OPTIMAL GROUNDNESS ANALYSIS USING PROPOSITIONAL LOGIC

AGOSTINO CORTESI, GILBERTO FILE', AND WILLIAM WINSBOROUGH

▷ It is well-known that propositional formulas form a useful and computationally efficient abstract interpretation for different data-flow analyses of logic programs and, in particular, for groundness analysis. This article gives a complete and precise description of an abstract interpretation, called **Prop**, composed of a domain of positive, propositional formulas and three operations: abstract unification, least upper bound, and abstract projection. All three abstract operations are known to be correct. They are shown to be optimal in the classical sense. Two alternative stronger notions of optimality of abstract operations are introduced, which characterize very precise analyses. We determine whether the operations of **Prop** also satisfy these stronger forms of optimality. ◁

1. INTRODUCTION

Abstract interpretation, a technique for constructing verified analyses of program execution behavior [4], has been extensively applied to Prolog programs. A comprehensive list of references can be found in [5]. More recent contributions include among others [1, 8, 28, 29, 33].

In general, an abstract interpretation framework provides a (collecting) semantics that is parameterized with respect to the computation domain and the operations used. By instantiating these parameters with the concrete domains and operations, one obtains a concrete semantics; instantiating the parameters with non-standard (abstract) domain and operations, one obtains a static analysis (i.e., a abstract semantics). We call the combination of a domain and operations on it an interpretation, which may be either concrete or abstract. The framework also provides

Address correspondence to Agostino Cortesi, University of Venezia, Via Torino 155, I-30170 Mestre-VE, Italy, or Gilberto File', University of Padova, Via Belzoni 7, I-35131 Padova, Italy, or William Winsborough, Pennsylvania State University, Pond Laboratory, University Park, PA 16802.

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/94/$7.00

safety conditions on the abstract interpretation and the *concretization* function, which maps abstract domain elements to concrete ones (sets of substitutions in the case of Prolog). When the safety conditions are met, the induced analysis is guaranteed to safely approximate the concrete semantics.

Probably the most well-known analysis of pure Prolog programs is the analysis of *groundness*. (See for instance [18].) One candidate abstract domain for representing groundness consists of sets V of variables, where V represents the set of substitutions that ground each variable in V. However, such a domain fails to capture the propagation of groundness among program variables bound to terms that contain the same free variables. This motivates extending the domain to capture also *equivalence of variables:* two sets of variables S_1 and S_2 are equivalent with respect to a set of substitutions Σ if

$$\sigma \in \Sigma \Longrightarrow \bigcup_{x \in S_1} Var(\sigma x) \;=\; \bigcup_{x \in S_2} Var(\sigma x),$$

where $Var(t)$ is the set of variables occurring in term t. The significance of this information is that, at any subsequent stage of the computation, all variables in S_1 become ground if and only if all variables in S_2 become ground, too.

Marriott and Søndergaard [26, 27] have proposed the use of propositional formulas for representing variable groundness and equivalence. Intuitively, the formula $x \wedge y$ says that x and y are both ground; the formula $x \vee y$ says that either x or y is ground, or both. The formula $x \wedge y \wedge z \leftrightarrow u \wedge w$ expresses the equivalence of $\{x, y, z\}$ and $\{u, w\}$. It approximates, for instance, the substitution $\{u \mapsto f(x, y), w \mapsto f(y, z)\}$.

The aim of this paper is to provide a precise description of an abstract interpretation for groundness analysis based on propositional formulas.

1. We characterize both semantically and syntactically the class of formulas that are meaningful for groundness analysis, that is, the formulas that approximate non-empty sets of substitutions and such that any two inequivalent formulas represent distinct sets of substitutions. This class consists of (the equivalence classes of) the formulas that are true when all their variables are set to true [31]. These formulas are called *positive* [28]. Also a syntactic characterization of this class is given by showing that the class of positive formulas equals that of the formulas obtained by using only the connectives, $\wedge$ and $\leftrightarrow$. A preliminary report of these results appeared in [11]. In addition to this characterization, we show that all inequivalent positive formulas are useful for deriving different groundness properties in real programs.
2. The abstract domain *Prop*, using positive formulas, is defined. Our domain *Prop* consists of elements of the form $[f, U]$, where f is (the equivalence class of) a positive formula (or the false constant, F) whose variables are all contained in the finite set U. Intuitively, U contains the *variables of interest*, i.e. all the variables about which the abstract value f is supposed to give information. When bottom and top elements are added to the set of these pairs, *Prop* becomes a complete lattice with respect to the following partial order:

   $$\text{for } [f_1, U_1], [f_2, U_2] \in \mathit{Prop},\ [f_1, U_1] \le [f_2, U_2] \text{ iff } f_1 \models f_2 \text{ and } U_1 = U_2$$

   Thus, *Prop* comprises separate complete finite lattices for each finite set of

variables, plus a bottom and a top.
We also maintain explicitly the set of variables of interest in our definition of the concrete domain *Rsub* (*Rsub* stands for restricted substitutions). An element of *Rsub* is a pair $[\Sigma, \mathrm{U}]$, where Σ is a set of substitutions and U a finite set of variables[1]. Adding top and bottom elements, *Rsub* becomes a complete lattice with respect to a partial order similar to that defined above for *Prop*. Our motivation for explicitly including the variables of interest in the domains is discussed in Section 7.

3. The abstract operations on *Prop* are defined, obtaining the abstract interpretation **Prop**. The least upper bound (lub) and the projection are natural extensions of disjunction and existential quantification. This extension is needed mainly to handle the second component of the elements of *Prop*. Thus, they are very similar to the corresponding operations defined in [29]. The abstract unification $\mathbf{U}_p$ presents several differences with respect to that of [29], (cf. the discussion in Section 4.2).
All three abstract operations are correct with respect to the corresponding concrete ones, as observed in [28, 29]. Moreover, we show that these three operations are also optimal. To the best of our knowledge, these are the first optimality results shown for an abstract interpretation for logic programs.
4. Two new forms of optimality, α- and γ-optimality, are defined and applied in our study of **Prop**. Both α- and γ-optimality imply the usual optimality of [4], and have interesting implications. Assume that the dataflow semantics of a given logic program P is a function, $\mathcal{P}[\![P]\!] : Atom \to X \to X$, as in [29], where X is a generic domain and $Atom$ is the set of atoms. Let **C** and **D** be a concrete and an abstract interpretation, whose domains C and D are related by means of a Galois insertion with adjoined functions, α and γ. Let us now use **C** and **D** to interpret appropriately the domain X and the operation symbols of the dataflow semantics in order to obtain a concrete and an abstract semantics that are functions $\mathcal{P}[\![P]\!]_C$: $Atom \to C \to C$ and $\mathcal{P}[\![P]\!]_D$: $Atom \to D \to D$, respectively.
If the operations of **D** are α-optimal with respect to those of **C**, then $\forall c \in C$ and $\forall B \in Atom$, $\alpha(\mathcal{P}[\![P]\!]_C\ B\ c) = \mathcal{P}[\![P]\!]_D\ B\ (\alpha c)$. Thus, α-optimality implies that the abstract semantics is the best possible with respect to the chosen abstract domain D. The reader may wonder whether this fact is already true when normal optimality, [4], holds instead of α-optimality. In general, this is not the case. We show that, even though **Prop** is optimal in the classic sense, the above property does not hold and the unification of **Prop** is not α-optimal).
Let us turn to γ-optimality, which implies

$$\forall d \in D \text{ and } B \in Atom, \gamma(\mathcal{P}[\![P]\!]_D\ B\ d) \ = \ \mathcal{P}[\![P]\!]_C\ B\ (\gamma d)$$

This property is even stronger than the previous one. In fact, it implies that no loss of information is caused by computing on D instead of C (when the concrete computation starts from a value that is the image of an abstract value). Thus, γ-optimality is an unrealistic condition to require, since it implies that the abstract domain is in some sense equivalent to the concrete

[1] Unlike in approaches based on parametric substitutions [27], the domain of $\sigma \in \Sigma$ has no bearing on the variables of interest, U.

one. The results we show about these strong forms of optimality are in agreement with the above intuition: the lub and projection of **Prop** are α-optimal but not γ-optimal; the abstract unification $\mathbf{U}_p$ is neither α- nor γ-optimal.

Let us conclude this introduction with an example that shows how **Prop** can be used for the groundness analysis of a Prolog program. In order to focus attention on the use of the operations, the example omits some steps performed by a typical, general-purpose analysis.

Example 1.1. Consider the following program fragment.

$$\texttt{?-}\ ^{\langle 0\rangle}p(u,v,w),^{\langle 1\rangle}\ q(u),^{\langle 2\rangle}\ v=x,^{\langle 3\rangle}\ w=x,^{\langle 4\rangle}\ use(x).$$

$p(z_1, z_1, z_2).$ $\quad\quad$ $q(c).$ $\quad\quad$ $use(x)$ `:-` $\ldots.$
$p(z_1, z_2, z_1).$

A typical concrete collecting semantics computes a set of substitutions (called the *activation set*) for each program point, whereas an abstract semantics computes an abstract activation approximating that set. Using **Prop** as the abstract interpretation, we trace the behavior of the program under both semantics.

Let $W = \{u, v, w, x\}$ and assume that at point $\langle 0\rangle$ the activation set is $[\{id\}, W]$, where id stands for the empty substitution. The corresponding abstract activation is: $[\mathsf{T}, W]$ (where T stands for true). The concrete unification of $p(u, v, w) = p(z_1, z_1, z_2)$ produces $[\{\{u \mapsto v, z_1 \mapsto v, z_2 \mapsto w\}\}, W \cup \{z_1, z_2\}]$. Abstract unification produces $d = [(u \leftrightarrow z_1) \wedge (v \leftrightarrow z_1) \wedge (w \leftrightarrow z_2), W \cup \{z_1, z_2\}]$. Note that in both concrete and abstract activations, the second component has been modified to account for the introduction of the new variables z_1 and z_2. As part of computing the abstract activation at point $\langle 1\rangle$, d is projected onto the variable set W by means of existential quantification:

$$[\exists\{z_1, z_2\}.(u \leftrightarrow z_1) \wedge (v \leftrightarrow z_1) \wedge (w \leftrightarrow z_2), W] \ \equiv [u \leftrightarrow v, W]$$

The corresponding concrete projection simply replaces the set of variables of interest by W:

$$[\{\{u \mapsto v, z_1 \mapsto v, z_2 \mapsto w\}\}, W].$$

In the remainder of the example, we show only the results after projection, and we omit the second component from activations, because it is always W.

In both the concrete and the abstract cases, the result of using the second clause for p is similar to that of the first. Results from the two clauses are combined by using the join operations of the respective domains. At point $\langle 1\rangle$ the concrete activation set is $\{\ \{u \mapsto v, z_1 \mapsto v, z_2 \mapsto w\}, \{u \mapsto w, z_1 \mapsto w, z_2 \mapsto v\}\ \}$ and the abstract activation is $(u \leftrightarrow v) \vee (u \leftrightarrow w)$.

When a new unification is simulated, such as that of $q(u) = q(c)$, the abstraction of the *mgu* is conjoined with the previous abstract context. In this case the concrete *mgu* is $\{u \mapsto c\}$, which is abstracted simply by u, so we obtain[2] at program point

[2]The attentive reader may be justifiably concerned that in standard execution the current substitution is applied to the equation before its *mgu* is found. We take advantage of the fact that for idempotent substitutions σ and equation sets E, $mgu(\sigma(E)) \circ \sigma$ is renaming-equivalent

$\langle 2\rangle$:

$$\left\{\begin{array}{l}\{u \mapsto c, v \mapsto c, z_1 \mapsto c, z_2 \mapsto w\}, \\ \{u \mapsto c, w \mapsto c, z_1 \mapsto c, z_2 \mapsto v\}\end{array}\right\} \text{ and } \begin{array}{l}((u \leftrightarrow v) \vee (u \leftrightarrow w)) \wedge u \\ \equiv u \wedge (v \vee w)\end{array}$$

We obtain at point $\langle 3\rangle$

$$\left\{\begin{array}{l}\{u \mapsto c, v \mapsto c, x \mapsto c, z_1 \mapsto c, z_2 \mapsto w\}, \\ \{u \mapsto c, w \mapsto c, v \mapsto x, z_1 \mapsto c, z_2 \mapsto x\}\end{array}\right\} \text{ and } \begin{array}{l}(u \wedge (v \vee w)) \wedge (v \leftrightarrow x) \\ \equiv u \wedge ((v \wedge (v \leftrightarrow x)) \\ \quad \vee (w \wedge (v \leftrightarrow x))) \\ \equiv u \wedge ((v \wedge x) \vee (w \wedge (v \leftrightarrow x)))\end{array}$$

and at program point $\langle 4\rangle$

$\{\{u \mapsto c, v \mapsto c, x \mapsto c, w \mapsto c, z_1 \mapsto c, z_2 \mapsto c\}\}$ and $u \wedge v \wedge w \wedge x$

which tells us that each variable is ground.

The rest of the paper is organized as follows. Section 2 recalls some basic notation and properties of abstract interpretation, of propositional formulas, and of substitutions. In particular, Section 2.4 explains the function that maps formulas to sets of substitutions and shows some of its properties. Section 3 gives a semantic and a syntactic characterization of the formulas that are useful for groundness analysis. Section 4 precisely defines and illustrates the concrete and abstract interpretations, **Rsub** and **Prop**. In Section 5 the relation between **Prop** and **Rsub** is studied: in Section 5.1, we show that there is a Galois insertion between their two domains; Sections 5.2 contains some interesting lemmas relating a value of **Rsub** with its abstraction. These results are used in Section 5.3, where we show the correctness and optimality of the abstract unification and projection. In Section 6 we study stronger optimality results for the abstract operations. Finally, the paper is closed by a section discussing related work, followed by a short conclusion.

