_1C}(t_i) \sqsupseteq_C \gamma_{R_2C}(k_i).$$

In fact, from $(\star)$ it follows that $\alpha_{R_2R_1}(k_i) \sqsubseteq_{R_1} t_i$ and thus, as α_{R_1P} is monotone, $\alpha_{R_2P}(k_i) = \alpha_{R_1P}(\alpha_{R_2R_1}(k_i)) \sqsubseteq_P \alpha_{R_1P}(t_i)$.

basis Let us first consider $i = 0$. We want to show that

$$(1) \qquad \gamma_{R_1C}(t_0) \sqsubseteq_C \gamma_{R_2C}(k_0).$$

By Remark 2.5, $t_0 = \alpha_{CR_1}(c)$ and $k_0 = \alpha_{CR_2}(c)$. Since R_1 abstracts R_2, by Lemma 2.7, there exists $k \in R_2$ such that $\gamma_{R_2C}(k) = \gamma_{R_1C}(t_0)$ and thus, $\gamma_{R_2C}(k) \sqsupseteq_C c$ which implies that $k \sqsupseteq_{R_2} k_0$. Thus (1) holds.

step Let us now prove $(\star)$ for $i > 0$. We want to show that

$$(2)\ \gamma_{R_1C}(t_{i-1}) \sqsupseteq_C \gamma_{R_2C}(k_{i-1}) \quad \Rightarrow \quad (3)\ \gamma_{R_1C}(t_i) \sqsupseteq_C \gamma_{R_2C}(k_i).$$

By the assumption of R_1- and R_2-optimality of **D** and **L**, we know that

$$t_i = \alpha_{CR_1}(\mu_C(\gamma_{R_1C}(t_{i-1})))$$

where μ_C is the concrete operation corresponding to the i-th operation of π_D and π_L. A similar relation holds for k_i.

By assumption (2) and the monotonicity of μ_C it follows that

$$\mu_C(\gamma_{R_1C}(t_{i-1})) \sqsupseteq_C \mu_C(\gamma_{R_2C}(k_{i-1}))$$

and thus, by the same reasoning used for the case $i = 0$, we have that (3) holds. □

Theorem 3.6 *If R_1 properly abstracts R_2 (coherently with respect to C) then* **L** *is strictly more precise than* **D** *for computing P.*

proof: By Proposition 2.7(*ii*), there exists $k_0 \in R_2$ s.t. $\gamma_{R_2C}(k_0) \notin \gamma_{R_1C}(R_1)$. Let $c_0 = \gamma_{R_2C}(k_0)$. We will show that there are corresponding computations π_D and π_L of operations of **D** and **L**, respectively, such that

$$(1) \quad \alpha_{DP}(\pi_D(\alpha_{CD}(c_0))) \sqsupseteq_P \alpha_{LP}(\pi_L(\alpha_{CL}(c_0)))$$

Let $l_0 \in L$ be such that $\gamma_{LC}(l_0) = c_0$. This l_0 exists because R_2 abstracts L. Let us now consider $d_0 = \alpha_{CD}(c_0)$. By the assumptions made on R_1 and R_2, we know that if $t_0 = \sqcup[d_0]_P$ the following is true for every sequence s_D of operations of **D**

$$(2) \quad \alpha_{DP}(s_D(d_0)) = \alpha_{DP}(s_D(t_0)).$$

Consider now $\hat{c} = \gamma_{DC}(t_0)$. By definition of t_0, $\hat{c} \in \gamma_{R_1C}(R_1)$. Thus, $\hat{c}$ is the least element of $\gamma_{R_1C}(R_1)$ greater or equal to c_0. Thus $\hat{c} \sqsupseteq_C c_0$ because $c_0 \notin \gamma_{R_1C}(R_1)$, by hypothesis. Let now $l = \alpha_{CL}(\hat{c})$. It is easy to see that $\gamma_{LC}(l) = \hat{c}$, using the property of Galois insertion's and the fact that $\hat{c} \in \gamma_{R_1C}(R_1)$ and $\gamma_{R_1C}(R_1) \subseteq \gamma_{R_2C}(R_2) \subseteq \gamma_{LC}(L)$. Moreover, since $\gamma_{LC}(l) = \hat{c} \sqsupseteq_C c_0 = \gamma_{LC}(l_0)$ it is true that $l \sqsupseteq_L l_0$. Using the fact that $\hat{c}, c_0 \in \gamma_{R_2C}(R_2)$, by Remark 2.5(*iii*), it is also true that $[l]_P \neq [l_0]_P$.

It suffices now to make the following two observations:

(3) By the previous theorem and point (2), for all corresponding sequences s_D and s_L of operations of **D** and **L**, it is true that:

$$\alpha_{DP}(s_D(\alpha_{CD}(c_0)) = \alpha_{DP}(s_D(t_0)) = \alpha_{DP}(s_D(\alpha_{CD}(\hat{c}))) \sqsupseteq_P \alpha_{LP}(s_L(\alpha_{CL}(\hat{c}))).$$

(4) Since $l \sqsupseteq_L l_0$ and $[l]_P \neq [l_0]_P$, by definition of quotient, there exists a sequence π_L of operations of **L** such that $\alpha_{LP}(\pi_L(l)) \neq \alpha_{LP}(\pi_L(l_0))$. By the monotonicity of the operations of **L**, it must be that $\alpha_{LP}(\pi_L(l)) \sqsupseteq_P \alpha_{LP}(\pi_L(l_0))$.

Let π_D be the sequence of operations of **D** corresponding to π_L of point (4). Substituting in (3) s_D and s_L with π_D and π_L, respectively, and merging it with (4), we obtain:

$$\begin{array}{rcl} \alpha_{DP}(\pi_D(\alpha_{CD}(c_0))) & = & \alpha_{DP}(\pi_D(\alpha_{CD}(\hat{c}))) \\ & \sqsupseteq_P & \alpha_{LP}(\pi_L(\alpha_{CL}(\hat{c}))) \\ & \sqsupseteq_P & \alpha_{LP}(\pi_L(l_0)) \\ & = & \alpha_{LP}(\pi_L(\alpha_{CL}(c_0))) \end{array}$$

that proves the theorem. □

4 Applications

We apply the theory developed in the previous section comparing two well known abstract interpretations for logic programming, namely **Prop** [11] and **Sharing** [10]. This section is organized as follows. After some preliminary definitions concerning substitutions, in subsection 4.2 we introduce the concrete interpretation **Rsub**. In subsections 4.3 and 4.4 we describe the two abstract interpretations we wish to compare (**Prop** and **Sharing**), and the domain GR representing groundness. The quotients $\mathcal{Q}_{GR}$(Sharing) and $\mathcal{Q}_{GR}$(Prop), whee GR is a domain representing groundness are described in subsection 4.5. The main result of the application part is shown in subsection 4.5.4, where using Theorem 3.5 and 3.6 we show that **Prop** is strictly more precise than **Sharing** with respect to groundness computation.

It is important to observe that the interpretations that will be considered in this section are complete for the analysis of logic programming, in the sense that they consider all the necessary operations, namely least upper bound, projection and unification. Notice also that these interpretations extend the simple notion of Definition 2.3, as already announced. In order to guarantee this completeness, and in particular to formally deal with projections, the concrete and abstract interpretations have a special form. Each element is a pair, where the second component explicitely gives the variables about which the first component is supposed to provide information.

4.1 Preliminaries

Let $\mathbf{V}$ be a countable set of variables. $FP(\mathbf{V})$ denotes the set of finite subsets of variables of $\mathbf{V}$. Let $\mathbf{G}$ be an alphabet of function symbols and let $\mathbf{T}_{\mathbf{V},\mathbf{G}}$ denote the set of finite terms over $\mathbf{V}$ and $\mathbf{G}$. A *substitution* σ is a function in $\mathbf{V} \longrightarrow \mathbf{T}_{\mathbf{V},\mathbf{G}}$ such that $\sigma(x) \neq x$ for only a finite number of variables x. The *set of support* of σ is given by $supp(\sigma) = \{x \mid \sigma(x) \neq x\}$. The *variable range* of σ is given by $var\text{-}range(\sigma) = \bigcup\{Var(\sigma x) \mid x \in supp(\sigma)\}$, where $Var(t)$ denotes the set of variables occurring in t. The set of variables occurring in σ is given by $Var(\sigma) = supp(\sigma) \cup var\text{-}range(\sigma)$. A substitution is typically specified by listing its non-trivial bindings. So $\sigma = \{x/\sigma x | x \in supp(\sigma)\}$.

Consider two substitutions σ_1 and σ_2. If there exists ϑ such that $\sigma_2 = \vartheta \circ \sigma_1$, then σ_1 is *more general* than σ_2, which we write $\sigma_2 \trianglelefteq \sigma_1$. We call σ_2 an instance of σ_1.

Let E be a set of term equations. If σ makes $\sigma(t_1)$ syntactically identical to $\sigma(t_2)$ for each $(t_1 = t_2) \in E$, σ is called a *unifier* of E. The *most general unifier* of E (denoted mgu(E) is a unifier of E that is more general than all other unifiers. A set of equations E is in *solved form* if it has the form $\{x_1 = t_1, \ldots, x_n = t_n\}$ where each x_i is a distinct variable occurring in none of the terms t_j. Given a set of equations $E = \{x_1 = t_1, \ldots, x_n = t_n\}$ in solved form, the substitution $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$ is an idempotent *mgu* of E; we denote E by $Eq(\sigma)$.

We are particularly interested in idempotent substitutions because of their correspondence with sets of equations in solved form. We write *Subst* for the set of idempotent substitutions. Although *Subst* is not closed under composition, in a step of the execution of a logic program in which $\vartheta \circ \sigma$ is constructed, it is always the

case that, $var\text{-}range(\vartheta) \cap supp(\sigma) = \emptyset$, which provided that ϑ and σ are idempotent ensures that $\vartheta \circ \sigma$ is also idempotent.

As we will consider sequences of concrete/abstract operations of "real" domains, i.e. containing not only unary operations, as it was assumed in Definition 2.2 for the sake of simplicity, it is necessary to make precise this notion for any set of operations. Assume to have an interpretation $\mathbf{D} = \langle D, \mu_1, \ldots, \mu_k \rangle$. A *derived operator* over D is a term t constructed using the symbols in $\mu_1, \ldots, \mu_k$, the values in D, values of any other domain that may be required by the operations (like substitutions in the domains described below), and exactly one variable z, The derived operator t is assumed to be a place where a value of D is expected. The following example illustrates this notion for $\mathbf{Z} = \langle Z, +, * \rangle$, where Z represents the set of all integers completed with top and bottom elements.