2. PRELIMINARIES

2.1. Basic Notions Concerning Abstract Interpretation

As mentioned in the Introduction, in a general framework one obtains an abstract analysis by defining an abstract domain and operations that simulate the concrete ones and that meet some safety conditions. In this subsection we recall the most common safety conditions that are used in the literature [4, 5].

Let the concrete interpretation be $\mathbf{C}=(C, op_C)$ and $\mathbf{A}=(A, op_A)$ be an abstract interpretation. For the sake of simplicity we assume, without loss of generality, that the interpretations have only one operation and that these operations are unary. Both C and A are assumed to be complete lattices with partial orders $\leq_C$ and $\leq_A$, respectively.

A *Galois connection* between C and A is defined by two functions γ: $A \longrightarrow C$, called *concretization*, and α: $C \longrightarrow A$, called *abstraction*. They must satisfy three conditions: (i) α and γ are monotonic; (ii) $\forall c \in C$, $c \leq_C \gamma(\alpha(c))$; (iii) $\forall a \in A$, $\alpha(\gamma(a)) \leq_A a$. In that case, γ and α are called *adjoint*. A Galois connection is denoted by (A, γ, C, α). A Galois connection (A, γ, C, α) that has γ injective is

to $mgu(E \cup Eq(\sigma))$.

called a *Galois insertion*. In this case, condition (iii) above becomes an equality. The abstract operation op_A is *correct* with respect to op_C when

$$\forall a \in A, \alpha(op_C(\gamma(a))) \ \leq_A \ op_A(a)$$

and op_A is *optimal* if that inequality can be replace by an equality.

2.2. Propositional Formulas

For an introduction to the basic concepts of propositional logic, see for instance [2]. Let $\mathbf{V}$ be a countably infinite set of propositional variables. $FP(\mathbf{V})$ denotes the set of finite subsets of variables of $\mathbf{V}$. The set of propositional formulas constructed over the variables of $\mathbf{V}$ and the logical connectives in $\Gamma \subseteq \{\wedge, \vee, \leftrightarrow, \neg\}$ is denoted by $\Omega(\Gamma)$. For any $U \in FP(\mathbf{V})$, $\Omega_U(\Gamma)$ consists of formulas using only the variables of U and the connectives of Γ. The propositional constants T and F (for *true* and *false*) are not included unless otherwise indicated.

A *truth assignment* is a function $r : \mathbf{V} \longrightarrow \{true, false\}$. Given a formula $f \in \Omega\{\wedge, \vee, \leftrightarrow, \neg\}$[3], $r \models f$ means that r satisfies f and $f_1 \models f_2$ is a shorthand for "$r \models f_1$ implies $r \models f_2$." $\Omega\{\wedge, \vee, \leftrightarrow, \neg\}$ is ordered by $f_1 \sqsubseteq f_2$ if $f_1 \models f_2$. Two formulas f_1 and f_2 are logically equivalent, denoted $f_1 \equiv f_2$, if $f_1 \models f_2$ and $f_2 \models f_1$.

We are interested in the lattice obtained by taking the quotient of $\Omega(\Gamma)$ with respect to logical equivalence. To avoid burdensome notation, in the rest of the paper we simply write f for the class of formulas equivalent to f. Sometimes we implicitly select a representative of this equivalence class by treating f first as an equivalence class and then as a formula in that equivalence class. Since the results of the operations that we apply to f do not depend on which representative formula is selected, this abuse neither causes nor hides any problems. Notice that for any $U \in FP(\mathbf{V})$, $\Omega_U\{\wedge, \vee, \leftrightarrow, \neg\}/_{\equiv}$ is a complete lattice with least upper bound $\vee$ and greatest lower bound $\wedge$.

For subsets A_1 and A_2 of $\Omega\{\wedge, \vee, \leftrightarrow, \neg\}$, we write $A_1 \equiv A_2$ when every formula of one is equivalent to a formula of the other and vice versa. For example, it is well-known that $\Omega\{\wedge, \vee, \leftrightarrow, \neg\} \equiv \Omega\{\wedge, \neg\}$.

Let the truth values be ordered by *false* $\leq$ *true* and extend this order pointwise to truth assignments: $r_1 \leq r_2$ if for all $x \in \mathbf{V}$, $r_1(x) \leq r_2(x)$. A formula f is *monotonic* if for all r_1 and r_2 satisfying $r_1 \leq r_2$, $r_1 \models f$ implies $r_2 \models f$. A formula f has the model-intersection property if, for all truth assignments r_1 and r_2, $r_1 \models f$ and $r_2 \models f$ implies $r \models f$, where r is defined so that for each $x \in \mathbf{V}$, $r(x) = true$ if and only if $r_1(x) = r_2(x) = true$.

In addition to truth assignments we use substitutions that replace a finite set of variables by an element of $\{\mathsf{T}, \mathsf{F}\}$. Let $U = \{x_1, \ldots, x_n\}$. A function $s : U \longrightarrow \{\mathsf{T}, \mathsf{F}\}$ is called a *truth substitution* over U. The application of truth substitutions to a formula f is denoted by $s(f)$. We also write $f\langle x/Q\rangle$ for $\langle x/Q\rangle(f)$, where Q is a truth value and $\langle x/Q\rangle$ is the truth substitution mapping x to Q. If r is a truth

[3] We often write $\Omega\{\wedge, \vee, \leftrightarrow, \neg\}$ for $\Omega(\{\wedge, \vee, \leftrightarrow, \neg\})$, $\alpha\{\sigma\}$ for $\alpha(\{\sigma\})$, σx for $\sigma(x)$, and even $assign\, \sigma$ for $assign(\sigma)$ when the extra parentheses seem to make the function application more difficult to read.

assignment, we denote by $r|_U$ the truth substitution $s : U \longrightarrow \{\mathsf{T}, \mathsf{F}\}$ given by

$$s(x) = \begin{cases} \mathsf{T} \text{ if } r(x) = \mathit{true} \\ \mathsf{F} \text{ if } r(x) = \mathit{false} \end{cases}$$

We conclude this subsection with some simple propositions.

Proposition 2.1. For all $f \in \Omega\{\wedge, \vee, \leftrightarrow, \neg\}$, such that $f \not\equiv \mathsf{F}$, f is monotonic if and only if there exists some $f' \in \Omega\{\wedge, \vee\}$ such that $f' \equiv f$.

Proposition 2.2. $\Omega\{\wedge, \vee, \leftrightarrow\} = \Omega\{\vee, \leftrightarrow\} = \Omega\{\wedge, \leftrightarrow\}$.

Proof. Follows from observing

$$f_1 \vee f_2 \equiv (f_1 \leftrightarrow f_2) \leftrightarrow ((f_1 \wedge f_2) \wedge (f_1 \leftrightarrow f_2))$$
$$f_1 \wedge f_2 \equiv (f_1 \leftrightarrow f_2) \leftrightarrow (f_1 \vee f_2)$$

□

Proposition 2.3. $\Omega\{\wedge, \vee\} \subset \Omega\{\wedge, \leftrightarrow\}$. *(Note that the inclusion is strict.)*

Proof. By Propositions 2.1 and 2.2, it suffices to show that there is a formula in $\Omega\{\wedge, \leftrightarrow\}$ that is not monotonic. Define r_1 by $r_1(x) = \mathit{false}$ for all $x \in \mathbf{V}$; define r_2 to be the same as r_1 except that $r_2(x_1) = \mathit{true}$; define $f = x_1 \leftrightarrow x_2$. We have $r_1 \leq r_2$, $r_1 \models f$, and not $r_2 \models f$. □

The unit assignment, u is defined by $u(x) = \mathit{true}$ for all $x \in \mathbf{V}$. Define the set of positive formulas by:

$$\mathit{Pos} = \left\{ f \in \Omega\{\wedge, \vee, \leftrightarrow, \neg\} \,\middle|\, u \models f \right\}$$

Some obvious examples: $\mathsf{T}, x_1 \in \mathit{Pos}$ and $\mathsf{F}, \neg x_1 \notin \mathit{Pos}$.

Proposition 2.4. $\Omega\{\wedge, \leftrightarrow\} \subseteq \mathit{Pos}$.

Proof. Straightforward, by structural induction on the formula in $\Omega\{\wedge, \leftrightarrow\}$. □

Proposition 2.5. $\Omega\{\wedge, \leftrightarrow\} \subset \Omega\{\wedge, \vee, \leftrightarrow, \neg\}$.

Proof. By Proposition 2.4, the formula $\neg x_1$ cannot be expressed in $\Omega\{\wedge, \leftrightarrow\}$. □

The expressive powers of $\Omega\{\wedge, \leftrightarrow\}$ and $\Omega\{\wedge, \neg\}$ are more similar than one might at first expect. The former has a sort of quasi-negation, illustrated by the following proof.

Theorem 2.1 [24]. Consider any $f \in \Omega\{\wedge, \neg\}$. Assume that all variables of f are $\mathbf{V}_n = \{x_1, \ldots, x_n\}$. Then there exists $f' \in \Omega\{\wedge, \leftrightarrow\}$ such that $f' \equiv f \vee \bigwedge \mathbf{V}_n$.

Proof. Inductively construct f' from f by replacing each subformula of the form $\neg f^*$ by $f^* \leftrightarrow \bigwedge \mathbf{V}_n$. We prove $f' \equiv f \vee \bigwedge \mathbf{V}_n$ by showing that for all assignments r, $r \models f'$ iff $r \models f \vee \bigwedge \mathbf{V}_n$

case 1: $r \models \bigwedge \mathbf{V}_n$. By construction, $f' \in \Omega\{\wedge, \leftrightarrow\}$, so by Proposition 2.4, $r \models f'$. The result follows immediately by the semantics of $\vee$.

case 2: $r \not\models \bigwedge \mathbf{V}_n$. The proof that $r \models f$ iff $r \models f'$ is by induction on the structure of f. The basis is trivial: f is a propositional variable. In the inductive step, two cases are distinguished. If $f = f_1 \wedge f_2$ then $f' = f_1' \wedge f_2'$ and thus $r \models f$ iff $r \models f'$ by the induction hypothesis. In the other case $f = \neg f_1$, thus $f' = f_1' \leftrightarrow \bigwedge \mathbf{V}_n$.

$$\begin{array}{llll} r \models f' & \text{iff} & r \models f_1' \leftrightarrow \bigwedge \mathbf{V}_n & \text{by construction of } f' \\ & \text{iff} & r \not\models f_1' & \text{by assumption on } r \\ & \text{iff} & r \models \neg f_1 & \text{by induction assumption} \\ & \text{iff} & r \models f & \text{by assumption on } f \quad \square \end{array}$$

We now can observe the equivalence of the syntactic and semantic characterizations of this set of formulas.

Corollary 2.1. $\Omega\{\wedge, \leftrightarrow\} \equiv Pos$.

Proof. This is an immediate consequence of Proposition 2.4 and Theorem 2.1. $\square$

2.3. Substitutions

Since propositional formulas are used to express properties of substitutions, it is convenient to use the same set of variables $\mathbf{V}$ for both formulas and substitutions.

Let $\mathbf{G}$ be an alphabet of function symbols and let $\mathbf{T}_{\mathrm{V,G}}$ denote the set of finite terms over $\mathbf{V}$ and $\mathbf{G}$. A *substitution* σ is a function in $\mathbf{V} \longrightarrow \mathrm{T}_{\mathrm{V,G}}$ such that $\sigma(x) \neq x$ for only a finite number of variables x. The restriction of σ to $A \subset \mathbf{V}$ is given by

$$\sigma|_A(x) = \begin{cases} \sigma(x) & \text{if } x \in A \\ x & \text{otherwise} \end{cases}$$

The *set of support* of σ is given by $supp(\sigma) = \{x | \sigma(x) \neq x\}$. The *variable range* of σ is given by $var\text{-}range(\sigma) = \bigcup\{ Var(\sigma x) | x \in supp(\sigma)\}$, 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 renaming σ is an invertible substitution.

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 also 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. Since all the idempotent most general unifiers of a set of equations E are renamings of one another [23], we use the notation $mgu(E)$ to denote an arbitrary such unifier. If $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is an idempotent substitution, then we write $Eq(\sigma)$ for the set of equations $\{x_1 = t_1, \ldots, x_n = t_n\}$. Note that σ is a most general unifier of $Eq(\sigma)$. *Subst* is the set of idempotent substitutions.

2.4. Representing Substitutions by Formulas

This subsection defines γ, the function that maps a propositional formula to the set of substitutions that it approximates, and that forms the basis of the concretization function of *Prop*, defined in Section 4.1. A substitution's groundness and variable equivalence properties are preserved under instantiation: 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 also equivalent with respect to any $\sigma' \trianglelefteq \sigma$. Thus, to represent the groundness and equivalence of a substitution σ, it is natural that the formulas that represent these properties approximate only instantiation closed sets of substitutions. (This closure property makes abstract unification easily expressible as the greatest lower bound of the abstract domain.) 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. We use an auxiliary function that maps a substitution to a truth assignment that assigns the value *true* to each variable that the substitution grounds [26, 27].

$$\begin{array}{l} \mathit{assign} : \mathit{Subst} \longrightarrow \mathbf{V} \longrightarrow \{\mathit{true}, \mathit{false}\} \\ \quad \mathit{assign}\, \sigma\, x = \mathit{true} \ \mathit{iff}\ \sigma \text{ grounds } x \end{array}$$

In the examples that follow h, g and a, c denote function and constant symbols; f is always a formula.

Example 2.1.

Consider the formulas $f_1 = (\neg x_1 \vee x_2)$ and $f_2 = x_1 \wedge (x_2 \leftrightarrow x_3)$ and the substitutions $\sigma_0 = \{x_1 \mapsto g(x_4)\}$, $\sigma_1 = \{x_2 \mapsto a\}$ and $\sigma_2 = \{x_1 \mapsto g(a),\ x_2 \mapsto h(x_5),\ x_3 \mapsto (h(x_6))\}$. Both $\mathit{assign}\, \sigma_0$ and $\mathit{assign}\, \sigma_1$ satisfy f_1, and $\mathit{assign}\, \sigma_2$ satisfies f_2. However, σ_2 does not possess the equivalence property expressed in f_2 by $x_2 \leftrightarrow x_3$. This is revealed when we consider the instance of σ_2, $\sigma_2' = \{x_1 \mapsto g(a),\ x_2 \mapsto h(a),\ x_3 \mapsto g(h(x_6)),\ x_5 \mapsto a\}$: $\mathit{assign}\, \sigma_2' \not\models f_2$. On the other hand, $\mathit{assign}\, \sigma_2$ and $\mathit{assign}\, \sigma_2'$ both model $x_1 \wedge (x_2 \leftrightarrow x_5)$. The strongest formula that approximates σ_2 is $x_1 \wedge (x_2 \leftrightarrow x_5) \wedge (x_3 \leftrightarrow x_6)$.

These observations motivate the following definition of the function γ [27]:

$$\begin{array}{l} \quad \gamma : \Omega\{\wedge, \vee, \leftrightarrow, \neg\} \longrightarrow \wp(\mathit{Subst}), \\ \gamma(f) = \{\sigma \in \mathit{Subst} : \forall \sigma' \trianglelefteq \sigma \,.\, \mathit{assign}\, \sigma' \models f\}. \end{array}$$

Thus, $\gamma(\mathsf{T}) = \mathit{Subst}$ and $\gamma(\mathsf{F}) = \emptyset$.

Example 2.2. Let $\sigma = \{x_1 \mapsto x_4,\ x_2 \mapsto x_5,\ x_3 \mapsto g(x_4, x_5)\}$. It is easy to see that for all $\sigma' \trianglelefteq \sigma$, $\mathit{assign}\, \sigma' \models f_1 \vee f_2 \vee f_3 \vee f_4$ where,

$$\begin{array}{l} f_1 = \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \\ f_2 = x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4 \wedge \neg x_5 \\ f_3 = \neg x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge x_5 \\ f_4 = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \end{array}$$

Consequently, $\sigma \in \gamma(f_1 \vee f_2 \vee f_3 \vee f_4)$. In fact, $f_1 \vee f_2 \vee f_3 \vee f_4$ is the strongest formula approximating σ. A more compact and evocative equivalent formula is $(x_1 \leftrightarrow x_4) \wedge (x_2 \leftrightarrow x_5) \wedge (x_3 \leftrightarrow (x_4 \wedge x_5))$. This intuition will be formalized in Theorem 5.5.

Proposition 2.6. γ is monotonic.

Proof. Suppose $f_1 \models f_2$. It follows that if $\forall \sigma' \trianglelefteq \sigma.\mathit{assign}\, \sigma' \models f_1$ then $\forall \sigma' \trianglelefteq \sigma.\mathit{assign}\, \sigma' \models f_2$. So we have $\gamma(f_1) \subseteq \gamma(f_2)$. □

Proposition 2.7. For all $f_1, f_2 \in \Omega\{\wedge, \vee, \leftrightarrow, \neg\}$, $\gamma(f_1 \wedge f_2) = \gamma(f_1) \cap \gamma(f_2)$.

PROOF.

$$\begin{aligned}\gamma(f_1 \wedge f_2) &= \{\sigma | \forall \sigma' \trianglelefteq \sigma. assign\, \sigma' \models f_1 \wedge f_2\} \\ &= \{\sigma | \forall \sigma' \trianglelefteq \sigma. assign\, \sigma' \models f_1\} \cap \{\sigma | \forall \sigma' \trianglelefteq \sigma. assign\, \sigma' \models f_2\} \\ &= \gamma(f_1) \cap \gamma(f_2)\end{aligned}$$

□

3. THE FORMULAS USEFUL FOR GROUNDNESS ANALYSIS

Three results are shown in this section. First, we prove that the formulas that represent nonempty sets of substitutions are exactly the formulas of *Pos*. Second, we show that inequivalent members of *Pos* represent different sets of substitutions. Finally, we show that all elements of *Pos* are necessary in the following sense: for any two inequivalent formulas f_1 and f_2, there is a program such that its abstract analysis, starting from f_1 as initial activation, infers different groundness with respect to that computed by the analysis starting from f_2.

Since all representable sets of substitutions are closed under instantiation, nonempty sets must contain substitutions that ground arbitrarily large finite sets of variables. Consequently, nonempty sets are represented by formulas that assume the value true on the unit assignment u.

Proposition 3.1. For each $f \in \Omega\{\wedge, \vee, \leftrightarrow, \neg\}$, $\gamma(f) \neq \emptyset$ if and only if $f \in Pos$.

PROOF.

- $\Rightarrow$) Fix any $\sigma \in \gamma(f)$. From the definition of γ, $\sigma' \trianglelefteq \sigma$ implies $\sigma' \in \gamma(f)$. Clearly there exists $\sigma' \trianglelefteq \sigma$ such that σ' grounds each variable occurring in f. By assumption, $assign\, \sigma' \models f$ and $assign\, \sigma' \models x$ for each $x \in Var(f)$. It follows that $u \models f$.
- $\Leftarrow$) For any $f \in Pos$, $\gamma(f)$ contains all substitutions that grounds each variable of f. □

Proposition 3.1 tells us that if we include in our abstract domain the formulas of *Pos* together with the formula F, there will be only one element representing the empty set of substitutions, namely F. Proposition 3.2 extends this by telling us that there will be only one element representing each represented set.

Proposition 3.2. The function γ restricted to Pos is injective.

PROOF. Consider arbitrary $[f_1]_\equiv$ and $[f_2]_\equiv \in Pos$ such that $[f_1]_\equiv \neq [f_2]_\equiv$. Since f_1 and f_2 are not equivalent there must be a truth assignment $r : \mathbf{V} \longrightarrow \{true, false\}$ such that (without loss of generality) $r \models f_1$ and $r \not\models f_2$. Let σ_r be the substitution

$$\sigma_r(x) = \begin{cases} a, & \text{if } r(x) = true \text{ and } x \in Var(f_1), \\ z, & \text{if } r(x) = false \text{ and } x \in Var(f_1), \\ x, & \text{if } x \notin Var(f_1), \end{cases}$$

where a is some constant and z is some variable satisfying $r(z) = false$. (We know such a z exists because $r \not\models f_2$.) Now for each $\sigma' \trianglelefteq \sigma_r$, σ' either grounds z or it does not. That is

either for all $x \in Var(f_1)$, $assign\, \sigma'(x) = true$
or else for all $x \in Var(f_1)$, $assign\, \sigma'(x) = r(x)$

In the former case, $assign\,\sigma' \models f_1$ follows from $f_1 \in Pos$; in the latter case, we have $assign\,\sigma' \models f_1$ because $r \models f_1$. Thus, $\sigma_r \in \gamma(f_1)$. On the other hand, since we assumed that $r \not\models f_2$, $\sigma_r \notin \gamma(f_2)$. Thus, $\gamma(f_1) \neq \gamma(f_2)$ as desired. □

Together, Proposition 3.1 and Proposition 3.2 characterize semantically the formulas that are useful for representing distinct non-empty sets of substitutions. By Corollary 2.1, these are the formulas of $\Omega\{\wedge, \leftrightarrow\}$.

We conclude this section by showing that if we omit any element from *Pos*, we diminish the power of the analysis. We have shown in Proposition 3.2 that inequivalent elements of *Pos* denote different subsets of *Subst*. This is theoretically interesting; however for compilation, we are typically interested only in which variables are ground at each program point. Thus, we need to distinguish two formulas only if either (1) they have different atomic consequences, or (2) they lead to different atomic consequences after further analysis of some subject program. Since abstract activations are computed by conjoining formulas, it suffices to answer the following question: Given two inequivalent formulas $f_1, f_2 \in Pos$, does there exists a formula that, when conjoined with f_1 and f_2, leads to different atomic consequences? Proposition 3.3 answers that question in the affirmative.

Proposition 3.3. Let $f_1, f_2 \in Pos$ and $f_1 \not\equiv f_2$. There exists a third formula $f_3 \in Pos$ such that one of $f_1 \wedge f_3$ and $f_2 \wedge f_3$ is equivalent to $\wedge\mathbf{V}_n$ and the other is not.

Proof. Follows easily by viewing f_1 and f_2 in their disjunctive normal forms. □

4. THE CONCRETE AND THE ABSTRACT INTERPRETATION

In this section the domain and the operations of both the concrete and the abstract interpretation are defined. The concrete interpretation is **Rsub**=(*Rsub*, $\bigsqcup_c$,$\mathbf{U}_c$, π_c). The abstract interpretation is **Prop**=(*Prop*, $\bigsqcup_p$, $\mathbf{U}_p$, π_p). *Rsub*, $\bigsqcup_c$, *Prop*, and $\bigsqcup_p$ are described in Section 4.1. That section also contains the definition of the concretization and abstraction functions relating *Rsub* and *Prop*. The operations of concrete and abstract unification ($\mathbf{U}_c$ and $\mathbf{U}_p$) and projection (π_c and π_p) are defined in Section 4.2.

4.1. The Domains Rsub and Prop

Recall that $FP(\mathbf{V})$ denotes the set of finite subsets of variables of $\mathbf{V}$.

$$Rsub = \{\top_c, \bot_c\} \cup (\wp(Subst) \times FP(\mathbf{V}))$$

Rsub stands for restricted substitutions. The partial order of *Rsub* is as follows: $\top_c$ is the largest element; $\bot_c$ is the smallest. For any other two elements $[\Sigma_1, U_1]$ and $[\Sigma_2, U_2]$ of *Rsub*, $[\Sigma_1, U_1] \leq_c [\Sigma_2, U_2]$ if and only if $U_1 = U_2$ and $\Sigma_1 \subseteq \Sigma_2$. This ordering yields $\bigsqcup_c$ as follows: for any $c \in Rsub$, $\top_c \bigsqcup_c c = \top_c$, $\bot_c \bigsqcup_c c = c$, for the other elements,

$$[\Sigma_1, U_1] \bigsqcup\nolimits_c [\Sigma_2, U_2] = \begin{cases} [\Sigma_1 \cup \Sigma_2, U_1] & \text{if } U_1 = U_2 \\ \top_c & \text{otherwise} \end{cases}$$

The greatest lower bound (glb) of *Rsub* is analogous. *Rsub* is a complete lattice.

Recall from Section 2.2 that we write simply f for the class of formulas equivalent to f.

$$Prop = \{\top_p, \bot_p\} \cup \{\, [f, U] : U \in FP(\mathbf{V}) \;\&\; f \in \Omega_U(\wedge, \leftrightarrow)/_{\equiv} \cup \{[\mathsf{F}]_{\equiv}\}\,\}$$

Prop is partially ordered as follows: $\top_p$ is the largest element and $\bot_p$ is the smallest; for the other elements, $[f_1, U_1] \leq_p [f_2, U_2]$ if and only if $U_1{=}U_2$ and $f_1 \models f_2$. This ordering yields $\bigsqcup_p$ as follows: for all $d \in Prop$, $\top_p \bigsqcup_p d = \top_p$ and $\bot_p \bigsqcup_p d = d$; for the other elements,

$$[f_1, U_1] \bigsqcup\nolimits_p [f_2, U_2] = \begin{cases} [f_1 \vee f_2, U_1] & \text{if } U_1 = U_2 \\ \top_p & \text{otherwise} \end{cases}$$

The glb of *Prop* is found in a similar way. The fact that *Prop* is a complete lattice follows from the fact that, over a finite set of variables, there is only a finite number of equivalence classes of propositional formulas. Were we not to restrict attention to finite sets of variables, we would have to include infinite formulas to obtain a complete lattice.

The relation between *Rsub* and *Prop* is expressed by two functions: the concretization function, γ_p, and the abstraction function, α_p. γ_p is based on the function γ defined in Section 2.4.

$$\gamma_p : Prop \longrightarrow Rsub$$

$$\gamma_p(d) = \begin{cases} \text{if } d = \top_p, & \top_c \\ \text{if } d = \bot_p, & \bot_c \\ \text{if } d = [f, U], & [\Sigma,\ U], \text{where} \\ & \Sigma = \gamma(f) = \{\sigma \in Subst : \forall \sigma' \trianglelefteq \sigma \,.\, assign\, \sigma' \models f\} \end{cases}$$

The function $\alpha_p : Rsub \longrightarrow Prop$ is the usual adjoint [5] of γ_p, i.e., $\alpha_p(c) = \sqcap_p\{d \in Prop \mid \gamma_p(d) \geq_c c\}$. $(Prop, \gamma_p, Rsub, \alpha_p)$ is a Galois Insertion (see Section 5.1).