Example 4.1 *A derived operator for* $\mathbf{Z}$ *is* $t = +(*(z, 3), +(5, 1))$.

Clearly, a derived operator t is a functions $t : D \longrightarrow D$. Intuitively, the result of the function t for a given input $d \in D$ is obtained by evaluating $t(z \leftarrow d)$, interpreting the function symbols in t according to $\mathbf{D}$. Such result is indicated by $t(d)$. In the above example, $t(0) = 6$ and $t(2) = 12$.

4.2 The "concrete" interpretation Rsub = $\langle$Rsub, $\sqcup_{Rs}, \pi_{Rs}, \mathbf{U}_{Rs}\rangle$

Domain The "concrete" domain we consider is the domain of computation of logic programming. The set Rsub=$[\wp(Subst) \times FP(\mathbf{V})] \cup \{\top_{Rs}, \bot_{Rs}\}$. Rsub stands for restricted substitutions. The partial order of Rsub is defined as follows. $\top_{Rs}$ is the largest element, $\bot_{Rs}$ is the smallest one, for any other two elements $[\Sigma_1, U_1]$ and $[\Sigma_2, U_2]$ of Rsub, $[\Sigma_1, U_1] \sqsubseteq_{Rs} [\Sigma_2, U_2]$ iff $U_1 = U_2$ and $\Sigma_1 \subseteq \Sigma_2$. Rsub is a complete lattice.

Least upper bound The operation $\sqcup_{Rs}$, which produces the least upper bound of two elements of Rsub, is as follows: for any $k \in$ Rsub, $\top_{Rs} \sqcup_{Rs} k = \top_{Rs}$, $\bot_{Rs} \sqcup_{Rs} k = k$, for the other elements,

$$[\Sigma_1, U_1] \sqcup_{Rs} [\Sigma_2, U_2] = \begin{cases} [\Sigma_1 \cup \Sigma_2, U_1] & \text{if } U_1 = U_2 \\ \top_{Rs} & \text{otherwise} \end{cases}$$

Projection The concrete projection π_{Rs} is as follows:

$$\begin{array}{l} \pi_{Rs} : \text{Rsub} \times FP(\mathbf{V}) \longrightarrow \text{Rsub} \\ ([\Sigma, U_1], U_2]) \longmapsto [\Sigma, U_1 \cap U_2] \end{array}$$

Unification In order to define the concrete unification $\mathbf{U}_{Rs}$, it is convenient to introduce first the following function $\mathbf{u}_{Rs}$:

$$\begin{array}{rccl} \mathbf{u}_{Rs} : & Subst \times Subst & \longrightarrow & Subst \\ & (\sigma, \delta) & \longmapsto & mgu(Eq(\sigma) \cup Eq(\delta)) \end{array}$$

$$\begin{array}{rlll} \mathbf{U}_{Rs}: & \wp(\mathtt{Rsub}) \times \mathit{Subst} & \longrightarrow & \wp(\mathtt{Rsub}) \\ & ([\Sigma, U], \delta) & \longmapsto & [\{\mathbf{u}_{Rs}(\sigma, \delta) \mid \sigma \in \Sigma\}, U \cup \mathit{Var}(\delta)] \end{array}$$

4.3 The interpretation Prop=$\langle$Prop, $\sqcup_{Pr}, \pi_{Pr}, \mathbf{U}_{Pr}\rangle$

Propositional Formulas For any set of variables $U \in FP(\mathbf{V})$, the set of propositional formulas constructed over the variables $\mathbf{V}$ and the logical connectives in the set $\{\wedge, \vee, \leftrightarrow\}$ is denoted by Ω_U. The propositional constants T and F (for *true* and *false*) are not included unless otherwise indicated.

We are interested in the lattice obtained by taking the quotient with respect to logical equivalence. To avoid burdensome notation, we simply write f for the class of formula equivalent to f. Notice that for any $U \in FP(\mathbf{V})$, $\Omega_U \cup \{\mathsf{F}\}$ is a complete lattice with least upper bound $\vee$ (logical disjunction) and greatest lower bound $\wedge$ (logical conjunction).

Formulas to represent substitutions The groundness and variable equivalence properties of a substitution are preserved under instantiation of the substitution: if σ grounds x then any $\sigma' \trianglelefteq \sigma$ grounds x; if two sets of variables S_1 and S_2 are equivalent with respect to σ, then S_1 and S_2 are equivalent also with respect to any $\sigma' \trianglelefteq \sigma$. Thus, the variable equivalence of a substitution is related to a propositional formula by examining the set of variables made ground by each of the substitution's instances. In order to formalize and clarify this idea, an auxiliary function is introduced that maps a substitution to an assignment, that assigns the value true to the variables that the substitution grounds, [11].

$$\mathit{assign}\ \sigma\ x = \mathit{true}\ \mathit{iff}\ \sigma \text{ grounds } x$$

Now we are in position to describe the domain Prop.

Domain Prop contains the bottom and the top elements denoted $\bot_{Pr}$ and $\top_{Pr}$, respectively, and the set: $\{[f, U] : f \in \Omega_U \cup \{\mathsf{F}\}, U \in FP(\mathbf{V})\}$. Prop is partially ordered as follows: $\top_{Pr}$ is the largest element and $\bot_{Pr}$ is the smallest one; for the other elements, $[f_1, U_1] \sqsubseteq_{Pr} [f_2, U_2]$ iff $U_1{=}U_2$ and $f_1 \models f_2$.

Concretization and Abstraction Functions The relation between Rsub and Prop is expressed by two functions: the concretization function, denoted $\gamma_{Pr\text{-}Rs}$, and the abstraction one, denoted $\alpha_{Rs\text{-}Pr}$.

$$\gamma_{Pr\text{-}Rs}([f, U]) = [\Sigma,\ U], \text{ where } \Sigma = \{\sigma \in \mathit{Subst} : \forall \sigma' \trianglelefteq \sigma\ .\ \mathit{assign}\ \sigma' \models f\}$$
$$\alpha_{Rs\text{-}Pr}([\Sigma, U]) = \sqcup_{Pr}\{[f, U] \mid \gamma_{Pr\text{-}Rs}([f, U]) \sqsubseteq_{Rs} [\Sigma, U]\}$$

Least Upper Bound The least upper bound is defined trivially for $\bot_{Pr}$ and $\top_{Pr}$. For any two elements $[f_1, U_1]$ and $[f_2, U_2]$,

$$[f_1, U_1] \sqcup_{Pr} [f_2, U_2] = \begin{cases} [f_1 \vee f_2, U_1] & \text{if } U_1 = U_2 \\ \top_{Pr} & \text{otherwise} \end{cases}$$

Example 4.2 *Let $U = \{x, y, z, v\}$. The pair $[x \wedge (y \leftrightarrow z), U]$ is an element of* Prop *which represents substitutions σ such that for every instance σ' of σ the term $\sigma'(x)$ is ground, and such that $\sigma'(y)$ is a ground term if and only if also $\sigma'(z)$ is. In particular $\sigma_1 = \{x/a, y/b, z/c\}$ and $\sigma_2 = \{x/b, y/w, z/w, u/v\}$ satisfy this property. Thus, $[\{\sigma_1, \sigma_2\}, U] \sqsubseteq_{Rs} \gamma_{Pr\text{-}Rs}([x \wedge (y \leftrightarrow z), U])$.*

Projection and Unification on Prop are obtained by means of existential quantification and logical conjunction, respectively. The proof of the following theorem can be found in [7].

Theorem 4.3 *The operations $\mathbf{U}_{Pr}, \pi_{Pr}$ and $\sqcup_{Pr}$, corresponding to $\mathbf{U}_{Rs}, \pi_{Rs}$ and $\sqcup_{Rs}$ respectively, are optimal with respect to the concrete interpretation* Rsub.

4.4 The interpretation Sharing = $\langle$Sharing, $\sqcup_{Sh}, \pi_{Sh}, \mathbf{U}_{Sh}\rangle$

Domain The abstract domain Sharing proposed by Jacobs and Langen in [JaLa 89] in order to represent variable aliasing, covering, and groundness is defined by

$$\mathtt{Sharing} = [\wp(\wp(\mathbf{V})) \times FP(\mathbf{V})] \cup \{\top_{Sh}, \bot_{Sh}\}.$$

Sharing is partially ordered: $\top_{Sh}$ is the largest element and $\bot_{Sh}$ is the smallest one; for the other elements, $[A_1, U_1] \sqsubseteq_{Sh} [A_2, U_2]$ iff $U_1{=}U_2$ and $A_1 \subseteq A_2$.

Concretization and Abstraction Functions Let $\mathit{occ}(\sigma, y) = \{z \in \mathit{supp}(\sigma) \mid y \in \mathit{Var}(\sigma z)\} \cup \{y\}$. The abstraction and concretization functions between Sharing and Rsub are defined, on nontrivial elements by

$$\alpha_{Rs\text{-}Sh}(c) = \begin{cases} \text{if } c = [\{\sigma\}, U] \text{ then } [A_1 \cup A_2, U] \text{ where} \\ \qquad A_1 = \{\mathit{occ}(\sigma, y) \cap U \mid y \in \mathit{var\text{-}range}(\sigma)\} \\ \qquad A_2 = \{\{y\} \mid y \in (U - \mathit{Var}(\sigma))\} \\ \text{if } c = [\Sigma, U] \text{ then } \sqcup_{Sh}\{\alpha_{Rs\text{-}Sh}([\{\sigma\}, U]) \mid \sigma \in \Sigma\} \end{cases}$$
$$\gamma_{Sh\text{-}Rs}([A, U]) = \sqcup_{Rs}\{[\Sigma, U] \mid \alpha_{Rs\text{-}Sh}([\Sigma, U]) \sqsubseteq_{Sh} [A, U]\}$$

An element of the first component of $\alpha_{Rs\text{-}Sh}([\{\sigma\}, U])$ is a set of variables that under σ share the same variable. A variable of U is ground in $[A, U]$ iff it does not appear in any set of the abstract state. Two variables of U are independent in σ iff there is no element of $\alpha_{Rs\text{-}Sh}([\{\sigma\}, U])$ that contains both of them.