4.2. Concrete and Abstract Unification and Projection

We start by describing the projection operations, which are illustrated in Example 1.1 (cf. Section 1). Both preserve $\bot$ and $\top$. The concrete projection π_c is as follows:

$$\pi_c : Rsub \times FP(\mathbf{V}) \longrightarrow Rsub$$
$$([\Sigma, U_1], U_2]) \longmapsto [\Sigma, U_1 \cap U_2]$$

The abstract projection π_p amounts to existentially quantifying a formula [27, 29]. The existential quantification of a propositional formula obeys

$$\exists x . f \equiv f\langle x/\mathsf{T}\rangle \vee f\langle x/\mathsf{F}\rangle$$

(This truth substitution notation $\langle x/\mathsf{T}\rangle$ is introduced in Section 2.2, above.) This generalizes straightforwardly to quantification over sets of variables. It can be seen as follows that if $f \in Pos$, then $\exists U.f$ is also in Pos [28]. Take the truth substitution s' on U that replaces by T all variables of U. It is true that $u \models s'(f)$. Thus $u \models \bigvee\{s(f) \mid s \in U \longrightarrow \{\mathsf{T}, \mathsf{F}\}\}$. Applying Corollary 2.1, it follows that there exists $f' \in \Omega\{\wedge, \leftrightarrow\}$ equivalent to $\exists U.f$ and containing the same free variables. This fact ensures that the following definition of π_p is well defined — projection yields an element of *Prop*.

$$\begin{array}{l} \pi_p : Prop \times FP(\mathbf{V}) \longrightarrow Prop \\ ([f_1, U_1], U_2) \;\longmapsto [\exists U_1 - U_2.\, f_1, U_1 \,\cap\, U_2] \end{array}$$

The following example shows the use of π_p.

Example 4.1. Let $f = ((x_1 \vee x_2) \leftrightarrow (x_3 \wedge x_4)) \wedge x_5$.

$$\begin{array}{l} \pi([f, \{x_1, x_2, x_3, x_4, x_5\}],\ \{x_2, x_3, x_5\}) \\ \quad \equiv [F, \{x_2, x_3, x_5\}], \text{where} \end{array}$$

$$\begin{array}{rl} F = & \exists x_1\, x_4.(((x_1 \vee x_2) \leftrightarrow (x_3 \wedge x_4)) \wedge x_5) \\ \equiv & (((\mathsf{T} \vee x_2) \leftrightarrow (x_3 \wedge \mathsf{T})) \wedge x_5) \vee \\ & (((\mathsf{T} \vee x_2) \leftrightarrow (x_3 \wedge \mathsf{F})) \wedge x_5) \vee \\ & (((\mathsf{F} \vee x_2) \leftrightarrow (x_3 \wedge \mathsf{T})) \wedge x_5) \vee \\ & (((\mathsf{F} \vee x_2) \leftrightarrow (x_3 \wedge \mathsf{F})) \wedge x_5) \\ \equiv & (x_3 \wedge x_5) \vee \mathsf{F} \vee ((x_2 \leftrightarrow x_3) \wedge x_5) \vee ((x_2 \leftrightarrow \mathsf{F}) \wedge x_5) \\ \equiv & (x_3 \wedge x_5) \vee ((x_2 \leftrightarrow x_3) \wedge x_5) \vee ((\neg x_2) \wedge x_5) \\ \equiv & (x_3 \wedge x_5) \vee ((x_2 \leftrightarrow x_3) \wedge x_5) \vee ((x_2 \leftrightarrow (x_2 \wedge x_3 \wedge x_5) \wedge x_5) \\ & \text{by noting that the formula is in } Pos \text{ and} \\ & \text{using the technique used in the proof of Theorem 2.1} \\ \equiv & (x_3 \wedge x_5) \vee ((x_2 \leftrightarrow x_3) \wedge x_5). \end{array}$$

Let us now turn to the unification operations. As different frameworks require different combinations of unification and composition, we provide operations that subsume those of most frameworks. In order to define the concrete unification $\mathbf{U}_c$, it is convenient to introduce first the following function $\mathbf{u}_c$:

$$\begin{array}{l} \mathbf{u}_c : Subst \times Subst \times Subst \longrightarrow Subst \\ \qquad\qquad (\sigma_1, \sigma_2, \delta) \longmapsto mgu(Eq(\sigma_1) \cup Eq(\sigma_2) \cup Eq(\delta)) \end{array}$$

$\mathbf{U}_c$ and $\mathbf{U}_p$ are strict: if either of the first two arguments is $\perp$, the result is $\perp$. Otherwise, if one of these is $\top$, the result is $\top$. The other cases are as follows:

$$\begin{array}{l} \mathbf{U}_c : Rsub \times Rsub \times Subst \longrightarrow Rsub \\ \qquad ([\Sigma_1, U_1],\ [\Sigma_2, U_2], \delta) \longmapsto [\{\mathbf{u}_c(\sigma_1, \sigma_2, \delta) | \sigma_1 \in \Sigma_1 \ \&\ \sigma_2 \in \Sigma_2\}, \\ \qquad\qquad\qquad\qquad\qquad\qquad U_1 \cup U_2 \cup Var(\delta)] \end{array}$$

The abstract unification $\mathbf{U}_p$ here is a modest generalization of the abstract unification operation found in [26, 29].

$$\begin{array}{l} \mathbf{U}_p : Prop \times Prop \times Subst \longrightarrow Prop \\ \qquad ([f_1, U_1], [f_2, U_2], \delta) \longmapsto [f_1 \wedge f_2 \wedge g,\ U_1 \cup U_2 \cup Var(\delta)] \\ \qquad\qquad\qquad\qquad\qquad\qquad where\ [g,\ Var(\delta)] = \alpha_p([\{\delta\},\ Var(\delta)]). \end{array}$$

We have chosen not to incorporate renaming into $\mathbf{U}_c$ and $\mathbf{U}_p$ in order to keep the operations as simple as possible. These technical details can be found in [13].

The reader may wonder why $\mathbf{U}_p$ takes two abstract values as arguments (together with a substitution), whereas in the corresponding operation $unify_{gro}$ of [29] only one such argument is present. When considering the abstract unification of the domain *Prop* there is no difference between these two types of operations. However this is due to the fact that *Prop* has the property of being *condensing* , cf. [28].

Intuitively, a domain is condensing when the knowledge about the initial activation is not important for the final result, i.e., no information is lost if one runs the analysis with "empty" initial activation and then combines the obtained result with the activation only afterwards. When dealing with non-condensing abstract domains, an abstract unification can be more precise by taking two arguments than by taking only one. A simple instance of this phenomenon is illustrated by the following example.

Example 4.2.
Consider the simple domain A for groundness analysis consisting of $\wp(U)$, for $U \in FP(V)$. $B \in A$ approximates the substitutions that instantiate (at least) all variables in B to ground terms. Consider now the following unification step : the calling atom is $T = p(x_1, [x_0|x_1], z)$, and the abstract activation is $a_1 = \{x_0\}$. The clause involved is $l = p(w_1, w_2, w_2) :- Body$.

Obviously, the forward unification computes $\emptyset$ for the variables of l. Assume that, after having processed *Body*, the abstract value is $a_2 = \{w_1\}$. If we perform the backward unification using only a_2 and the mgu of T and the head of l, we compute $\{x_1\}$ which, combined with a_1, gives $\{x_0, x_1\}$. On the other hand, if a_1 is available during the backward unification step, then one could infer that also w_2 is ground and thus also z is ground. Thus, the final result would be: $\{x_0, x_1, z\}$.

This difference would not occur if *Prop* was used. In fact, with *Prop* in both cases we would obtain $x_0 \wedge x_1 \wedge (x_0 \wedge x_1 \leftrightarrow z)$, which implies that z is ground.

When condensing domains are used, our approach is equivalent to that of [29]. However, to the best of our knowledge, *Prop* is the only abstract domain known to be condensing, whereas it is known that *Sharing* [21], and *Def* [1], are not. Therefore, we have chosen, for our abstract unification, the type that encompasses an optimal unification for any abstract domain and not only for condensing ones.

Example 4.3, which is a continuation of Example 1.1, illustrates the use of $\mathbf{U}_c$ and of $\mathbf{U}_p$ both for "forward" and "backward" unifications.

Example 4.3. Suppose we modify the program fragment shown in Example 1.1 so that the definition of q is replaced by

$$q(z) :\text{-}^{\langle 5 \rangle} z = c^{\langle 6 \rangle}.$$

Recall that W=$\{u, v, w, x\}$ and that the activation set and the abstract activation at program point $\langle 1 \rangle$ are, respectively:

$$c_1 = [\{\{u \mapsto v\}, \{u \mapsto w\}\}, W]$$
$$d_1 = [(u \leftrightarrow v) \vee (u \leftrightarrow w), W].$$

The unification q(u)=q(z) is simulated at the concrete level by

$$c_2 = \mathbf{U}_c(c_1, [\{id\}, \{z\}], \{u \mapsto z\}) = [\{\{u \mapsto v, z \mapsto v\}, \{u \mapsto w, z \mapsto w\}\}, W \cup \{z\}]$$

and at the abstract level by

$$d_2 = \mathbf{U}_p(d_1, [\mathsf{T}, \{z\}], \{u \mapsto z\}) = [((u \leftrightarrow v) \vee (u \leftrightarrow w)) \wedge (u \leftrightarrow z), W \cup \{z\}].$$

These are forward unification steps. Observe that in such a step the second argument is a sort of identity value whose principal role is to introduce the variables of the new clause as variables of interest.

The values c_2 and d_2 are projected onto the variables of the called clause, i.e., $\{z\}$, in order to obtain the activations at program point $\langle 5\rangle$. As in Example 1.1, we exhibit the bindings of the variables of interest only.

$$\begin{aligned} c_3 &= \pi_c(c_2, \{z\}) = [\{\{z \mapsto v\}, \{z \mapsto w\}\}, \{z\}] \\ d_3 &= \pi_p(d_2, \{z\}) = [\exists\{u, v, w\}.((u \leftrightarrow v) \vee (u \leftrightarrow w)) \wedge (u \leftrightarrow z), \{z\}] \;\equiv\; [\top\ , \{z\}]. \end{aligned}$$

For program point $\langle 6\rangle$ we obtain:

$$\begin{aligned} c_4 &= \mathbf{U}_c(c_3, [\{id\}, \{z\}], \{z \mapsto c\}) = [\{\{z \mapsto c\}, \{z\}] \\ d_4 &= \mathbf{U}_p(d_3, [\top, \{z\}], \{z \mapsto c\}) \quad\ = [z, \{z\}]. \end{aligned}$$

We now compute the activations at point $\langle 2\rangle$. They are obtained by a backward unification that combines the activation at point $\langle 1\rangle$ and the equation $\{q(u) = q(z)\}$, together with the activation at point $\langle 6\rangle$, as follows:

$$\begin{aligned} \mathbf{U}_c(c_1, c_4, \{u \mapsto z\}) &= [\{\{u \mapsto c, z \mapsto c, v \mapsto c\}, \\ &\qquad\quad \{u \mapsto c, z \mapsto c, w \mapsto c\}\}, W \cup \{z\}] \\ \mathbf{U}_p(d_1, d_4, \{u \mapsto z\}) &= [u \wedge z \wedge (v \vee w), W \cup \{z\}] \end{aligned}$$

When these values are projected onto W, the variables in the calling clause, we obtain the activations associated with program point $\langle 2\rangle$ in Example 1.1.

Our backward unification accomplishes two tasks that in other frameworks (e.g. [29]) are performed by separate operations.

5. CORRECTNESS AND OPTIMALITY

This section is divided into three parts. Section 5.1 shows that $(Prop, \gamma_p, Rsub, \alpha_p)$ is a Galois insertion. Sections 5.2 characterizes in several ways the relationship between an element of *Rsub* and its abstraction in *Prop*. These results are used in Section 5.3 where the correctness and optimality of $\mathbf{U}_p$ and π_p are shown.

5.1. Galois Insertion

First we show that there is a Galois connection between *Prop* and *Rsub*. The desired Galois insertion will then follow from the injectivity of γ_p.

Theorem 5.1 (Galois Connection). $(Prop, \gamma_p, Rsub, \alpha_p)$ *is a Galois connection.*

Proof. By [5] (Proposition 7), it is sufficient to prove that γ is a complete meet-morphism. By Corollary 2.1 and Proposition 2.7, γ is a meet-morphism. Its completeness derives from the observation that the meet of any infinite subset of **Prop** is $\perp$. □

Theorem 5.2. The concretization function $\gamma_p : Prop \longrightarrow Rsub$ *is injective.*

Proof. The result follows easily from Propositions 3.1 and 3.2. □

Injectivity of the concretization function is desirable because otherwise the abstract domain contains different elements that represent the same concrete element.

Corollary 5.1. $(Prop, \gamma_p, Rsub, \alpha_p)$ *is a Galois insertion.*

That the least upper bound operation of **Prop** is both safe and optimal follows from the preceding Corollary. In fact, the following general result holds, [5].

Theorem 5.3. Let (D, γ, C, α) *be a Galois insertion and let* $d_1, d_2 \in D$. *Then* $\sqcup_D$ *is optimal (equivalently,* α *preserves lubs) that is:*

$$\alpha(\gamma(d_1) \sqcup_C \gamma(d_2)) = d_1 \sqcup_D d_2.$$

In Section 6 it will be proven that $\bigsqcup_p$ is also α-optimal, but not γ-optimal.