Example 4.4 *For instance, let $U = \{x, y, z, v\}$, the pair $[\{\{y, z\}, \{u, y, z\}\}, U]$ is an element of* Sharing *which represents substitutions σ such that $\sigma(x)$ is ground, and such that $\sigma(y)$ and $\sigma(z)$ may share a common variable, and so $\sigma(y)$, $\sigma(z)$ and $\sigma(u)$. In particular $\sigma_1 = \{x/a, y/b, z/c\}$ and $\sigma_2 = \{x/b, y/w, z/w, u/w\}$ satisfy this property. Thus, $[\{\sigma_1, \sigma_2\}, U] \sqsubseteq_{Rs} \gamma_{Sh\text{-}Rs}([\{\{y, z\}, \{u, y, z\}\}, U])$.*

Lemma 4.5 *Let $\gamma_{Sh\text{-}Rs}([A, U]) = [\Sigma, U]$. Then*

$$\Sigma = \{\sigma \in \mathit{Subst} \mid \forall x, y \in U : (\mathit{Var}(\sigma x) \cap \mathit{Var}(\sigma y) \neq \emptyset \Rightarrow \exists H \in A : \{x, y\} \subseteq H)\}$$

Operations The least upper bound of any two elements $[A_1, U_1]$ and $[A_2, U_2]$ is defined by

$$[A_1, U_1] \sqcup_{Sh} [A_2, U_2] = \begin{cases} [A_1 \cup A_2, U_1] & \text{if } U_1 = U_2 \\ \top_{Sh} & \text{otherwise} \end{cases}$$

The projection π_{Sh} is obtained by intersection and the abstract unification is a sequence of set operations involving union and closures. The formal definition of $\mathbf{U}_{Sh}$ and the proof of the following theorem are showed in the Appendix. Notice that the results of Theorem 4.6 are new.

Theorem 4.6 *The operations $\mathbf{U}_{Sh}, \pi_{Sh}$ and $\sqcup_{Sh}$, corresponding to $\mathbf{U}_{Pr}, \pi_{Pr}$ and $\sqcup_{Pr}$ respectively, are optimal.*

4.5 Quotients with respect to Groundness

The interpretations **Sharing** and **Prop** are incomparable with respect to the notion of abstraction [5]. The intuition behind this result is the following. On the one hand, by means of disjunctions `Prop` represents also possible equivalence (and thus also groundness), whereas `Sharing` does not. On the other hand, `Sharing` represents variable independence that is not expressible in `Prop`.

As both interpretations compute groundness information, we are interested to measure their relative precision in the computation of groundness.

4.5.1 The domain GR

The simplest domain `GR` representing groundness is as follows. Given an element $[\Sigma, U] \in$ `Rsub`, its groundness information can be represented by the variables grounded by every substitution $\sigma \in \Sigma$.

Domain $\mathtt{GR} = [\wp(\mathbf{V}) \times FP(\mathbf{V})] \cup \{\top_{Gr}, \bot_{Gr}\}$. The set `GR` is partially ordered in a natural way. $\top_{Gr}$ is the top element, and $\bot_{Gr}$ the bottom one. $[B_1, U_1] \sqsubseteq_{Gr} [B_2, U_2]$ if $U_1 = U_2$ and $B_1 \supseteq B_2$. Obviously, `GR` is a complete lattice. The least upper bound of two elements $[B_1, U_1]$ and $[B_2, U_2]$ is defined as

$$[B_1, U_1] \sqcup_{Gr} [B_2, U_2] = \begin{cases} [B_1 \cap B_2, U_1] & \text{if } U_1 = U_2 \\ \top_{Gr} & \text{otherwise} \end{cases}$$

Concretization and Abstraction Functions The relation between `Rsub` and `GR` is expressed by two functions: the concretization function, denoted $\gamma_{Gr\,Rs}$, and the abstraction one, denoted $\alpha_{Rs\,Gr}$. On nontrivial elements they are defined by

$$\alpha_{Rs\,Gr}([\Sigma, U]) = [\textstyle\bigcap_{\sigma \in \Sigma} \{x \in supp(\sigma) \mid Var(\sigma x) = \emptyset\}, U]$$
$$\gamma_{Gr\,Rs}([B, U]) = \textstyle\bigsqcup_{Rs} \{[\Sigma, U] \mid \alpha_{Rs\,Gr}([\Sigma, U]) \sqsubseteq_{Gr} g\}$$

Lemma 4.7 *The following facts hold*
- `GR` *abstracts* `Prop` *coeherently with respect to* `Rsub`.
- $\alpha_{Pr\,Gr}([f, U]) = [\{x \in U \mid f \models x\}, U]$.
- `GR` *abstracts* `Sharing` *coeherently with respect to* `Rsub`.
- $\alpha_{Sh\,Gr}([A, U]) = [U - Var(A), U]$.

4.5.2 The Quotient of Prop with respect to GR

In the previous sections we have proved that the quotient of an interpretation with respect to a given domain is a domain abstracting the starting domain, provided the associated relation is additive.

The quotient of **Prop** with respect to `GR` is `Prop` itself. This is due to the fact that none of the formulas of `Prop` is irrelevant for the computation of groundness [7].

Lemma 4.8 *Let $[f_1, U], [f_2, U] \in$ `Prop`. If $f_1 \not\equiv f_2$, there exists a derived operator t on the alphabet $\{\sqcup_{Pr}, \pi_{Pr}, \mathbf{U}_{Pr}\}$ such that $\alpha_{Pr\,Gr}(t([f_1, U])) \neq \alpha_{Pr\,Gr}(\mathbf{U}_{Pr}(t([f_2, U])))$.*

Corollary 4.9 *r_{Pr} is the identity on* `Prop`*, and thus it is obviously additive.*

Theorem 4.10 $Q_{\mathtt{GR}}(\mathtt{Prop}) = \mathtt{Prop}$.

proof: Follows immediately from Corollary 4.9 and Theorem 3.4. □

4.5.3 The Quotient of Sharing with respect to GR

According to Definition 3.1, we consider the following relation for $S_1, S_2 \in$ `Sharing`, where t is any derived operator on $\{\mathbf{U}_{Sh}, \pi_{Sh}, \sqcup_{Sh}\}$:

$$(S_1, S_2) \in r_{Sh} \Leftrightarrow \alpha_{Sh\,Gr}(t(S_1)) = \alpha_{Sh\,Gr}(t(S_2))$$

Notation Given a (possibly empty) set of propositional formulas $\{f_1, \ldots, f_n\}$, we denote by $\wedge\{f_1, \ldots, f_n\}$ the formula $f_1 \wedge \ldots \wedge f_n$ if $n \geq 0$ and the constant T if $n = 0$.

The `Sharing` information about groundness can be isolated by means of propositional formulas. For $[A, U] \in$ `Sharing` define

$$C([A, V]) = \wedge\{\wedge W_1 \rightarrow \wedge W_2 \mid W_1, W_2 \subseteq U \text{ and } \forall N \in A : (W_2 \cap N) \neq \emptyset \Rightarrow (W_1 \cap N) \neq \emptyset\}$$

Example 4.11 *Consider $S = [\{\{x, y\}, \{x\}, \{x, z\}\}, \{x, y, z, u\}]$. The formula $C(S) = u \wedge (x \rightarrow (y \wedge z))$ outlines the fact that if $\gamma_{Sh\,Rs}(S) = [\Sigma, \{x, y, z, u\}]$, for every $\sigma \in \Sigma$ the term $\sigma(u)$ is a ground term and if $\sigma(x)$ is a ground term, both $\sigma(y)$ and $\sigma(z)$ are ground terms too.*

Lemma 4.12 *Let $S_1 = [A_1, U], S_2 = [A_2, U]$ be elements of* `Sharing`.

$$C(S_1) \equiv C(S_2) \Rightarrow \alpha_{Sh\,Gr}(S_1) = \alpha_{Sh\,Gr}(S_2)$$

The following lemma show that for all three operations of `Sharing` the C-value of the results depends only on the C-values of the operands.

Lemma 4.13 *Let* $S_1 = [A_1, U], S_2 = [A_2, U]$ *be elements of* Sharing.

i) $\mathcal{C}(S_1) \equiv \mathcal{C}(S_2) \Rightarrow \forall \delta \in Subst : \mathcal{C}(\mathbf{U}_{Sh}(S_1, \delta)) \equiv \mathcal{C}(\mathbf{U}_{Sh}(S_2, \delta))$

ii) $\mathcal{C}(S_1) \equiv \mathcal{C}(S_2) \Rightarrow \forall U \subseteq \mathbf{V} : \mathcal{C}(\pi_{Sh}(S_1, U)) \equiv \mathcal{C}(\pi_{Sh}(S_2, U))$

iii) $\mathcal{C}(S_1) \equiv \mathcal{C}(S_2) \Rightarrow \forall S = [A, U] \in \mathtt{Sharing} : \mathcal{C}(S_1 \sqcup_{Sh} S)) \equiv \mathcal{C}(S_2 \sqcup_{Sh} S))$

Theorem 4.14 *Let* $S_1, S_2 \in \mathtt{Sharing}$, *where* $S_1 = [A_1, U_1], S_2 = [A_2, U_2]$.

$$(S_1, S_2) \in \mathbf{r}_{Sh} \Leftrightarrow U_1 = U_2 \text{ and } \mathcal{C}(S_1) \equiv \mathcal{C}(S_2).$$

proof:

$\Rightarrow$ From the definition of $\mathbf{r}_{Sh}$ it follows immediately that $U_1 = U_2$: it is sufficient to consider the empty sequence of operations to obtain

$$([A_1, U_1], [A_2, U_2]) \in \mathbf{r}_{Sh} \Rightarrow \alpha_{Sh\mathcal{G}r}([A_1, U_1]) = \alpha_{Sh\mathcal{G}r}([A_2, U_2]) \Rightarrow U_1 = U_2.$$

It remains to show that $\mathcal{C}(S_1) \equiv \mathcal{C}(S_2)$. Assume the converse. This means that there exists a formula $\psi = \wedge W_1 \rightarrow \wedge W_2$ such that $\mathcal{C}(S_1) \models \psi$ and $\mathcal{C}(S_2) \not\models \psi$. Consider the substitution $\delta = \{x/a \mid x \in W_1\}$ and let

$$\alpha_{Sh\mathcal{G}r}(\mathbf{U}_{Sh}(S_1, \delta)) = [B_1, U] \quad \text{and} \quad \alpha_{Sh\mathcal{G}r}(\mathbf{U}_{Sh}(S_2, \delta)) = [B_2, U]$$

As $W_2 \subseteq B_1$ whereas $W_2 \not\subseteq B_2$, $[B_1, U] \neq [B_2, U]$ which contradicts the hypothesis $(S_1, S_2) \in \mathbf{r}_{Sh}$.