5.2. *Characterization of the Abstraction of Rsub*

In this section we study the relationship between an element $[\Sigma, U]$ of *Rsub* and its abstraction $\alpha_p([\Sigma, U])$. First we show that the case in which Σ contains several substitutions can be reduced to that of a single substitution, and then we concentrate on the latter case. The results of this section are needed for showing the correctness and optimality of $\mathbf{U}_p$ and π_p, but they are also interesting on their own. It follows easily from Lemma 5.1 that α_p is a complete join-morphism.

Lemma 5.1. If $[\Sigma, U] \in Rsub$, *then*

$$\alpha_p([\Sigma, U]) = \bigsqcup_p \{\alpha_p([\{\sigma\}, U]) \mid \sigma \in \Sigma\}$$

Proof. Let $F_0 = \{f_\sigma \mid [f_\sigma, U] = \alpha_p([\{\sigma\}, U]),\ \sigma \in \Sigma\}$. By properties of $\bigsqcup_p$,

$$\bigsqcup_p \{\alpha_p([\{\sigma\}, U]) \mid \sigma \in \Sigma\} = [\bigvee F_0, U]$$

By definition of α_p, $\alpha_p([\Sigma, U]) = [\bigwedge F_1, U]$, where $F_1 = \{f \mid Var(f) \subseteq U,\ \Sigma \subseteq \gamma(f)\}$. We show that $\bigvee F_0 \equiv \bigwedge F_1$.

($\Rightarrow$) We show that for any $f_\sigma \in F_0$ and $f' \in F_1$, $f_\sigma \models f'$. Consider any $\sigma \in \Sigma$ and let f_σ satisfy $\alpha_p([\{\sigma\}, U]) = [f_\sigma, U]$. Also let $f' \in F_1$. Since $\{\sigma\} \subseteq \Sigma \subseteq \gamma(f')$, by monotonicity of α_p, it follows that,

$$\alpha_p([\{\sigma\}, U]) \leq_p \alpha_p(\gamma_p([f', U]))$$

Since $\alpha_p \circ \gamma_p$ is the identity, from this it follows that:

$$\alpha_p([\{\sigma\}, U]) = [f_\sigma, U] \leq_p [f', U]$$

This implies that $f_\sigma \models f'$.

($\Leftarrow$) We show that $[\Sigma, U] \leq_c \gamma_p([\bigvee F_0, U])$, which implies that $\bigvee F_0 \in F_1$.

$$\begin{aligned}
\gamma_p([\bigvee F_0, U]) &= \gamma_p([\bigvee \{f_\sigma \mid [f_\sigma, U] = \alpha_p([\{\sigma\}, U]) \ \&\ \sigma \in \Sigma\}, U]) \\
&\quad \text{by definition of } F_0, \\
&\geq_c \textstyle\bigsqcup_c \{\gamma_p([f_\sigma, U]) \mid [f_\sigma, U] = \alpha_p([\{\sigma\}, U]) \ \&\ \sigma \in \Sigma\} \\
&\quad \text{by monotonicity of } \gamma_p \\
&= \textstyle\bigsqcup_c \{\gamma_p(\alpha_p([\{\sigma\}, U])) \mid \sigma \in \Sigma\} \\
&\quad \text{by definition of } [f_\sigma, U] \\
&\geq_c [\Sigma, U] \\
&\quad \text{because } \gamma_p \circ \alpha_p([\{\sigma\}, U]) \geq_c [\{\sigma\}, U]
\end{aligned}$$

$\square$

Let us then consider the abstraction of a single substitution. Some new terminology is necessary. Let σ be an idempotent substitution and $U \in FP(V)$.

- $Ax(\sigma) = \{assign\,\sigma' \mid \sigma' \trianglelefteq \sigma\}$. For $\Sigma \subseteq Subst, Ax(\Sigma) = \bigcup\{Ax(\sigma) \mid \sigma \in \Sigma\}$
- $A^U_\sigma = \{f \mid f \in Pos,\, Var(f) \subseteq U,\, \forall \sigma' \trianglelefteq \sigma, assign\,\sigma' \models f\}$.
- $A_\sigma = \{f \mid f \in Pos,\, \forall \sigma' \trianglelefteq \sigma, assign\,\sigma' \models f\}$.
- $F_\sigma = \bigwedge \{x \leftrightarrow \wedge Var(\sigma(x)) | \, x \in supp(\sigma)\}$.

For any formula f, $Models(f)$ is the set of truth-assignments that model f. If r is a truth-assignments, $r\mid_U$ is its restriction to U, i.e., $r\big|_U : U \to \{true, false\}$. Restriction to a finite set of variables will be applied to a set of truth-assignments with the obvious meaning. If $r\mid_U$ is a restricted truth-assignment, then $F(r\mid_U) = \bigwedge\{t_x \mid x \in U\}$, where $t_x = x$ if $r \models x$, and $t_x = \neg x$, otherwise. For a restricted set of truth-assignments $T\mid_U$, $F(T\mid_U) = \bigvee\{F(r\mid_U) \mid r \in T\}$. Observe that if $U \supseteq Var(f)$, then $F(Models(f)\mid_U) \equiv f$.
We first relate $Ax(\sigma)$ and F_σ.

Lemma 5.2. Let $U = Var(\sigma)$, $Ax(\sigma)\mid_U = Models(F_\sigma)\mid_U$.

PROOF.

($\supseteq$) Consider any $r \in Model(F_\sigma)$. From r we want to construct $\hat{\sigma} \trianglelefteq \sigma$ such that $assign\,\hat{\sigma}\mid_U = r\mid_U$. $\hat{\sigma} = \sigma' \circ \sigma$, where σ' is as follows: for the variables not in U, σ' is fixed arbitrarily, whereas, for each $x \in U, \sigma'$ is:

$$\sigma'(x) = \begin{cases} a \text{ if } r \models x \\ x \text{ otherwise} \end{cases}$$

We want to show that $assign\,\hat{\sigma}\mid_U = r\mid_U$. For each $x \in U - supp(\sigma)$, this is immediate from the definition of σ' and the idempotency of σ. For each $x \in supp(\sigma)$, the following holds:

$$\begin{aligned} assign\,\hat{\sigma} \models x &\Leftrightarrow assign\,\sigma' \models \wedge Var(\sigma(x)) \\ &\Leftrightarrow r \models \wedge Var(\sigma(x)) \\ &\Leftrightarrow r \models x \qquad\qquad \text{since } r \models x \leftrightarrow \wedge Var(\sigma(x)) \end{aligned}$$

($\subseteq$) Consider any instance $\hat{\sigma}$ of σ. We want to show that $assign\,\hat{\sigma} \models F_\sigma$. Let $\hat{\sigma} = \sigma' \circ \sigma$. If $assign\,\hat{\sigma} \not\models F_\sigma$, then there must be a formula, $x \leftrightarrow \wedge Var(\sigma(x))$ in F_σ such that $assign\,\hat{\sigma} \not\models x \leftrightarrow \wedge Var(\sigma(x))$. Let us assume that $assign\,\hat{\sigma} \models x$, but $assign\,\hat{\sigma} \not\models \wedge Var(\sigma(x))$. The other case is similar. If $assign\,\hat{\sigma} \models x$, then $\forall y \in Var(\sigma(x)), \sigma'(y)$ is a ground term. But, by the idempotency of σ, $\hat{\sigma}(y) = \sigma'(y)$ for all such variables. Thus the initial assumption is contradicted. $\square$

It is important to understand what happens to the models of a formula when some of its variables are existentially quantified.

Lemma 5.3. Let f be any formula, $U \in FP(V)$ and $W = Var(f)$.
$Models(\exists W - U.\ f)\mid_U = Models(f)\mid_U$

PROOF. The $\supseteq$-direction is easy: just observe that any model of f is also a model of the quantified formula.

For the other direction, consider $r \in Models(\exists W - U.\ f)$, there is a truth-substitution c on $W - U$ such that, $r \models c(f)$. It suffices now to combine r and c into a truth-assignment r' as follows: on $W - U$ r' agrees with c, on U it coincides with r, and on the remaining variables is fixed arbitrarily. It is easy to see that $r' \in Models(f)$. □

The following simple result is important for the sequel.

Lemma 5.4. Consider any $s \in Ax(\sigma)$. Let us construct s' from s as follows: fix any finite set W of variables such that $W \cap Var(\sigma) = \emptyset$, for the variables outside of W, s' coincides with s, whereas for those in W it takes arbitrary values. Then $s' \in Ax(\sigma)$.

Proof. It suffices to observe that variables as those in W are unconstrained by σ and thus there are instances of σ that instantiate them to ground/non-ground values in all possible ways. □

Using the previous result we can characterize the formula $\wedge A_\sigma^U$. The intuition is that in $\wedge A_\sigma^U$ only the variables in $Var(\sigma) \cap U$ are useful, the remaining variables of U are like those in W of lemma 5.4.

Lemma 5.5. Let $U \in FP(V)$, and $W = Var(\sigma) \cap U$, then $\wedge A_\sigma^U \equiv F(Ax(\sigma) \mid_W)$.

Proof. Let $F_0 = F(Ax(\sigma) \mid_W)$. First observe that F_0 belongs to A_σ^U. In fact it is a positive formula containing only variables in U and obviously, $\forall \sigma' \trianglelefteq \sigma$, *assign* $\sigma' \models F_0$. Thus $\wedge A_\sigma^U \models F_0$.

On the other hand, $F_0 \models \wedge A_\sigma^U$ because every model r of F_0 is such that $r|_W =$ *assign* $\sigma'|_W$, for some $\sigma' \trianglelefteq \sigma$. This fact, together with Lemma 5.4, proves that $r|_U \in Ax(\sigma)|_U$ and thus, $r \models \wedge A_\sigma^U$. □

The previous result has several important consequences:

Theorem 5.4. For $\sigma \in Subst$, the following points are true:

1. $\wedge A_\sigma \equiv F(Ax(\sigma) \mid_{Var(\sigma)})$;
2. $\wedge A_\sigma \equiv \wedge F_\sigma$;
3. $\wedge A_\sigma^U \equiv \exists\, Var(\sigma) - U.\ \wedge A_\sigma$.

Proof. Point (1) is true because, $A_\sigma = \bigcup\{A_\sigma^U\ :\ U \in FP(V)\}$. By Lemma 5.5, this implies that: $\wedge A_\sigma = \bigwedge\{F(Ax(\sigma) \mid_{Var(\sigma)\cap U})\ : U \in FP(V)\}$
which is equivalent to $F(Ax(\sigma) \mid_{Var(\sigma)})$. Point (2) follows directly from Lemma 5.2 and point (1) above. Point (3) follows from point (1), Lemma 5.5 and 5.3. □

The next Theorem puts together all results shown so far and states several interesting characterizations of $\alpha_p[\{\sigma\}, U]$.

Theorem 5.5. Let $\alpha_p([\{\sigma\}, U]) = [f, U]$. The following points hold:

1. $f \equiv \exists\, Var(\sigma) - U. \wedge A_\sigma$;
2. $f \equiv \exists\, Var(\sigma) - U. F_\sigma$;
3. *for* $\hat{f} \in Pos, f \equiv \hat{f}$ *iff* $Ax(\sigma) \mid_U = Models(\hat{f}) \mid_U$.

Proof. Point (1) and (2) follows immediately from Theorem 5.4 (3) and (2), respectively. Let us consider point(3).

($\Rightarrow$) Lemma 5.5 shows that this result holds for the variables in $Var(\sigma) \cap U$, in fact, it states that: $\hat{f} \equiv F(Ax(\sigma) \mid_{Var(\sigma) \cap U})$. Using Lemma 5.4, it is easy to see that the other variables in U are irrelevant.
($\Leftarrow$) By the hypothesis, $\hat{f} \equiv F(Ax(\sigma) \mid_U)$. Using the fact that the variables in $U - Var(\sigma)$ are unconstrained by σ and thus take any value in $Ax(\sigma)$, one can prove that: $\hat{f} \equiv F(Ax(\sigma) \mid_{Var(\sigma) \cap U})$. Thus, by Lemma 5.5, $\hat{f} \equiv \wedge A_\sigma^U = \alpha_p([\{\sigma\}, U])$. □

Using the previous Theorem and Lemma 5.1, it is easy to characterize the abstraction of elements of *Rsub* that contain more than one substitution.

Corollary 5.2. Let $[\Sigma, U] \in Rsub$. *If* $\alpha_p([\Sigma, U]) = [f, U]$, *then* $Models(f) \mid_U = Ax(\Sigma) \mid_U$.

The following result is immediate from Corollary 5.2.

Lemma 5.6. For any $\Sigma \subseteq Subst$ *and* $U_1, U_2 \in FP(\mathbf{V})$ *satisfying* $U_1 \subseteq U_2$, *if* $\alpha_p([\Sigma, U_1]) = [f_1, U_1]$ *and* $\alpha_p([\Sigma, U_2]) = [f_2, U_2]$, *then* $f_2 \models f_1$.

We conclude this section with a technical Lemma that is useful in the sequel.

Lemma 5.7. Let E *be a solvable set of equations and let* $U \in FP(\mathbf{V})$. *Assume that* $\alpha_p(\{[mgu(E), U]\}) = [f, U]$. *Then,*

$$f \models \exists Var(E) - U. \bigwedge \{\wedge Var(t_0) \leftrightarrow \wedge Var(t_1) \mid (t_0 = t_1) \in E\}.$$

Proof. Let $\sigma = mgu(E)$ and let $\alpha_p([\{\sigma\}, U]) = [f_\sigma, U]$. We show that for all $(t_0 = t_1) \in E$,

$$f_\sigma \models \exists Var(E) - U. \wedge Var(t_0) \leftrightarrow \wedge Var(t_1).$$

By Theorem 5.5, $f_\sigma \models \exists\ Var(E) - U. \bigwedge \{x \leftrightarrow \wedge Var(\sigma(x)) \mid x \in supp(\sigma)\}$. Thus, for any term t,

$$f_\sigma \models \exists Var(E) - U. \wedge Var(t) \leftrightarrow \wedge Var(\sigma(t)).$$

Applying this fact to t_0 and t_1 and using $\sigma(t_0) = \sigma(t_1)$ yields the desired result. □

5.3. *Correctness and Optimality of* $\mathbf{U}_p$ *and* π_p

We are now ready to prove correctness and optimality of the operations of **Prop**.

Theorem 5.6 (Correctness of $\mathbf{U}_p$*). For all elements* d_1 *and* d_2 *of Prop and* $\delta \in$ *Subst,*

$$\alpha_p(\mathbf{U}_c(\gamma_p(d_1), \gamma_p(d_2), \delta) \leq_p \mathbf{U}_p(d_1, d_2, \delta).$$

Proof. The result follows easily if either d_1 or d_2 is either $\perp$ or $\top$. Assume the following notation: $d_1 = [f_1, U_1]$ and $d_2 = [f_2, U_2]$, $W = U_1 \cup U_2 \cup Var(\delta)$ and $\gamma_p(d_1) = [\Sigma_1, U_1]$ and $\gamma_p(d_2) = [\Sigma_2, U_2]$. If either f_1 or f_2 is F, the result follows easily. Assume this is not the case and hence that neither Σ_1 nor Σ_2 is empty.

$$
\begin{aligned}
&\alpha_p(\mathbf{U}_c(\gamma_p(d_1), \gamma_p(d_2), \delta)) \\
&= \bigsqcup_p \{\alpha_p([\{\mathbf{u}_c(\sigma_1, \sigma_2, \delta)\}, W]) \mid \sigma_1 \in \Sigma_1 \ \&\ \sigma_2 \in \Sigma_2\} \\
&\quad \text{by Lemma 5.1 and the definition of } \mathbf{U}_c \\
&= \bigsqcup_p \{\alpha_p([\{mgu(Eq(\sigma_1) \cup Eq(\sigma_2) \cup Eq(\delta))\}, W]) \mid \sigma_1 \in \Sigma_1 \ \&\ \sigma_2 \in \Sigma_2\} \\
&\quad \text{by definition of } \mathbf{u}_c \\
&= [\bigvee F_0,\ W], \text{by definition of } \bigsqcup_p \text{, where} \\
&\quad F_0 = \{f \mid [f, W] = \alpha_p([\{mgu(Eq(\sigma_1) \cup Eq(\sigma_2) \cup Eq(\delta))\}, W]) \ \& \\
&\qquad\qquad \sigma_1 \in \Sigma_1 \ \&\ \sigma_2 \in \Sigma_2\}.
\end{aligned}
$$

By definition of $\mathbf{U}_p$, $\mathbf{U}_p(d_1, d_2, \delta) = [f_1 \wedge f_2 \wedge g,\ W]$, where $[g,\ Var(\delta)] = \alpha_p([\{\delta\},\ Var(\delta)]$.

In order to prove the theorem, it suffices to show that for any truth assignment r such that $r \models \bigvee F_0$, it is true that $r \models f_1 \wedge f_2 \wedge g$. Assuming $r \models \bigvee F_0$, there must be $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$ such that $[f, W] = \alpha_p([mgu(Eq(\sigma_1) \cup Eq(\sigma_2) \cup Eq(\delta)), W])$ and $r \models f$. By Lemma 5.7, $f \models \exists \widehat{W}. \bigwedge\{x \leftrightarrow \wedge Var(t) \mid x = t \in Eq(\sigma_1) \cup Eq(\sigma_2) \cup Eq(\delta)\}$, where $\widehat{W} = [Var(\sigma_1) \cup Var(\sigma_2) \cup Var(\delta)] - W$.

Therefore, by Theorem 5.5 (2), $r \models f_{\sigma_1} \wedge f_{\sigma_2} \wedge g'$, where:

$$
\begin{aligned}
[f_{\sigma_1}, W] &= \alpha_p([\{\sigma_1\}, W]) \\
[f_{\sigma_2}, W] &= \alpha_p([\{\sigma_2\},\ W]) \\
[g', W] &= \alpha_p([\{\delta\},\ W])
\end{aligned}
$$

Since $\{\sigma_1\} \subseteq \Sigma_1$, by monotonicity of α_p, it is true that

$$[f_{\sigma_1}, W] = \alpha_p([\{\sigma_1\}, W]) \leq_p \alpha_p([\Sigma_1, W])$$

Let $\alpha_p([\Sigma_1, W]) = [f',\ W]$. Then, Lemma 5.6 implies that $f' \models f_1$, because, $[f_1,\ U_1] = \alpha_p(\gamma_p([f_1,\ U_1])) = \alpha_p([\Sigma_1, U_1])$ and $U_1 \subseteq W$. Hence, $f_{\sigma_1} \models f_1$. Similarly, one shows that $f_{\sigma_2} \models f_2$ and that $g' \models g$. Thus, since $r \models f_{\sigma_1} \wedge f_{\sigma_2} \wedge g'$, it must be that $r \models f_1 \wedge f_2 \wedge g$, as desired. □

Theorem 5.7 (Optimality of $\mathbf{U}_p$*). For all* d_1 *and* d_2 *of Prop and* $\delta \in$ *Subst,*

$$\alpha_p(\mathbf{U_c}(\gamma_p(d_1), \gamma_p(d_2), \delta)) \equiv \mathbf{U_p}(d_1, d_2, \delta)$$

PROOF. One direction of the logical equivalence, namely $\leq_p$, is shown in Theorem 5.6. We use in this proof the notation introduced in that of Theorem 5.6. Again, the result follows easily if either d_1 or d_2 is either $\perp$ or $\top$, and also if either f_1 or f_2 is F.

We show that for any truth assignment r such that $r \models f_1 \wedge f_2 \wedge g$, $r \models \bigvee F_0$. In the case where $r \models x$ for all $x \in W$, $r \models \bigvee F_0$ follows from the fact that $\bigvee F_0$ is a positive formula in W. In the case where there exists $z \in W$ such that $r \not\models z$, we show that the following holds: there exist substitutions σ, σ_1, and σ_2 such that points (i) and (ii) below are satisfied.

(i) $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$, and
(ii) $mgu(Eq(\sigma_1) \cup Eq(\sigma_2) \cup Eq(\delta)) = \sigma$, thus the unification does not fail, and if $\alpha_p([\{\sigma\},\ W]) = [f_\sigma, W]$, then $r \models f_\sigma$.

By case assumption, there exists a variable in W that is falsified by r. Letting z be such a variable, we define $\sigma = \sigma_1 = \sigma_2 = \sigma' \circ \delta$, where σ' is defined on each

$x \in W$ as follows:

$$\sigma'(x) = \begin{cases} a \text{ if } r \models x \\ z \text{ otherwise} \end{cases}$$

We show that conditions (i) and (ii) hold for σ_1 and σ_2 defined in this way.

- *Proof of (i)*:
  Since f_1 and f_2 have only variables in W, it suffices to consider this set of variables. For each $x \in W$,

  $$\begin{aligned} assign\,\sigma \models x &\Leftrightarrow assign\,\sigma' \models \wedge Var(\delta(x)) \\ &\Leftrightarrow r \models \wedge Var(\delta(x)) \end{aligned}$$

  Since $r \models f_1 \wedge f_2 \wedge g$ and $[g, Var(\delta)] = \alpha_p([\{\delta\}, Var(\delta)])$, we have

  $$r \models x \leftrightarrow \wedge Var(\delta(x))$$

  Thus, for each $x \in W$, $assign\,\sigma \models x \Leftrightarrow r \models x$, from which it follows that $assign\,\sigma \models f_1$. It suffices now to observe that for each $\hat{\sigma} \trianglelefteq \sigma$,

  — either for all $x \in W$, $assign\,\hat{\sigma} \models x \;\Leftrightarrow\; assign\,\sigma \models x$,
  — or for all $x \in W$, $assign\,\hat{\sigma} \models x \;\Leftrightarrow\; u \models x$
  (Recall that u is the unit assignment).

  Obviously, $u \models f_1$. Thus, for all $\hat{\sigma} \trianglelefteq \sigma$ we have shown that $assign\,\hat{\sigma} \models f_1$. This implies that $\sigma_1 \in \Sigma_1$ (recall that $\gamma_p([f_1, U_1]) = [\Sigma_1, U_1]$). By a similar argument, one can show that $\sigma_2 \in \Sigma_2$.
- *Proof of (ii)*:
  From the definition of $\sigma = \sigma_1 = \sigma_2$, it follows that

  $$\begin{aligned} mgu(Eq(\sigma_1) \cup Eq(\sigma_2) \cup Eq(\delta)) &= mgu(Eq(\sigma' \circ \delta) \cup Eq(\delta)) \\ &= mgu(Eq(\sigma' \circ \delta)) \\ &= \sigma \end{aligned}$$

  Thus the unification does not fail. It remains to show that $r \models f_\sigma$, where $[f_\sigma, W] = \alpha_p([\{\sigma\}, W])$. From Theorem 5.5 (2), we know that:

  $$f_\sigma \;=\; \exists\; Var(\sigma) - W.\; \bigwedge \{x \leftrightarrow \wedge Var(\sigma(x)) \;\mid\; x \in supp(\sigma)\}$$

  In order to show that $r \models f_\sigma$, we prove the stronger fact that:
  $\forall x \in supp(\sigma), r \models x \leftrightarrow \wedge Var(\sigma(x))$
  As noted in the proof of condition (i), for each $x \in W$, $assign\,\sigma \models x \Leftrightarrow r \models x$. Also note that for each $x \in W$, when $\sigma(x)$ is not ground then, by definition of σ', $Var(\sigma(x)) = \{z\}$. Thus the following holds:

  $$Var(\sigma(x)) = \begin{cases} \emptyset & \text{if } r \models x \\ \{z\} & \text{otherwise} \end{cases}$$

  In the case that $r \models x$, obviously, $r \models \wedge Var(\sigma(x))$ as the empty conjunction is true. In the case that $r \not\models x$, we also have that, $r \not\models \wedge(\sigma(x)) = z$, because $r \not\models z$ by assumption. Thus it is true that $r \models x \leftrightarrow \wedge Var(\sigma(x))$, as desired.
  $\square$

In a typical semantic construction, renaming is performed before unification, thus the substitutions in the first two arguments of $\mathbf{U}_c$ are variable disjoint. This restriction is not considered in Theorems 5.6 and 5.7. Both theorems continue to hold when renaming is applied. The proof of the first need not be changed, whereas that of the second becomes more complicated, though it still follows the same outline. Since adding renaming adds technical details to the proof and does not significantly alter the proof method, we present the simpler result here. The stronger result can be found in [13].

Let us now show the correctness and optimality of π_p.

Theorem 5.8 (Correctness and Optimality of π_p). For any $d \in Prop$, and $U' \in FP(\mathbf{V})$,

$$\pi_p(d, U') = \alpha_p(\pi_c(\gamma_p(d), U')).$$

Proof. The result follows trivially if d is either $\perp$ or $\top$. Thus assume that $[f, U] = d$ and let f' be defined as follows:

$$\begin{aligned}[f', U \cap U'] &= \alpha_p(\pi_c(\gamma_p([f, U]), U')) \\ &= \alpha_p(\pi_c([\gamma(f), U], U')) \text{ by definition of } \gamma_p \\ &= \alpha_p([\gamma(f), U \cap U']) \text{ by definition of } \pi_c\end{aligned}$$

From this fact, we reason as follows:

$$\begin{aligned}Models(f')\mid_{U\cap U'} &= Ax(\gamma(f))\mid_{U\cap U'} \\ &\quad \text{by theorem 5.5(3)} \\ &= (Ax(\gamma(f))\mid_U)\mid_{U'} \\ &= (Models(f)\mid_U)\mid_{U'} \\ &\quad \text{again by theorem 5.5(3)} \\ &= Models(f)\mid_{U\cap U'} \\ &= Models(\exists U - U'.f)\mid_{U\cap U'} \\ &\quad \text{by Lemma 5.3}\end{aligned}$$

Thus, $f' \equiv \exists U - U'.f$ (recall that both formulas contain only variables in $U \cap U'$). To conclude the proof it suffices now to recall that $\pi_p([f, U], U') = [\exists U - U'.f, U \cap U']$. $\square$

6. STRONG OPTIMALITIES

This section introduces and studies two alternative notions of optimality for abstract operations, called α- and γ-optimality. Let $\mathbf{C}$ and $\mathbf{D}$ be a concrete and an abstract interpretation. Assume that there is a Galois insertion between their domains, C and D, with α and γ abstraction and concretization functions. Let also op_C and op_D be corresponding operations of $\mathbf{C}$ and $\mathbf{D}$ (that are supposed unary for simplicity).

- op_D is *α-optimal* if $\forall c \in C, \alpha(op_C(c)) = op_D(\alpha(c))$.
- op_D is *γ-optimal* if $\forall d \in D, op_C(\gamma(d)) = \gamma(op_D(d))$.