$\Leftarrow$ The claim is: for any derived operator t on $\{\sqcup_{Sh}, \pi_{Sh}, \mathbf{U}_{Sh}\}$,

$$\mathcal{C}(S_1) \equiv \mathcal{C}(S_2) \wedge U_1 = U_2 \Rightarrow \alpha_{Sh\mathcal{G}r}(t(S_1)) = \alpha_{Sh\mathcal{G}r}(t(S_2)).$$

. By structural induction on the derived operator t, it is straightforward to show that $\mathcal{C}(t(S_1)) \equiv \mathcal{C}(t(S_2))$, using lemma 4.13. For concluding the proof it is sufficient to use Lemma 4.12. □

Corollary 4.15 $\mathcal{Q}_{\mathtt{GR}}(\mathtt{Sharing}) = \{[\mathcal{C}([A, U]), U] \mid [A, U] \in \mathtt{Sharing}\} \cup \{\top_{Sh}, \bot_{Sh}\}$.

Corollary 4.16 $\mathbf{r}_{Sh}$ *is additive.*

proof: By Lemma 4.13 and Theorem 4.14. □

4.5.4 Comparison of the Quotients

Theorem 4.17 $\mathcal{Q}_{\mathtt{GR}}(\mathtt{Sharing})$ *properly abstracts* Prop, *coherently with respect to* Rsub.

proof: By Corollary 4.15, we know that the elements of $\mathcal{Q}_{\mathtt{GR}}(\mathtt{Sharing})$ are pairs $[f, U]$ where f is a propositional formula of type $f = \wedge\{\wedge S_1 \rightarrow \wedge S_2 \mid S_1, S_2 \subseteq U\}$. Dart [9] observed that the union of these formulas is a (proper) sublattice of Prop (called *Def* in [9, 12]). The intuition behind that proof is the following: formulas like $(x \vee y)$ are not expressable by *Def*. Therefore $\mathcal{Q}_{\mathtt{GR}}(\mathtt{Sharing})$ is a (proper) abstraction of Prop: it is sufficient to choose as concretization function the identity, and as abstraction function the usual adjoint. □

By combining Theorem 4.17 with Theorem 3.5 and Theorem 3.6 we get the announced result.

Theorem 4.18 **Prop** *is strictly more precise than* **Sharing** *with respect to the domain* GR *representing groundness.*

References

[1] Codish M., Mulkers A., Bruynooghe M., Garcia de la Banda M., Hermenegildo M., "Improving abstract interpretations by combining domains". In *Proc. ACM PEPM*, Copenhagen, 1993.

[2] Cousot, P. and Cousot, R., "Abstract interpretation: a unified framework for static analysis of programs by construction of approximation of fixpoints." *Proc. Fourth ACM POPL*, 1977.

[3] Cousot, P. and Cousot, R., "Abstract Interpretation and applications to logic programs". *Journal of Logic Programming*, 13, 1992.

[4] Cortesi, A., Filè, G. and Winsborough W., "Prop revisited: Propositional formula as abstract domain for groundness analysis." *Proc. Sixth IEEE LICS*, Amsterdam 1991.

[5] Cortesi, A., Filè, G. and Winsborough W., "Comparison of Abstract Interpretations." *Proc. 19th ICALP*, Wien 1992, LNCS 623.

[6] Cortesi, A. "Domini Astratti per l'Analisi Statica di Programmi Logici", PhD thesis, Universitá di Padova. 1992.

[7] Cortesi, A., Filè, G. and Winsborough W., "Propositional logic as abstract interpretation for groundness analysis." Tech. Rep. Universitá di Padova (forthcoming), 1994.

[8] Cortesi, A., Van Hentenryck P. and Le Charlier B., "Combinations of Abstract Domains for Logic Programming". *Proc. 21th ACM POPL*, Portland 1994.

[9] Dart, P. "Dependency Analysis and query interfaces for deductive databases." PhD Dissertation, Tech. rep. 88/35, The University of Melbourne, 1988.

[10] Jacobs, D. and Langen, A., "Accurate and efficient approximation of variable aliasing in logic programs." *Proc. NACLP*, Cleveland, 1989.

[11] Marriott, K. and Søndergaard, H., "Notes for a tutorial on abstract interpretation of logic programs." NACLP. Cleveland, 1989.

[12] Marriott, K. and Søndergaard, H., "Precise and Efficient Groundness Analysis for logic programs." Tech. Rep. 93/7, University of Melbourne, 1993.

Loop Checking for Reduced SLD-Derivations*

Filomena Ferrucci** Giuliano Pacini* Maria I. Sessa**

* Dipartimento di Matematica Applicata ed Informatica
Università di VENEZIA - I
Via Torino, 155 - 30170 Mestre (VE) - I
pacini@di.unipi.it

** Dipartimento di Informatica ed Applicazioni
84081 Baronissi SALERNO - I
{filfer, mis}@udsab.dia.unisa.it

Abstract

In the paper, the completeness of the equality loop check is addressed in the context of Reduced SLD-resolution [4] that performs a systematic elimination of redundant atoms. The notion of *section*, capturing the idea of set of atoms chained by common variables, is introduced. On the basis of such a notion a reduction rule, called RR*, is introduced and the *e.b.s.v.* property (*extended bounded spreading variable* property) in the resolvents of an RSLD-derivation is identified. The e.b.s.v. property states the existence of a limitation on the number of spreading variables in any section in the resolvents. Then, the completeness of equality loop check via RR* is proved for classes of programs for which the e.b.s.v. property holds. The equality check is complete for the classes of function-free logic programs for which more complex loop checks are proved complete in [2]. Moreover, the completeness of equality check via RR* is proved for a new class of programs.

1 Introduction

The problem of detecting any infinite branch in an SLD-tree is obviously undecidable, as the logic programming has the full power of recursion theory. However, for some classes of logic programs, some techniques can be applied that allow to prune every infinite derivation [2, 4, 6, 7]. These mechanisms are called loop checks, as they are based on excluding some kinds of repetitions in the derivations. Some basic properties are stated for

* This research has been supported by grants MURST

a loop check. The completeness property of a loop check concerns with the capability of pruning every infinite derivation. In contrast, the soundness property of a loop check concerns with the preservation of the results (successful, computed answer substitution).

In [2] several possible variants of loop checks are defined and analyzed with their soundness and completeness properties. In particular, some classes of logic programs are given for which complete loop checks can be found. Those classes are characterized by suitable constraints over the structure of the rules, on the basis of which the completeness of loop checks is shown. In [4] a somewhat different line is followed. A limitation on the number of a particular kind of variables in the resolvents is discussed, without any direct imposition about the program structure. As a matter of fact, a kind of operational characterization is given which is able to ensure the completeness of equality loop check. Thus, the proof of the suitability of structural restrictions on the program rules, ensuring the completeness of the loop check, is reconduced to verify that the imposed program structure implies the above mentioned limitation of variable number in the resolvents.

The specific nature of the variables to be limited is suggested by the observation that the non-existence of a complete simple loop check, even for function-free programs, is due to the presence in the resolvents of potentially unlimited sequences of atoms which are chained by common variables. It is intuitive that the growth in size of such chains is due to the availability of an unlimited number of variables with multiple occurrences in the resolvents. In particular, this kind of chains is formed through variables with multiple occurrences which appear at least in one atom with other multiple occurrence variables (called *spreading* variables in the sequel of the paper). The existence of a limitation on the number of spreading variables in the resolvents is expressed by the *bounded spreading variables* property (*b.s.v.* property for short).

In [4] the study of the termination problem is addressed in the context of Reduced SLD-resolution, (called RSLD-resolution) that performs a systematic elimination of redundant atoms. Such form of resolution is a sound and complete variant of SLD-resolution. It applies a redundant atom elimination technique (*reduction* in the sequel), in order to limit the size of the resolvents. Indeed, it is well known that the increase in size of resolvents is a factor that prevents SLD-resolution being decision procedure.

As a result, the loop check based on equality is complete for a quite large class of function-free programs, namely all the function-free programs with the b.s.v. property. Such a class includes all the classes for which the more complex loop checks based on subsumption are proved complete in [2]. This is rather unexpected, since it is well known that, in the context of SLD-resolution, the equality loop checks are less strong than subsumption loop checks. The explanation of this apparent contradiction resides in fact that the equality check benefits of the systematic elimination of redundant atoms performed by the RSLD-resolution applying a well defined reduction rule $\bar{R}\bar{R}$.

In this paper, results stated in [4] are extended by introducing the notion of *section*. This definition captures the idea of set of atoms chained by common variables, so that any resolvent can be partitioned in a set of sections. Then, the more general reduction rule RR* can be defined. The reduction rule RR* allows a better limitation of the growth in the size of the resolvents since it performs elimination of sections of atoms, too.

As a consequence, the completeness of the equality is stated for a larger class of programs that observe the *extended bounded spreading variables* property (*e.b.s.v.* in the sequel). More precisely, we say that a derivation D has the e.b.s.v. property if in any resolvent R of D, every section of R has a number of spreading variables bounded by a finite value. Since the e.b.s.v. property implies the b.s.v. one, the equality check via RR* is obviously complete for all the classes for which it is complete when the reduction rule

$\bar{R}\bar{R}$ is applied. Moreover, it is proved the completeness of equality check via RR* for a new class of programs (*envi* logic programs) for which the e.b.s.v. property holds. This is a significant extension of the previous results. Indeed, the b.s.v. property does not hold for the new class and it is proved that EVRR via $\bar{R}\bar{R}$ is not complete for this class.

2 Reduced SLD-Derivations

In this section, we recall the definition of Reduced SLD-resolution (RSLD-resolution, for short) as introduced in [4]. Such form of resolution is a variant of SLD-resolution, since it applies a redundant atom elimination technique (*reduction* in the sequel), in order to limit the size of the resolvents. Indeed, it is well known that the increase in size of resolvents is a factor that prevents SLD-resolution being decision procedure. It is proved in [4] that such reduction technique is sound and complete. In order to define the RSLD-resolution, the definition of reduced goal is given. In the following, var(F) denotes the set of variables of a formula F and the equality (subsumption) relation between goals regarded as lists is denoted by the symbol $=_L$ ($\subseteq_L$).

Definition 2.1 Let $\mathbf{X}$ be a set of variables, σ a substitution and G a goal. A goal G' is a *reduced goal* of G by σ up to $\mathbf{X}$, denoted by $G \rhd_\sigma G'$, if the following conditions hold:

i) $G' \subseteq_L G$;
ii) $A\sigma \in G'$, $\forall A \in G$;
iii) $x\sigma = x$, $\forall x \in (\mathbf{X} \cup var(G'))$. ❑

In agreement with the above definition, a subset $\bar{G}$ of atoms in G can be eliminated if a substitution σ exists such that for any atom B in $\bar{G}$, $B\sigma \in G' = (G - \bar{G})$, provided that σ does not affect either the variables in G' or those in $\mathbf{X}$.

Example 2.1 Given the goal $G = \leftarrow \mathbf{p(Z,V)}$, q(b), p(W,V), **p(Z,V)**, q(V) and the set $\mathbf{X} = \{W\}$, the subset $\bar{G} = \{p(Z, V), p(Z, V)\}$ can be eliminated by the substitution $\sigma = \{Z/W\}$. Indeed $\bar{G}\sigma = \{p(W, V), p(W, V)\}$ and p(W, V) is an atom of $G' = \leftarrow q(b)$, p(W,V), q(V). It is worth to note that neither p(W, V) nor q(V) can be eliminated in G. ❑

The possibility of performing reductions in the resolvents of an SLD-derivation is really a modification of the resolution process. Then, a modified definition of the SLD-resolution has been formalized, namely the *Reduced SLD-resolution*. For any substitution θ and set of variables $\mathbf{X}$, $\mathbf{X}\theta$ is the set of variables obtained by applying θ to each element of $\mathbf{X}$.

Definition 2.2 (Reduced SLD-derivation) Given a logic program P and a negative clause G_0, an *RSLD-derivation* of $P \cup \{G_0\}$ consists of a sequence of pairs $(G_0, N_0), (G_1, N_1), \ldots$ of negative clauses together with a sequence $C_0, C_1, \ldots$ of variants of clauses from P and two sequences $\theta_0, \theta_1, \ldots$ and $\alpha_0, \alpha_1, \ldots$ of substitutions such that for all $i = 0, 1, \ldots$

i) N_i is a reduced goal of G_i by α_i up to $var(G_0\theta_0 \ldots \theta_{i-1})$;
ii) G_{i+1} is a resolvent of N_i and C_i with the mgu θ_i;
iii) C_i has no variable in common with $G_0, C_0, \ldots, C_{i-1}$. ❑

As usual the goals $G_0, G_1, \ldots$ are called *resolvents*. The goals $N_0, N_1, \ldots$ are called *reduced resolvents* of the derivation. When one of the resolvents G_n is empty then $N_n = G_n = \square$ and it is the last negative clause of the derivation. Such a derivation is then called an *RSLD-refutation* (or a *successful RSLD-derivation*) and the composition of mgu's $\theta_0\theta_1 \ldots \theta_{n-1}$ restricted to the variables of G_0 is the *computed answer substitution*. Note that the substitutions 'α's do not contribute to the computed answer. By *resultant (of level i)* we mean the formula $G_i^{\sim} \rightarrow G_0^{\sim}\theta_0\theta_1 \ldots \theta_{i-1}$ and by *reduced resultant (of level i)* we mean the formula $N_i^{\sim} \rightarrow G_0^{\sim}\theta_0\theta_1 \ldots \theta_{i-1}$. According to the definition of RSLD-derivation, at any step, a choice that gives the reduction of the produced resolvent is made. More formally, let us state the following definition

Definition 2.3 (Reduction Rule) A *reduction rule RR* is a function that, given a goal G and a set of variables $\mathbf{X}$, yields a goal N and a substitution α such that N is a reduced goal of G by α up to $\mathbf{X}$ (it is denoted by $RR(G, \mathbf{X}) = (N, \alpha)$). ❑

Given a reduction rule *RR* and a selection rule *SR* we say that an RSLD-derivation of $P \cup \{G_0\}$ is *via RR* and *SR* if all choices of the reduced goals and of the selected atoms in the derivation are performed according to *RR* and *SR*, respectively. Then, given a logic program P and a goal G_0, the *search space* is formed by the totality of the RSLD-derivations which can be constructed. This search space can be organized into RSLD-trees according to the selection rule and reduction rule used. The *RSLD-tree* for $P \cup \{G_0\}$ *via RR* and *SR* groups all RSLD-derivations of $P \cup \{G_0\}$ *via RR* and *SR*.

Example 2.2 Consider the logic program P formed by the clauses C_1 and C_2:
C_1 : p(X) ←. C_2: p(X) ← p(X), p(X).
and $G = \leftarrow p(Y), p(Y)$ a goal. An RSLD-tree of $P \cup \{G\}$ is shown in Fig. 2.1.

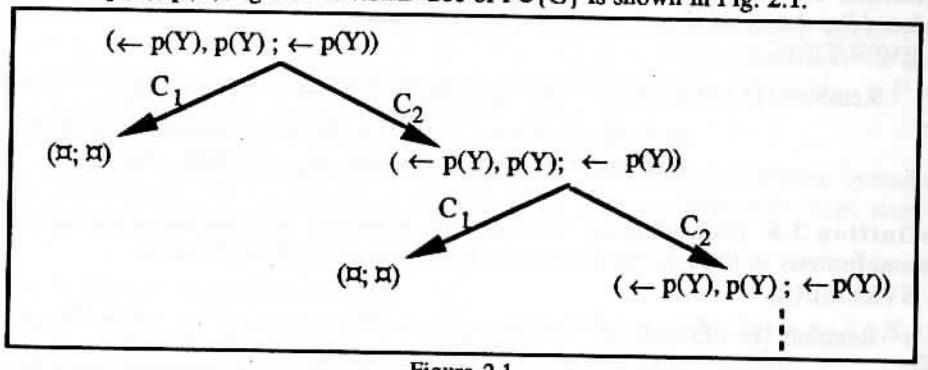

Figure 2.1

The RSLD-resolution is sound and strong complete. More formally:

Theorem 2.1 *Let P be a logic program and $G_0 = \leftarrow A_1, \ldots, A_k$ a goal.*
i) (Soundness) *If there exists an RSLD-refutation of $P \cup \{G_0\}$ with the sequence of mgu's $\theta_0, \theta_1, \ldots, \theta_n$, then $\forall((A_1 \wedge \ldots \wedge A_k)\theta_0\theta_1 \ldots \theta_n)$ is a semantic consequence of P.*
ii) (Strong Completeness) *Let ρ be a correct answer substitution of $P \cup \{G_0\}$. Every RSLD-tree of $P \cup \{G_0\}$ contains a successful branch with computed answer substitution σ such that $\sigma \leq \rho$ up to renaming.* ❑

3 Loop Checks for RSLD-Derivations

In this section some definitions and results about loop checks in the context of RSLD-resolution are recalled, as they are presented in [4]. Generally speaking, a loop check is defined as a set of RSLD-derivations that are pruned exactly at their last node. More formally, the following definitions are given:

Definition 3.1 Let L be a set of RSLD-derivations. *RemSub(L)* = {D ∈ L | L does not contain a proper subderivation of D}. L is *subderivation free* if L = RemSub(L). ❑

Definition 3.2 A *simple loop check* is a computable set L of finite RSLD-derivations such that L is subderivation free and closed under variants.
A *loop check* is a computable function L from programs to sets of RSLD-derivations such that for every program P, L(P) is a simple loop check. ❑

We say that a derivation D of P ∪ {G} is *pruned by* a loop check *L* if L(P) contains a subderivation D' of D.

The basic properties of a loop check are soundness and completeness. The property of soundness is concerned with the preservation of the results (successful, computed answer substitution). In contrast, the completeness concerns with the pruning of every infinite derivation. In [2], two simple loop checks for SLD-derivations and some their variants are introduced with their respective soundness and completeness properties. These loop checks are based, respectively, on equality/subsumption of goal/resultant.

In [4] these definitions of loop checks are reformulated for RSLD-resolution.

Definition 3.3 (Equality Checks for Reduced Resultants) *Equals Variant/Instance of Reduced Resultant*$_L$ check is the set of RSLD-derivations
EVRR/EIRR$_L$ =

$$\text{RemSub}(\{D \mid D = (G_0 \rhd^{\alpha_0} N_0 \Rightarrow_{C_0,\theta_0} G_1 \rhd^{\alpha_1} N_1 \Rightarrow \ldots \Rightarrow_{C_{k-1},\theta_{k-1}} G_k \rhd^{\alpha_k} N_k)$$

such that for some i, $0 \le i < k$, there is a renaming/substitution τ such that $N_k =_L N_i\tau$ and $G_0\theta_0\theta_1\ldots\theta_{k-1} = G_0\theta_0\theta_1\ldots\theta_{i-1}\tau\}$). ❑

Definition 3.4 (Subsumption Checks for Reduced Resultants) *Subsumption Variant/Instance of Reduced Resultant*$_L$ check is the set of RSLD-derivations
SVRR/SIRR$_L$ =

$$\text{RemSub}(\{D \mid D = (G_0 \rhd^{\alpha_0} N_0 \Rightarrow_{C_0,\theta_0} G_1 \rhd^{\alpha_1} N_1 \Rightarrow \ldots \Rightarrow_{C_{k-1},\theta_{k-1}} G_k \rhd^{\alpha_k} N_k)$$

such that for some i, $0 \le i < k$, there is a renaming/substitution τ such that $N_i\tau \subseteq_L N_k$ and $G_0\theta_0\theta_1\ldots\theta_{k-1} = G_0\theta_0\theta_1\ldots\theta_{i-1}\tau\}$). ❑

Example 3.1 Fig. 3.1 shows the pruned tree obtained by applying the EVRR$_L$ loop check to the RSLD-tree of Fig. 2.1.

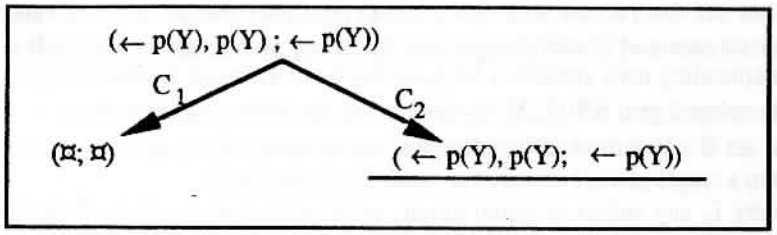

Figure 3.1

The loop checks based on equality have lower computational complexity than subsumption checks. In [4] the problem of finding general conditions on the resolvents ensuring completeness for EVRR$_L$ loop check is addressed. Generally speaking, the non-existence of a complete simple loop check arises from the presence in the resolvents of potentially unlimited sequences of atoms which are chained by common variables. It is intuitive that the growth in size of such chains is due to the availability of an unlimited number of variables with multiple occurrences in the resolvents. In particular, this kind of chains is formed through variables with multiple occurrences which appear at least in one atom with other multiple occurrence variables (called *spreading* variables).