If we look at $\mathbf{D}$ as source programs together with their meaning and at $\mathbf{C}$ as object programs with their meaning, then our notion of γ-optimality resembles quite closely the well-known "diagram-commuting" condition for compiler correctness of [32]. It seems natural to consider the appropriateness of this condition as a standard of precision for abstract interpretations. In the compiler-correctness view, one expects to find an encoding homomorphism between the meaning of source programs and that of corresponding object programs, which allows an operation to be performed in **D** rather than in **C** with not loss of information. In abstract interpretation some loss of information is intrinsic to the method, but it might be that this loss would be confined to certain operations, while other operations might behave exactly as their concrete counterparts do on the corresponding sublattice of the concrete domain. However, it turns out that even for an abstract domain as powerful as **Prop**, γ-optimality is too strong to be met.

On the other hand, the notion of α-optimality is met by some of the operations of **Prop**. Alone among the these operations, $\mathbf{U}_p$ is not α-optimal; $\mathbf{U}_p$ is the only one in which the structure of the terms in the substitutions affects the result of the operation, and this information is lost by **Prop**.

The section begins by comparing these alternative notions of optimality with the customary optimality of [4]. Then we characterize the precision of analyses based on strongly optimal abstract interpretations. The section concludes by applying the alternative criteria to **Prop**.

Both α and γ-optimality imply standard optimality. For instance, consider α-optimality. From it we have

$$\forall d \in D, \alpha(op_C(\gamma(d))) \;=\; op_D(\alpha(\gamma(d))).$$

Since in a Galois insertion, $\alpha(\gamma(d)) = d$, and we obtain

$$\alpha(op_C(\gamma(d))) \;=\; op_D(d)$$

which is standard optimality. The other implication is even simpler.

We say that an abstract interpretation is α- or γ-*optimal*, when each of its operations is, respectively, α- or γ-optimal. Let us consider what properties hold for the analyses induced by abstract interpretations that are either α- or γ-optimal. Some notation is needed. A data flow semantics of a given logic program P defines its denotation $\mathcal{P}[\![P]\!] \in Den = Atom \to X \to X$, where $Atom$ is the set of atoms and X a domain constant that must be interpreted. The meaning of a program is a function that, given an atom (goal) and some information about it (for instance a set of substitutions or a formula), produces the answer. This denotation is defined as the least fixpoint of a continuous function $\bigsqcup_{l \in P} \mathcal{C}[\![l]\!] : Den \to Den$, that collects the results of all the clauses l of P. A complete definition of $\mathcal{C}[\![l]\!]$ can be found in [13]. Recall the abstract interpretations **C** and **D**, introduced above. Interpreting the domain X and the operation symbols of the data flow semantics with those of **C** and **D** respectively, one obtains $\mathcal{P}[\![P]\!]_C$ and $\mathcal{P}[\![P]\!]_D$, the concrete and abstract meaning of P. Consider $\mathcal{P}[\![P]\!]_C$. By the hypothesis of continuity, this function is the lub of the chain of functions $g_i \in Den_C$ with $i \geq 0$, defined as follows:
$g_0(A, c) = \bot_C \; \forall A \in Atom$ and $c \in C$
for $i > 0 \;\; g_i = \bigsqcup_{l \in P} \mathbf{C}[\![l]\!]_C \; g_{i-1}$
$\mathbf{P}[\![P]\!]_D$ is the lub of the chain of functions $s_i \in Den_D$, defined analogously.

Theorem 6.1 (Consequences of α-optimality). If **D** *is α-optimal with respect to* **C**, *then $\forall i \geq 0$ the following holds: $\forall A \in Atom$ and $c \in C$,*

(1) $\alpha(g_i(A, c)) = s_i(A, \alpha(c))$
(2) $\alpha(\mathbf{P}[\![P]\!]_C Ac) = \mathbf{P}[\![P]\!]_D A(\alpha c)$.

PROOF.

(1) For $i = 0$ it suffices to observe that $\alpha(\bot_C) = \bot_D$, since $\alpha(\bot_C) = \bigwedge\{d : d \in D,\ \gamma(d) \geq \bot_C\} = \bot_D$.
For $i > 0$ a structural induction on the equations defining $\mathcal{C}[\![l]\!]$ (see [13]) using the optimality hypothesis, easily gives the result.

(2) Using point(1) and the α-optimality of $\bigsqcup_D$, it is easy to show that:
$\alpha(\mathcal{P}[\![P]\!]_C\ A\ c) = \alpha(\bigsqcup_C\{g_i(A,c) : i \geq 0\}) = \bigsqcup_D\{s_i(A,\alpha(c)) : i \geq 0\} = \mathcal{P}[\![P]\!]_D\ A\ (\alpha c)$.

∎

In the case that **D** is γ-optimal with respect to **C**, a result similar to that of Theorem 6.1 can be shown.

Theorem 6.2 (Consequences of γ-optimality). If **D** *is α-optimal with respect to* **C** *and if $\gamma(\bot_D) = \bot_C$, the following holds: $\forall A \in Atom$ and $d \in D$,
$\mathcal{P}[\![P]\!]_C\ A\ \gamma(d) = \gamma(\mathcal{P}[\![P]\!]_D\ A\ d)$.*

Proof. The proof is similar to that of Theorem 6.1. The extra condition about $\bot_D$ takes care of the case i=0. ∎

From the above theorems it follows that it is quite interesting to have abstract interpretations that are α- or γ-optimal. Thus, we will test below whether **Prop** satisfies one of these properties. Unfortunately, the answer is to the negative. In addition to this, we will show that, even though optimal, **Prop** does not satisfy the conditions of Theorems 6.1 and 6.2.

Lemma 6.1. $\sqcup_p$ is α-optimal.

Proof. We have to show that:

$$\forall c_1, c_2 \in Rsub,\ \alpha_p(c_1 \sqcup_c c_2) \ = \ \alpha_p(c_1) \sqcup_p \alpha_p(c_2)$$

When c_1 or c_2 is $\bot_c$ or $\top_c$, the result is immediate from the definitions of $\sqcup_c$, $\sqcup_p$, and α_p. Consider then that $c_1 = [\Sigma_1,\ U_1]$ and $c_2 = [\Sigma_2,\ U_2]$. If $U_1 \neq U_2$ then again the result is $\top_p$, by definition of $\sqcup_c$ and $\sqcup_p$. Thus assume that $U_1 = U_2 = U$. Also let $\alpha_p([\Sigma_1,\ U]) = [f_1,\ U]$ and $\alpha_p([\Sigma_2,\ U]) = [f_2,\ U]$.

By definition of $\sqcup_c$, it is true that:

$$\alpha_p([\Sigma_1,\ U] \sqcup_c [\Sigma_2,\ U]) \ = \ \alpha_p([\Sigma_1 \cup \Sigma_2,\ U])$$

Let $\alpha_p([\Sigma_1 \cup \Sigma_2,\ U]) = [f,\ U]$.

$$\begin{aligned} Models(f)|_U &= Ax(\Sigma_1 \cup \Sigma_2)|_U \text{ by Theorem 5.5} \\ &= Ax(\Sigma_1)|_U \cup Ax(\Sigma_2)|_U \text{ obvious} \\ &= Models(f_1)|_U \cup Models(f_2)|_U \text{ by Theorem 5.5} \\ &= Models(f_1 \vee f_2)|_U \end{aligned}$$

∎

That $\sqcup_p$ is not γ-optimal is quite surprising. In fact, Corollary 5.2 seems to imply that the models of f determine the substitutions in $\gamma(f)$. Now, $Models(f \vee g) = Models(f) \cup Models(g)$. Thus one would also expect that $\gamma(f \vee g) = \gamma(f) \cup \gamma(g)$. However, this is false as shown in the following lemma.

Lemma 6.2 [19]. $\sqcup_p$ is not γ-optimal.

The reason for this surprising negative result is that taking the union of the models of two formulas f and g, we allow for new substitutions to be approximated, namely those substitutions that have some instances that give truth assignments satisfying f and other instances that give truth assignments that satisfy g.

Let us consider now the projection π_p. As for $\sqcup_p$, π_p is α-optimal, but not γ-optimal.

Lemma 6.3. *π_p is α-optimal*

PROOF. We must show that for any $c \in Rsub$ and $U' \in FP(\mathbf{V})$, it is true that:

$$\pi_p(\alpha_p(c),\ U') \ = \ \alpha_p(\pi_c(c,\ U')).$$

If c is $\bot_c$ or $\top_c$, the result is immediate. Assume then that $c = [\Sigma,\ U]$. Theorem 5.8 shows this result for those elements $[\Sigma,\ U]$ of $Rsub$ that are images of elements of $Prop$, i.e., $[\Sigma,\ U] \ = \ \gamma_p([f,\ U])$. In fact, for such elements the following holds:

$$\begin{aligned}\pi_p(\alpha_p(\gamma_p([f,\ U])),\ U') &= \pi_p([f,\ U],\ U') \\ &\quad \text{since for a Galois insertion } \alpha_p \circ \gamma_p \text{ equals the identity} \\ &= \alpha_p(\pi_c([\Sigma,\ U],\ U')) \\ &\quad \text{by Theorem 5.8}\end{aligned}$$

From the assumption that $[\Sigma,\ U] \ = \ \gamma_p([f,\ U])$, from Corollary 5.2, we have that:

$$Ax(\Sigma)|_U = Models(f)|_U$$

Consider now any element $[\Sigma',\ U] \le_c [\Sigma,\ U]$ and such that $\alpha_p([\Sigma',\ U]) = [f,\ U]$. Again, by Corollary 5.2, we have that, $Ax(\Sigma')|_U = Models(f)|_U = Ax(\Sigma)|_U$. This implies that, if $[f', U \cap U'] = \alpha_p([\Sigma', U \cap U'])$, then

$$\begin{aligned}Models(f')|_{U\cap U'} &= Ax(\Sigma')|_{U\cap U'} \\ &= Models(f)|_{U\cap U'} \\ &= Models(\exists U - U'.f)|_{U\cap U'} \\ &\quad \text{by Lemma 5.3}\end{aligned}$$

Thus, $f' \equiv \exists U - U'.f$. □

Lemma 6.4. *π_p is not γ-optimal.*

PROOF. An easy counter example suffices. Consider $[x,\ \{x\}]$. Obviously, if $\gamma_p([x,\ \{x\}]) = [\Sigma_x, \{x\}]$, then Σ_x contains all the substitutions that ground x. Consider now,

$$\pi_c([x,\ \{x\}],\ \{y\}) = [\mathsf{T},\ \emptyset] \text{ and}$$
$$\pi_p([\Sigma_x,\ \{x\}],\ \{y\}) = [\Sigma_x,\ \emptyset]$$

Obviously, $\gamma_p([\mathsf{T},\ \emptyset]) \ \ge_c \ [\Sigma_x,\ \emptyset]$, because $Subst \supset \ \Sigma_x$. □

The abstract unification $\mathbf{U}_p$ is neither α- nor γ-optimal. The intuitive reason for this is that in the concrete unification, the substitutions produced depend on the particular unification performed. Thus, some substitutions that are in the arguments of the unification can disappear because of a unification failure. Clearly, this phenomenon cannot be taken care of by **Prop** at the abstract level.

Lemma 6.5. *$\mathbf{U}_p$ is neither α- nor γ-optimal.*

PROOF. The following counterexample shows that $\mathbf{U}_p$ is not α-optimal. Consider, $c = [\{\{x \mapsto a\}\}, \{x\}]$. Clearly,

$$\alpha_p(c) \; = \; [x, \{x\}]$$

In what follows $e_c = [\{id\}, \emptyset]$, and $e_p = [\mathsf{T}, \emptyset]$. These are dummy values that are used for the forward unification step, cf. Example 4.3. It is easy to see that:

$$\mathbf{U}_c(c, \; e_c, \; \{x \; \mapsto \; b\}) \; = \; [\emptyset, \; \{x\}]$$

and that

$$\begin{aligned} \alpha_p([\emptyset, \{x\}]) &= [\mathsf{F}, \{x\}], \text{ whereas} \\ \mathbf{U}_p([x, \{x\}], e_p, \{x \mapsto \; b\}) &= [x, \{x\}] \end{aligned}$$

A similar counterexample shows that $\mathbf{U}_p$ is not γ-optimal. Consider, $\gamma_p([x, \{x\}]) = [\Sigma_x, \{x\}]$. Obviously, Σ_x contains all substitutions that instantiate x to ground terms. Consider now,

$$\mathbf{U}_c([\Sigma_x, \; \{x\}], \; e_c, \; \{x \; \mapsto \; b\}) \; = \; [\Sigma, \; \{x\}]$$

Obviously, Σ does not contain any substitution that instantiates x to a. Hence $\Sigma \; \neq \; \Sigma_x$. However,

$$\begin{aligned} \gamma_p(\mathbf{U}_p([x, \{x\}], e_p, \{x \mapsto b\})) &= \gamma_p([x, \{x\}]) \\ &= [\Sigma_x, \{x\}] \end{aligned}$$

$\square$

Using the above Lemma, it is easy to show that **Prop** does not satisfy the statements of Theorems 6.1 and 6.2.

Lemma 6.6. For any logic program P, consider the functions $\mathcal{P}[\![P]\!]_{Rsub}$ and $\mathcal{P}[\![P]\!]_{Prop}$, the following two statements are false:

1. $\forall A \in$ *Atom and* $c \in Rsub$
   $\alpha_P(\mathcal{P}[\![P]\!]_{Rsub} A c) = \mathcal{P}[\![P]\!]_{Prop} A(\gamma_P c)$
2. $\forall A \in$ *Atom and* $\forall d \in$ **Prop**,
   $\mathcal{P}[\![P]\!]_{Rsub} A(\gamma_P d) = \gamma_P(\mathcal{P}[\![P]\!]_{Prop} A d)$

PROOF. It is easy to find counterexamples using the same ideas of the proof of the preceding lemma. For disproving both statements (1) and (2), consider the program P: $q(b)$ and the goal $q(x)$. For point (1) consider the element $[\{\{x \mapsto a\}\}, \{x\}] \in Rsub$, whose abstraction in **Prop** is $[x, \{x\}]$. It is easy to see, cf. the proof of the preceding Lemma, that:
$\mathcal{P}[\![P]\!]_{Rsub}\; q(x)\; [\{\{x \mapsto a\}\}, \{x\}] = [\{\;\}, \{x\}]$, whereas
$\mathcal{P}[\![P]\!]_{Prop}\; q(x)\; [x, \{x\}] \; = \; [x, \{c\}]$.
Obviously, $\alpha_P([\{\;\}, \{x\}]) \; = \; [\mathsf{T}, \{x\}]) \; \neq \; [x, \{x\}]$.
For point(2) it suffices to consider the element, $[x, \{x\}] \in Prop$ and its concretization in *Rsub*. $\square$

7. RELATED WORK

This section surveys the use of propositional formulas for abstract interpretation of logic programs. It then compares the domain constructions presented in this paper with those in [28, 29], discussing the differences.

The use of propositional formulas for representing variable groundness and equivalence was proposed for the first time by Marriott and Søndergaard in [26, 27]. However, some ambiguity remained about precisely which formulas were useful for representing substitutions. This ambiguity was clarified in [11] were it was shown that only positive formulas are useful and it was also proven that there is a Galois insertion between the set of positive formulas over some fixed finite set $\{x_1, .., x_n\}$ of variables and the concrete domain $\wp(Subst_n)$, where $Subst_n$ are the substitutions whose support is contained in $\{x_1, .., x_n\}$. It is easy to see (now) that the result is true also substituting $Subst_n$ with $Subst$.

More recently, in [29], a complete description of the abstract interpretation using positive formulas (called *Pos*) is given and the correctness of all its operations is proven. *Pos* is also studied in [28], where it is shown that it is condensing. In [12] it is shown that **Prop** is strictly more precise than *Sharing* [21], as far as the computation of groundness information is concerned. Recently, it has been shown that, in practice, an analysis based on positive formulas can be quite efficient [25, 6, 1, 34]. Positive formulas have been also used for type inference in pure Prolog programs [7], for the analysis of constraint languages, viz. for definiteness analysis, [9], and for detecting nonlinear constraints that are sure to become linear during any computation [20].

Turning the the second point of this section, there are two aspects of our presentation that differ from that of [29]. The first is the type of the abstract unification, which is discussed above in Section 4.2. The second aspect is the way in which we deal with the variables of interest: both *Rsub* and *Prop* consist of pairs where the second component explicitly gives the variables of interest.

Let us explain our choice starting from *Prop*. The simplest abstract domain based on positive formulas, is the set of all such formulas (together with the constant F). However, such a domain has a drawback. If we want it to be a complete lattice with a Galois insertion into the concrete domain, we have to fix a finite set U of variables and then consider only formulas on those variables, [11, 28]. In fact, if we would not restrict ourselves to formulas on a finite U, the domain would contain infinite formulas such as $\bigwedge_{i\geq 0} x_i$, obtained as glb of an infinite number of finite formulas. The presence of such formulas prevents the existence of a Galois insertion between the abstract and the concrete domain. In fact, such formulas approximate the empty set of substitutions (or the empty set of existentially quantified term equations, called ex-equations in [29]) because a substitution (ex-equation) constrains only a finite set of variables. Having to consider only formulas on a finite set of variables U seems inelegant to us because, in this way, there is not just one abstract domain but one for each set U, and a domain with “sufficient” variables must be chosen for analyzing any given program. Our construction of *Prop* avoids these problems.

The approach followed for the definition of *Prop* is simple and general. It can be applied to any abstract domain and has the effect of associating to each abstract value its variables of interest. The importance of this fact should not be underestimated. There are, in fact, abstract domains, such as the well-known domain *Sharing* of [21] for which is necessary to explicitly specify the variables of interest. This is true for *Sharing* because the absence of a variable in a *Sharing* value expresses that that variable is ground. Thus, abstract values should always contain all non-ground variables. Clearly this is incompatible with what is done in abstract analysis, viz. abstract values are projected onto the variables of a clause or of an atom forgetting the variables of other clauses (which can be still free).

In order to guarantee the existence of a Galois insertion between *Prop* and the concrete domain, we applied the same approach to our concrete domain *Rsub*. We have seen in Section 4.1 that an element of *Rsub* is a pair $[\Sigma, \mathrm{U}]$, where Σ is a set of substitutions and U a finite set of variables. We stress the fact that no restriction at all is imposed on the substitutions in Σ, i.e., any such substitution might contain some variable of U in its set of support and/or in its variable range or it might not contain any variable of U at all.

This concrete domain has two positive features. Its values are almost identical to the values (i.e., substitutions) computed by Prolog programs. Furthermore, the projection operation on *Rsub* is very simple: projecting $[\Sigma, U]$ onto U' yields $[\Sigma, U \cap U']$. This simplicity is very pleasing to us because it matches our feeling that projection does not belong to the concrete semantics, which only "inherits" it from the abstract semantics, where it is necessary for achieving finiteness.

Instead of using substitutions as the basis of the concrete domain, [29] use ex-equations. The variables of interest are then the free variables. This proposal is conceptually similar to ours: the unquantified variables of ex-equations are the variables of interest. In other words, one has to distinguish two types of variables: relevant ones (those of interest) and irrelevant ones. Indeed, it is well-known that sets of equations precisely correspond to substitutions [23]. However, since logic programs compute substitutions, a concrete domain based on substitutions seems slightly more natural. Set-intersection also seems to be a slightly simpler form of projection than the existential quantification of ex-equations. More importantly, however, it is unclear how one could use sets of ex-equations as the concrete domain while avoiding the inelegance of having to choose, for the analysis of any given program, a domain of formulas with "sufficient" variables.

8. CONCLUSIONS

The paper contains a complete and formal study of the abstract interpretation **Prop**. The main achievements of this study are as follows.

The formulas that are useful for groundness analysis are characterized both semantically and syntactically. The useful formulas are those that are true when all variables assume the value *true* and, equivalently, are the formulas obtained using the connectives $\wedge$ and $\leftrightarrow$. These results already appeared in [11].

On the basis of this characterization we give a complete and formal description of the abstract interpretation **Prop**, which is an elegant and practical basis for groundness analysis. We also outline a general method for identifying the variables of interest in concrete and abstract interpretations. By using this method, we construct a concrete interpretation that is based on the notion of substitutions.

We verify the correctness and optimality of the operations of **Prop**. Similar correctness results appear in [29]. To the best of our knowledge, ours are the first optimality results to be shown about any abstract interpretation for logic programs. Two new stronger forms of optimality are also introduced. It is shown that an abstract interpretation satisfying either of them is very precise. Unfortunately, even a domain as rich as **Prop** does not satisfy either of the two.

Acknowledgements

Detalied comments from the reviewers are gratefully acknowledged. This research was supported in part by *Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo* of CNR, under grant $n°91.000898$.PF69 and by NSF grant CCR-9210975.

REFERENCES

1. Amstrong T., Marriott K., Schachte P. and Søndergaard H., "Dependency analysis for logic programs." *Proc. 1st Static Analysis Symposium*, B. Le Charlier (ed.), LNCS, vol. 864, pp. 266–280, 1994.
2. Bell, J. L. and Machover, M., *A course in mathematical logic.* North Holland, 1977.
3. Bruynooghe, M., "A practical framework for the abstract interpretation of logic programs." *J. Logic Programming*, 10(2):91–124, (1991).
4. Cousot, P. and Cousot, R., "Abstract interpretation: a unified framework for static analysis of programs by construction or approximation of fixpoints." *Proc. Fourth ACM Symposium on Principles of Programming Languages*, 1977.
5. Cousot, P. and Cousot, R., "Abstract interpretation and application to logic programs." *J. Logic Programming* Vol. 13(2 and 3), pp.103–179, 1992.
6. Codish, M. and Demoen, B., "Analyzing logic programs using "Prop"-ositional logic programs and a magic wand." *Logic Programming — Proc., 1993 International Symposium.* D. Miller (ed.), MIT Press, pp. 114–129, 1993.
7. Codish, M. and Demoen, B., "Deriving polymorphic type dependencies for logic programs using multiple incarnations of Prop." *Proc. First Static Analysis Symposium*, LNCS Vol. 864, pp.281–296, 1994.
8. Codish, M., Falaschi, M. and Marriott, K., "Suspension analyses for concurrent logic programs." *ACM TOPLAS*, Vol. 16, pp.649–686, 1994.
9. Codognet, P. and Filé, G., "Computations, Abstractions and Constraints." *Proc. Int. Conf. on Computer Languages*, S.Francisco, 1992.
10. Cortesi, A. and Filè, G., "Abstract interpretation of logic programs: an abstract domain for groundness, sharing, freeness and compoundness analysis." *Proc. ACM Symposium on Partial Evaluation and Semantics Based Program Manipulation.* P. Hudak and N. Jones (eds.), SIGPLAN NOTICES Vol. 26, n.11, 1991.
11. Cortesi, A., Filè, G. and Winsborough W., "Prop revisited: Propositional formula as abstract domain for groundness analysis." *Proc. Sixth Annual IEEE Symposium on Logic in Computer Science.* G. Kahn (ed.), pp.322–327, Amsterdam, 1991.
12. Cortesi, A., Filè, G. and Winsborough W., "Comparison of Abstract Interpretations." *Proc. 19th International Colloquium on Automata, Languages and Programming.* LNCS 623, pp. 521–532, 1992.
13. Cortesi, A., Filè, G. and Winsborough W., "Optimal Groundness Analysis Using Propositional Formulas" Department of Mathematics, (Technical Report 94/11), University of Padova, 1994.
14. Cortesi, A., Van Hentenryck P. and Le Charlier B., "Combinations of Abstract Domains for Logic Programming". *Proc. 21th ACM Symposium on Principles of Programming Languages*, Portland 1994.
15. Dart, P., *Dependency analysis and query interfaces for deductive databases.* Ph.D. Dissertation, Department of Computer Science, (Technical Report 88/35), The University of Melbourne, Australia, 1988.
16. Dart, P.W., "On derived dependencies and connected databases." *J. Logic Programming,* Vol. 11, pp.163–188, 1991.
17. Dart, P.W. and Zobel, J., "Efficient Run-Time Checking of Typed Logic Programs." *J. Logic Programming*, Vol. 14, pp.31–69, 1992.

18. Debray, S. K., "Static inference of modes and data dependencies in logic programs." *ACM TOPLAS*, Vol. 11, pp.418–450, 1989.
19. Filé, G. and Ranzato, F., "Improving Abstract Interpretations by Systematic Lifting to the Powerset " *Proc. International Logic Programming Symposium*, Ithaca, 1994.
20. Hanus, M., "Analysis of Nonlinear Constraints in CLP(R)". *Proc. International Conference on Logic Programming*, pp. 83–99, Budapest, 1993.
21. Jacobs, D. and Langen, A., "Accurate and efficient approximation of variable aliasing in logic programs." *J. Logic Programming*, Vol. 13, pp. 291–314, 1992.
22. Jones, N. and Søndergaard, H., "A semantics based framework for the abstract interpretation of Prolog." In *Abstract Interpretation of Declarative Languages.* S. Abramsky and C. Hankin (eds.), Ellis Horwood, 1987.
23. Lassez, J.L., Maher, M.J. and Marriott, K.G., "Unification Revisited." In *Foundations of Deductive Databases and Logic Programming.* J. Minker (ed), 1986.
24. Lavrov I. A. and Maksimova L.L., *Problems in set theory, mathematical logics and algorithm theory*, 1984 (in Russian).
25. Le Charlier, B. and Van Hentenryck, P., "Groundness Analysis for Prolog: Implementation and Evaluation of the Domain `Prop`." *Proc. of the ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation.* Copenhagen, Denmark, June 1993.
26. Marriott, K. and Søndergaard, H., "Notes for a tutorial on abstract interpretation of logic programs." *North American Conference on Logic Programming.* MIT Press, 1989.
27. Marriott, K. and Søndergaard, H., "Abstract interpretation of logic programs: the denotational approach." *Proc. GULP'90.* A. Bossi (ed.), Padova, 1990.
28. Marriott, K. and Søndergaard, H., "Precise and Efficient Groundness Analysis for Logic Programs." *ACM LOPLAS* Vol. 2, pp.81–196, 1993.
29. Marriott, K., Søndergaard, H. and Jones, N., "Denotational Abstract Interpretation of Logic Programs." *ACM TOPLAS*, Vol. 16, pp.607–648, 1994.
30. Mellish, C. S., "The automatic generation of mode declarations for Prolog programs." *Proc. Workshop on Logic Programming for Intelligent Systems.* Los Angeles, 1981.
31. Post E. L., "The two valued iterative systems of mathematical logic." Vol. 5, *Annals of Math. Studies*, 1941.
32. Thatcher, J.W., Wagner, E.G., and Wright, J.B. , "More on advice on structuring compiler and proving them correct." *Theoretical Computer Science* Vol. 15, pp.223–249, 1981.
33. Van Hentenryck P., Cortesi, A. and Le Charlier, B., "Type Analysis of Prolog Using Type Graphs." *J. Logic Programming* Vol 22(3), pp.179–208,1995.
34. Van Hentenryck P., Cortesi, A. and Le Charlier, B., "Evaluation of the domain Prop." *J. Logic Programming* Vol. 23(3), pp.237–278, 1995.