This observation suggested the feasibility of characterizing classes of programs by imposing limitations on the number of spreading variables in the resolvents (b.s.v. property), in order to guarantee the existence of complete simple loop checks. In [4] formal definitions are given for the kind of variables to be limited and a particular reduction rule is introduced, namely the reduction rule $\bar{R}\bar{R}$. Such a property ensures the limitation in size of the reduced resolvents and then the completeness of EVRR$_L$ loop check for function-free programs, when RSLD-resolution procedure is performed via the reduction rule $\bar{R}\bar{R}$.

Definition 3.5 Given a formula F, a variable x has the *single variable occurrence* property (is *svo* in short) if it occurs only once in F, otherwise it has the *multiple variable occurrence* property (is *mvo* in short). A variable x is called *spreading* if x is mvo in F and x occurs at least once with an mvo variable in an atom of F. ❑

It is immediate that a spreading variable x occurs at least once with another spreading variable in an atom of F, in particular a variable which occurs at least twice in an atom of F is a spreading variable.

Example 3.2 Given the formula:
$p(X_1, X_1, X_8) \vee p_2(X_2, X_3) \wedge p_3(X_2, X_4, X_9) \vee p_4(X_5, X_6) \wedge p_5(X_5, X_7) \wedge p_6(X_6, X_8)$
X_1, X_5, X_6, X_8 are spreading variables, and X_2 is non spreading mvo variable. ❑

Definition 3.6 Let P be a logic program and G_0 a goal in L_P. If an RSLD-derivation (or SLD-derivation) D of P∪{G_0} is such that every resolvent of D has a number of spreading variables bounded by a finite value, then we say that D has the *bounded spreading variable property* (is *b.s.v.* in short). ❑

Now, we present the definition of the reduction rule $\bar{R}\bar{R}$:

Definition 3.7 (*Reduction Rule* $\bar{R}\bar{R}$) Let G be a goal and $\mathbf{X}$ a set of variables. Let A be the set of atoms of G containing a non spreading mvo variable, and let B = G - A. For any non spreading mvo variable x let I_x be the set of atoms in A containing an occurrence of x. The reduced goal $\bar{R}\bar{R}(G, \mathbf{X})$ is obtained by the following three steps:

i) In the set B any subset of equal atoms, up to renaming σ_1 of svo variables $\notin \mathbf{X}$, is reduced to a single atom, i.e., only one atom is retained in B'.

ii) In every I_x any subset of equal atoms, up to renaming σ_2 of svo variables $\notin \mathbf{X}$, is reduced to a single atom, (denote by I'_x the reduced set of I_x).

iii) In the family of sets obtained in ii), any set I'_x is eliminated if there exists a rename σ_3 of variables $\notin \mathbf{X}$ such that $I'_x\sigma_3$ is included in another set I'_y. ❑

Example 3.3 Given the set $\mathbf{X} = \{X_2, X_3\}$ and the goal:

$G = \leftarrow p(X_1, X_6, X_8), q(X_4, X_3), p(X_2, X_6, X_8), r(X_5, X_4), q(X_4, X_7), r(X_8, X_6), s(X_4), r(X_9, X_{10}), q(X_{10}, X_{11}), r(X_{12}, X_{13})$

Since X_4 and X_{10} are non spreading mvo variables, then

$A = \{q(X_4, X_3), r(X_5, X_4), q(X_4, X_7), s(X_4)\} \cup \{r(X_9, X_{10}), q(X_{10}, X_{11})\} = I_{X_4} \cup I_{X_{10}}$ and

$B = \{p(X_1, X_6, X_8), p(X_2, X_6, X_8), r(X_8, X_6), r(X_{12}, X_{13})\}$.

By i) since the atom $p(X_1, X_6, X_8)\sigma_1 = p(X_2, X_6, X_8)$ and $r(X_{12}, X_{13})\sigma_2 = r(X_8, X_6)$ where $\sigma_1 = \{X_1/ X_2\}$ and $\sigma_2 = \{X_{12}/ X_8, X_{13}/ X_6\}$ are renamings of svo variables $\notin \mathbf{X}$, we obtain

$B' = \{p(X_2, X_6, X_8), r(X_8, X_6)\}$.

By ii), since the atom $q(X_4, X_7)\sigma_3 = q(X_4, X_3)$ where $\sigma_3 = \{X_7/ X_3\}$ is a renaming of svo variables $\notin \mathbf{X}$, we obtain:

$I'_{X_4} = \{q(X_4, X_3), r(X_5, X_4), s(X_4)\}$ and

$I'_{X_{10}} = \{r(X_9, X_{10}), q(X_{10}, X_{11})\}$.

By iii), since the set $I'_{X_{10}}\sigma_4 \subset I'_{X_4}$ where $\sigma_4 = \{X_{10}/ X_4, X_{11}/ X_3, X_9/ X_5\}$ is a renaming of svo variables $\notin \mathbf{X}$, we obtain:

$A' = \{q(X_4, X_3), r(X_5, X_4), s(X_4)\}$.

Then $G' = \leftarrow q(X_4, X_3), p(X_2, X_6, X_8), r(X_5, X_4), r(X_8, X_6), s(X_4)$, is a reduced goal of G by $\sigma = \{X_1/ X_2, X_7/ X_3, X_{10}/ X_4, X_{11}/ X_3, X_9/ X_5, X_{12}/ X_8, X_{13}/ X_6\}$ up to $\mathbf{X} = \{X_2, X_3\}$. ❑

The reduction rule $\bar{R}\bar{R}$ is a particular reduction rule which ensures the completeness of $EVRR_L$ loop check for function-free programs, when, in any resolvents, a suitable limitation on the number of variables which are shared by different atoms is imposed. The proof of this result is based on the following Lemma:

Lemma 3.1 *Let P be a function-free program, G_0 a goal in L_P. If an RSLD-derivation D of $P \cup \{G_0\}$ via $\bar{R}\bar{R}$ is b.s.v., then the length of every reduced resolvent N_i in D is bounded.* ❑

Theorem 3.1 (Completeness of $EVRR_L$) *Let P be a function-free program, G_0 a goal in L_P. Any infinite b.s.v. RSLD-derivation D of $P \cup \{G_0\}$ via $\bar{R}\bar{R}$ is pruned by $EVRR_L$.* ❑

Now, we give the definitions of the classes of logic programs for which the completeness properties of the equality/subsumption loop checks has been proved in [2, 4].

Definition 3.8.

i) (Restricted Program) A program P is *restricted* if for every clause $H \leftarrow A_1, ..., A_n$ in P, the definitions of the predicates in $A_1,...,A_{n-1}$ do not depend on the predicate of H.

ii) (nvi Program) A program P is *non-variable introducing (nvi)* if for every clause $H \leftarrow A_1, ..., A_n$ in P, every variable that occurs in $A_1,...,A_n$ occurs also in H.

iii) (svo Program) A program P has the *single variable occurrence* property (is svo) if for every clause $H \leftarrow A_1, ..., A_n$ in P, no variable occurs more than once in $A_1,...,A_n$.

iv) (esvo Program) A program P has the *extended single variable occurence* property (is esvo) if in the body of every clause in P, every mvo variable occurs only with new svo variables. ❑

The following Table I summarizes the influence of reduction for the completeness of loop checks. As shown in the table, equality loop check becomes complete for several classes of programs, when it is used in combination with RSLD-resolution. For the same classes more complex subsumption checks are needed if usual SLD-resolution is exploited. In other words, the necessity of subsumption loop checks for several classes of logic programs appears to be originated by the presence of redundant atoms in the resolvents, which the reduction technique is able to eliminate.

Table I

SLD-resolution		
Loop checks based on Equality	restricted*	function-free
Loop checks based on Subsumption	restricted*	function-free
	svo	"
	nvi	"
RSLD-resolution via $\bar{R}\bar{R}$		
Loop checks based on equality	restricted*	function-free
	svo	"
	nvi	"
	esvo	"

* via leftmost selection rule

4 Sections and Reduction Rule RR*

In [3] an equivalence relation is introduced on a set of atoms. This definition captures the notion that two variables in a goal are related, so that they might be unified in an attempt

to refute the goal. Then, it is proved that when two variables occur unrelated in a certain goal, they cannot be related in any goal later in the derivation. Let us recall this result:

Definition 4.1 [3] Let F be a set of atoms. The relation ~ on variables is defined as:
x ~ y if there is an atom A in F such that x, y ∈ var(A).
Obviously, ~ is a symmetrical relation. Then, the relation ~~ is defined as the transitive and reflexive closure of ~ . Thus ~~ is an equivalence relation. An equivalence class of ~~ is called *chain*. For x ∈ var(F), the chain of x is denoted by $C_F[x]$, or C[x] whenever F is clear from the context. ❑

Lemma 4.1 [3] *Let* $D = G_0 \Rightarrow_{C_1,\theta_1} G_1 \Rightarrow \ldots \Rightarrow_{C_{i-1},\theta_{i-1}} G_i \Rightarrow \ldots$ *be an SLD-derivation and let* $0 < i \ (\leq |D|)$. *If* $y \in C_{G_i}[x]$ *and* $x, y \in var(G_{i-1})$, *then* $y \in C_{G_{i-1}}[x]$ ❑

It follows by the definition of ~~ that, for any atom A in a set F, all the elements of var(A) belong to the same chain. Moreover, it is worth to note that if two atoms A and B in a set F share a variables x, then var(A) and var(B) are both included in $C_F(x)$. The previous remarks suggest to group a subset of atoms chained by common variables in a suitable class of equivalence. Thus, let us introduce the following equivalence relation that is induced by ~~ and is defined on a set of atoms:

Definition 4.2 Let F be a set of atoms. We define the relation * on atoms of F as:
A*B if var(A) and var(B) are included in the same chain of ~~ on F.
An equivalence class of * is called *section*. For A ∈ F, the section of A is denoted by $S_F(A)$, or S(A) whenever F is clear from the context. ❑

It is easy to verify that * is an equivalence relation. The following result is a straight consequence of the definition of *:

Lemma 4.2 *Let G be a goal and* $S_1, \ldots, S_n$ *be the sections of G. Then,* $var(S_i) \cap var(S_j) = \emptyset$ *for* $i \neq j$ *and* $i, j \in \{1, \ldots, n\}$. ❑

The definition of section suggests an interesting view of a goal G, namely a goal may be seen as a quotient set of chain. Clearly all the atoms of G containing an mvo variable x belong to the same section and, all the atoms chained by spreading variables are contained in the same section. As a consequence, also the reduction rule $\bar{R}\bar{R}$ can be interpreted in this new light and a natural and powerful extension can be defined. Indeed, it is easy to verify that a set I_x in the definition of $\bar{R}\bar{R}$ is actually the section $S_G(A)$, where A is an atom in G containing the mvo non spreading variable x. Then, it can be introduced a new reduction rule RR* that acts on sections as well as $\bar{R}\bar{R}$ does on the sets I_x's. It means that the same kind of simplification defined between sets I_x and I_y is extended to the sections. More precisely, let us give the following definition:

Definition 4.3 (*Reduction Rule RR**) Let G be a goal and $\mathbf{X}$ a set of variables. Suppose that $S_1, \ldots, S_n$ are the sections of G. The reduced goal G' of G up to $\mathbf{X}$ via RR* (namely RR*(G, $\mathbf{X}$) = (G', σ)) is obtained by the following two steps:
i) In every S_i (i = 1, ..., n) any subset of equal atoms and of equal atoms up to a renaming γ_i of svo variables $\notin \mathbf{X}$, is reduced to a single atom. Let S'_i denote the reduced set of S_i.
ii) In the family of sets obtained in i), any set S'_i is eliminated if there exists a renaming τ_i of variables $\notin \mathbf{X}$ such that $S'_i\tau_i$ is included in another set S'_j with $i \neq j$ and $i, j \in \{1, \ldots, n\}$. ❑

Lemma 4.3 *Let G be a goal and* $\mathbf{X}$ *a set of variables. Then the goal G' such that RR*(G,* $\mathbf{X}$*) = (G', σ), is a reduced goal by σ up to* $\mathbf{X}$, *where σ is a renaming which is obtained by composing the* γ_i*'s and* τ_i*'s renamings.*
Proof. We show that the eliminations of atoms in i) and ii) of Definition 4.3 are actually reductions and then, RR* is a reduction. Indeed, the empty substitution ε and the γ_i's renamings in i) satisfy the conditions of Definition 2.1, since they do not affect either the variables of $\mathbf{X}$ (by definition) or the variables of the reduced goal because they apply only on svo variables. In order to convince that τ_i's renamings satisfy conditions of Definition 2.1, it is worth to note that, by Lemma 4.2, $var(S_i) \cap var(S_j) = \emptyset$ for $i \neq j$ and $i, j \in \{1, \ldots, n\}$. Thus, for any S'_i, τ_i's renamings obey to Definition 2.1 because they do not affect either the variables of $\mathbf{X}$ (by definition) or the variables of the reduced goal. ❑

Example 4.1 Let us apply the reduction rule RR* to the goal G and the set of variables $\mathbf{X} = \{X_2, X_3\}$ previously considered in the Example 3.3:
$G = \leftarrow p(X_1, X_6, X_8), q(X_4, X_3), p(X_2, X_6, X_8), r(X_5, X_4), q(X_4, X_7), r(X_8, X_6), s(X_4), r(X_9, X_{10}), q(X_{10}, X_{11}), r(X_{12}, X_{13})$
The sections of * in G are (where variables in $\mathbf{X}$ are stressed in bold characters):
$S_1 = p(X_1, X_6, X_8), p(\mathbf{X_2}, X_6, X_8), r(X_8, X_6)$
$S_2 = q(X_4, \mathbf{X_3}), r(X_5, X_4), q(X_4, X_7), s(X_4)$
$S_3 = r(X_9, X_{10}), q(X_{10}, X_{11})$
$S_4 = r(X_{12}, X_{13})$
By step i) of RR*, the following reduced sections are obtained:
$S'_1 = p(\mathbf{X_2}, X_6, X_8), r(X_8, X_6)$ by $\gamma_1 = \{X_1/ X_2\}$
$S'_2 = q(X_4, \mathbf{X_3}), r(X_5, X_4), s(X_4)$ by $\gamma_2 = \{X_7/ X_3\}$
$S'_3 = r(X_9, X_{10}), q(X_{10}, X_{11})$
$S'_4 = r(X_{12}, X_{13})$
By step ii) of RR* the final set of reduced sections in G' = RR*(G, $\mathbf{X}$) are:
$S'_1 = p(\mathbf{X_2}, X_6, X_8), r(X_8, X_6)$
$S'_2 = q(X_4, \mathbf{X_3}), r(X_5, X_4), s(X_4)$
since by $\tau_1 = \{X_{12}/ X_8, X_{13}/ X_6\}$ it is $S'_4\tau_1 \subseteq S'_1$ and by $\tau_2 = \{X_{10}/ X_4, X_{11}/ X_3, X_9/ X_5\}$ it is $S'_3\tau_2 \subseteq S'_2$. ❑

In this example, it can be verified that the reduced goal RR*(G, $\mathbf{X}$) is the same as $\bar{R}\bar{R}$ (G, $\mathbf{X}$). However, it is worth to note that the reduction rule RR* is more powerful

than $\bar{R}\bar{R}$, i.e., in general, more simplifications of atoms are possible via RR*. Indeed, by the definitions, if an atom can be eliminated via $\bar{R}\bar{R}$, then it can be eliminated via RR*, too; the vice versa is not always true as the following example shows:

Example 4.2 Let us consider the set of variables $\mathbf{X} = \{X_3, Y_3\}$ and the goal:
$G = \leftarrow A(X_1, Y_1), B(X_1, Y_1), C(X_1), A(X_2, Y_2), C(X_2), B(X_2, Y_2), A(X_3, Y_3), B(X_3, Y_3), C(X_3)$
Since X_1, Y_1, X_2, X_3, Y_3 are spreading variables, it can be verified that $\bar{R}\bar{R}(G, \mathbf{X}) = G$ whereas $RR^*(G, \mathbf{X}) = \leftarrow A(X_3, Y_3), B(X_3, Y_3), C(X_3)$. ❑

As previously recalled, in [4] it is shown that, in order to guarantee the existence of complete simple loop checks, the b.s.v. property characterizes a quite general class of programs without any direct imposition on the structures of the rules. Then, the completeness of the $EVRR_L$ loop check is proved for programs having SLD-derivations with the b.s.v. property. It is worth to notice that, if no reduction is performed, the limitation on the number of spreading variables in the resolvents does not imply any bound on the number of atoms, since we may have an unlimited number of occurrences of any single spreading variables. When the reductions are performed by applying the reduction rule $\bar{R}\bar{R}$ in b.s.v. derivations, the size of reduced resolvents is bounded [4].

Now, the view of a goal as a quotient set of the equivalence relation and the definition of the RR* reduction rule allow a generalization of this result. Indeed, a more general definition of the b.s.v. property can be introduced. In this case, the limitation on the number of spreading variables is requested only for the set of atoms belonging to the same section (class of equivalence). In other words, the resolvents may not be b.s.v. but any section in the resolvents must be b.s.v. As a consequence, if no reduction is performed, the number of *different* spreading variables in a resolvent is no more bounded, since the number of sections may be unlimited. More formally, let us give the following definition:

Definition 4.4 Let P be a logic program, G_0 a goal in L_P and D an SLD-derivation (or RSLD-derivation) of $P \cup \{G_0\}$. If in any resolvent R of D, every section of R has a number of spreading variables bounded by a finite value, then we say that D has the *extended bounded spreading variable property* (is *e.b.s.v.* for short). ❑

The following lemma proves that, for e.b.s.v. derivations, reductions via RR* give rice to resolvents limited in size. It allows to extend the completeness of the $EVRR_L$ loop check to programs with e.b.s.v. RSLD-derivations.

Lemma 4.4 *Let P be a function-free program and G_0 a goal. If an RSLD-derivation D of $P \cup \{G_0\}$ via RR* is e.b.s.v., then the length of every reduced resolvent N_k in D is bounded.*
Proof. Let us note that every program P has a finite number of predicate symbols and constants. Let $\{c_1,...,c_m\}$ and $\{p_1,...,p_n\}$ be the set of constants and predicate symbols of P, respectively. Moreover, given a goal G_0 it is always true that $|var(G_0\theta)| \le |var(G_0)|$ for any function-free substitution θ. As a consequence, the number of variables in the instantiated goal, i.e., $|var(G_0\theta_0...\theta_k)|$, is bounded for any j.

Let $S_1, ..., S_N$ be the sections in the resolvent G_k of D. By hypothesis, each section S_i (i= 1, ..., N) has a bounded number of spreading variables. Let $\{s_{i_1},...,s_{i_q}\}$ be the set of spreading variables of S_i. Let $(N_k, \alpha_k) = RR^*(G_k, var(G_0\theta_0...\theta_{k-1}))$ and let $S'_{i_1}, ..., S'_{i_M}$ be the reduced sections in N_k obtained according to Definition 4.3. We show that N_k has a bounded number of atoms by proving that the number M of such reduced sections is bounded and that any reduced section S'_i for i=1, ..., M has a bounded number of atoms.

Let us consider a section S_i with i = 1, ..., N. An atom of S_i with predicate symbol p_t, $1 \le t \le n$, of arity h, can be obtained by disposing:
1) (with repetitions) elements of the set: $\{s_{i_1},...,s_{i_q}\} \cup var(G_0\theta_0...\theta_{k-1}) \cup \{c_1,...,c_m\}$ in correspondence of j ($0 \le j \le h$) arguments of p_t,
2) svo variables $\notin var(G_0\theta_0...\theta_{k-1})$ in correspondence of the remaining h-j arguments.

Since the number of svo variables $\notin var(G_0\theta_0...\theta_{k-1})$ is not bounded and equal atoms are allowed, the number of atoms in S_i that have the same disposition according to 1) may be not bounded.

However, by step i) of RR*, equal atoms and equal atoms up to svo variables renaming (not belonging to the instantiated goal) are reduced to a single element. Then for any disposition according to 1), only one atom is left in S_i. Since the number of spreading variables in any section, constants and variables in $var(G_0\theta_0...\theta_{k-1})$ is bounded, the number of possible dispositions according to 1) for any p_t is bounded by a number L_t. Thus, the step i) of RR* reduces any section S_i for i=1, ..., N to a set S'_i with a number of atoms bounded by nL where $L = \max\{L_t \mid 1 \le t \le n\}$.

Let us consider the reduced sections $S'_1, ..., S'_N$ obtained from step i) of RR*. Since the number of spreading variables not belonging to the same section is not bounded, the number of the reduced sections S'_i may be not bounded, too.

However, for a fixed set of spreading variables $\{s_{i_1},...,s_{i_q}\}$ the set S'_i is identified by the choice (with repetition) of s predicate symbols $p_1, p_2, ..., p_s$ with $s \le n$ and for any p_r ($r \le s$) by a disposition according to 1) and 2). Then, since the number of spreading variables in any section is bounded by a fixed value it follows that the number of the possible sections S'_i given by these choices is bounded by a value H. Now, the step ii) of RR* eliminates a section S'_i if it is included in another section S'_j (for $i \ne j$ and $i, j \in \{1, ..., N\}$) up to renaming of variables $\notin var(G_0\theta_0...\theta_{k-1})$.

Thus, after step ii) of RR*, the set of reduced sections $S'_1, ..., S'_N$ is cut down and a bounded set $S'_1,..., S'_M$ of reduced sections is obtained. Since any reduced section S'_i for i = 1, ..,M has a bounded number of atoms, the whole reduced goal N_k contains a bounded number of atoms. ❑

Thus, using the previous Lemma 4.4, the completeness of the $EVRR_L$ loop check for e.b.s.v. derivation via RR* can be stated following the same proof given in [4] for b.s.v. derivation via $\bar{R}\bar{R}$:

Theorem 4.1 (Completeness of $EVRR_L$) *Let P be a function free program, G_0 a goal in L_P. Any infinite e.b.s.v. RSLD-derivation of $P \cup \{G_0\}$ via RR* is pruned by $EVRR_L$.* ❑

5 Classes of Programs for Which EVRR Via RR* is Complete

In [4] it is shown that the $EVRR_L$ loop check, via the reduction rule $\bar{R}\bar{R}$, is complete for a quite large class of function-free programs. Indeed, it is proved that the $EVRR_L$ check is complete not only for function-free restricted, svo and nvi programs, but also for a class of function-free esvo programs. Indeed, the loop check based on equality becomes complete because it is combined with the technique of reduction of resolvents proposed in the paper. This seems to suggest that the necessity of using subsumption loop checks instead of equality checks is essentially due to the presence of redundant atoms in resolvents. All these results of completeness are proved using Theorem 3.1, by showing that, for these classes of programs, derivations have the b.s.v. property. Now, notice that the b.s.v. property implies the e.b.s.v. property. Thus, the following results can be state:

Theorem 5.1 (Completeness of $EVRR_L$ for RSLD-derivation via RR*)
i) The $EVRR_L$ check is complete w.r.t. the leftmost selection rule for function-free restricted programs.
ii) The $EVRR_L$ check is complete for function-free nvi, svo and esvo programs. ❑

We previously stressed that the reduction rule RR* is more powerful than $\bar{R}\bar{R}$. As a matter of fact, let us consider the following example:

Example 5.1 Let P be a function free program with clauses:
C_1.= A(X, Y) ← B(X,Y), A(X,Y) and C_2.= B(X, Y) ← A(W,Z), D(W,Z), B(X,Y).
Let us consider the goal G_0 = ← $B(X_0,Y_0)$, $A(X_0,Y_0)$, $D(X_0,Y_0)$ and the following derivation D:

$$\leftarrow B(X_0,Y_0),\ A(X_0,Y_0),\ D(X_0,Y_0)$$
$$\Downarrow$$
$$\leftarrow A(X_1,Y_1),\ D(X_1,Y_1),\ B(X_0,Y_0),\ A(X_0,Y_0),\ D(X_0,Y_0)$$
$$\Downarrow$$
$$\cdots\cdots$$
$$\Downarrow$$
$$\leftarrow B(X_n,Y_n),\ A(X_n,Y_n),\ D(X_n,Y_n),\ \ldots B(X_i,Y_i),\ A(X_i,Y_i),\ D(X_i,Y_i),\ \ldots,\ B(X_0,Y_0),\ A(X_0,Y_0),\ D(X_0,Y_0)$$
$$\Downarrow$$
$$\cdots\cdots$$

It can be easily verified that D is extended b.s.v., but not b.s.v. Moreover, the corresponding RSLD derivation via RR* is pruned by $EVRR_L$, whereas the corresponding RSLD derivation via $\bar{R}\bar{R}$ is not. Indeed, the RSLD derivation via RR* is:

$$(G_0\ ;\ G_0)$$
$$\Downarrow$$
$$(\leftarrow A(X_1,Y_1),\ D(X_1,Y_1),\ B(X_0,Y_0),\ A(X_0,Y_0),\ D(X_0,Y_0)\ ;\ G_0)$$
$$\Downarrow$$
$$\cdots\cdots$$

that is pruned by $EVRR_L$ after the first step. ❑

It can be verified the program P in the Example 5.1 does not belong to any of the classes of restricted, nvi svo and esvo programs. Indeed, a new classe of programs can been identified. Such a class properly contains the nvi programs and, for this reason, it can be considered an extension of this class. More formally, let us give the following definition:

Definition 5.1 A clause C has the *extended non variable introducing* property (in short is *envi*) if every atom in the body of C, or contains no new variable or contains only new variables. A program P is *envi* if every clause in P is envi. ❑

The Example 5.1 shows that the loop check $EVRR_L$ via $\bar{R}\bar{R}$ is not complete for envi programs. The loop check $EVRR_L$ via RR* is complete for this new class of programs on the basis of the following result:

Lemma 5.1 *Let P be a function-free envi program and let G_0 be a goal in L_P. Let D be an RSLD-derivation of $P \cup \{G_0\}$ then D is extended b.s.v..*
Proof. Let us consider a resolvent G_i of D and let S_1, ..., S_k be the sections of G. Suppose that L is the selected atom of G_i, H ← A_1, ..., A_n the input clause and θ_i the mgu that unifies L and H.

If S_j is a section in G_i that non contains the selected atom L, then it is not affected by θ_i. On the other hand, if S_j is the section containing L in G_i, then, let us consider an atoms $A_j\theta_i$, for j = 1, ..., n introduced in the resolvent G_{i+1}. Since P is envi, two cases are possible:
i) $var(A_j) \subseteq var(H)$
Since θ_i is an mgu of L and H then $var(A_j\theta_i) \subseteq var(H\theta_i) = var(L\theta_i)$ then, no new spreading variable is introduced.
ii) $var(A_j)$ contains only new variables
In this case, the new section of $A_j\theta_i$ in G_i contains only atoms which are introduced by the body of the input clause. Thus, the number of variables in this section is bounded since the bodies of the clauses of a program are of finite size.
Therefore, since in any case the number of variables in every section of G_{i+1} is bounded, it can be state that D has the e.b.s.v. property. ❑

Theorem 5.2 *The $EVRR_L$ check is complete via RR* for function-free envi programs.*
Proof *By lemma 5.1 and Theorem 4.1.* ❑

6 Conclusions

In the paper, the introduced notion of section formalizes the idea of set of atoms chained by common variables. Such a concept, combined with the technique of reduction of resolvents allows to define a more powerful reduction rule that enforces the completeness of the equality loop check.

It is worth to note that, though we studied the loop checking problem in the context of function-free programs, our results can be easily generalized for programs with bounded term size property (b.t.s. for short). This property states that the nesting of functional terms do not grow beyond a certain limit during the computation. In order to state such

generalization, let us note that the Lemma 4.17 in [2] could be easily extended. Indeed, if P is a b.t.s. logic program, then for every finite set of variables V, goal G and $k \geq 1$, the relation $\sim_{V,G,k}$ has only finitely many equivalence classes. As a consequence, the Theorem 4.1 can be easily extended to b.t.s. programs.

References

[1] K.R. Apt. Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science* , volume B, pages 493-574. Elsevier, 1990.

[2] R.N. Bol, K.R. Apt and J.W Klop. An Analysis of Loop Checking Mechanisms for Logic Programs. In *Theoretical Computer Science* , volume 86, pages 35-79, 1991.

[3] R.N. Bol. Loop Checking in Logic Programming Ph.D. Thesis. Centrum voor Wiskunde en Informatica, Amsterdam, NL, 1991.

[4] F. Ferrucci, G. Pacini and M.I. Sessa. Redundancy Elimination and Loop Checks for Logic Programs. In *Information and Computation* , to appear .

[5] J.W. Lloyd. Foundations of Logic Programming. Springer-Verlag, Berlin, second edition, 1987.

[6] A. Van Gelder. Efficient Loop Detection in PROLOG Using the Tortoise-and-Hare Technique. In *Journal of Logic Programming*, volume 4, pages 23-31, 1987.

[7] L. Vieille. Recursive Query Processing: The Power of Logic. In *Theoretical Computer Science*, volume 69, pages 1-53, 1989.

Solving Systems of Equations over Hypersets

Agostino Dovier[♡] Eugenio G. Omodeo[◇]

Alberto Policriti[♣] Gianfranco Rossi[♠]

Abstract

A universe composed by rational ground terms is characterized (both constructively and axiomatically), where the interpreted set constructs $\underline{\emptyset}$ and `with` (the latter designating the element insertion operation) coexist with free Herbrand functors. Ordinary syntactic equivalence must be superseded by an equivalence relation $\approx$, between trees labelled over a signature, that suitably reflects the semantics of `with`. Membership (definable as "$d \in t =_{\mathrm{Def}} (t \texttt{ with } d) \approx t$") meets the non-well-foundedness property characteristic of hyperset theory. An algorithm for solving the NP-complete unification problem pertaining to hollow hyperset terms is provided, and shown to be totally correct. An application to the matching of finite state automata is hinted at.

1 Introduction

With this paper we wish to contribute to the field of constraint logic programming mainly in two ways. First, by characterizing *hypersets* both constructively and axiomatically (cf. Sec. 2, 3); then, by specifying a procedure for normalizing constraints of the most basic kind over hypersets. Our procedure will be shown to always terminate and to give the proper results (cf. Sec. 4). A by-product of our constraint normalization procedure is an algorithm for solving a novel NP-complete *unification* problem pertaining to hypersets, which has direct applications on its own: it can be used, for instance, to determine whether two finite state automata characterize the same language (cf. Sec. 5).

Hypersets of the kind we will propose are very intimately related to Aczel's non-well-founded sets (cf. [1]), but they all have finite cardinality and height; moreover they are *hybrid*, in that their construction involves free Herbrand functors. The finiteness restriction comes from our willingness to regard hypersets as instances of an algorithmic data structure; the proposed hybridization comes from our goal to

♡ *Univ. di Pisa*, Dip. di Informatica. Corso Italia 40, 56100-Pisa (I). dovier@di.unipi.it

◇ *Univ. di Roma "La Sapienza"*, Dip. di Inf. e Sist. Via Salaria, 113. 00198-Roma. omodeo@assi.dis.uniroma1.it

♣ *Univ. di Udine*, Dip. di Matematica e Informatica. Via Zanon, 6. 33100-Udine (I). policrit@dimi.uniud.it

♠ *Univ. di Parma*, Dip. di Matematica. Via M. D'Azeglio, 85/A. 43100-Parma (I). grossi@prmat.math.unipr.it