Algebraic Properties of Idempotent Substitutions *

Catuscia Palamidessi

Dipartimento di Informatica,
Università di Pisa,
Corso Italia 40, 56125 Pisa, Italy

October 29, 1996

Abstract

This paper presents an algebra of idempotent substitutions whose operations have many properties. We provide an algorithm to compute these operations and we show how they are related to the standard composition. The theory of Logic Programming can be rewritten in terms of these new operations. The advantages are that both the operational and the declarative semantics of Horn Clause Logic can be formalized in a compositional way and the proofs of standard results, like the switching lemma, get easier and more intuitive. Moreover, this formalization can be naturally extended to a parallel computational model, and therefore it can be regarded as a basis for a theory of concurrent logic programming.

1 Introduction.

A nice aspects of logic programming [1], [18] is to be based on a rigorous mathematical theory: the Horn Clause Logic (HCL). A critical aspects is that this mathematical framework makes an extensive use of the notion of substitution. The standard operational semantics of HCL [23], and also some recent approaches to the declarative semantics [7], [8] are based on this notion. People working in the theory of logic programming usually notice that the substitutions are a quite hard matter to deal with. In particular, it is problematic to deal with the only operation available on substitutions, the composition. The problem is that it allows to exploit very few properties: only the associativity and the existence of the neutral element. As a consequence, many proofs are quite involved, when compared to the intuition. Many notions based on compositions, like the preordering on substitutions ($\vartheta \leq \sigma$ iff $\exists \psi : \vartheta\psi = \sigma$) and the equivalence ($\vartheta \sim \sigma$ iff $\vartheta \leq \sigma$ and $\sigma \leq \vartheta$) are indeed not very intuitive. For instance, we have $\{x/y, y/x\} \sim \epsilon$ (the empty substitution), when we would expect $\{x/y, y/x\}$ to contain "more information" than ϵ, and $\{x/y\}\{x/a\} = \{x/y\}$, when we would expect $\{x/a\}$ to add "something" to $\{x/y\}$.

Another point on which the notion of composition results to be inadequate is related to the problem of describing the semantics of HCL in a compositional way, i.e. such that the meaning of a construct is a function of the meanings of its components (see [5]). Such a function cannot be based on the notion composition. Consider, for instance, the substitutions $\vartheta_1 = \{x/y, z/y\}$ and $\vartheta_2 = \{x/z, y/z\}$. We have $\vartheta_1 \sim \vartheta_2$, so, intuitively, they should be interchangeable in the semantics description of any

*This work is partially supported by ESPRIT 415: Parallel Architectures and Languages for Advanced Information Processing - A VLSI-directed approach.

construct. Unfortunately, if we compose now with a substitution $\sigma = \{z/a\}$, we get $\vartheta_1\sigma = \{x/y, z/y\} \not\sim \vartheta_2\sigma = \{x/a, y/a, z/a\}$.

The problem becomes even more relevant when dealing with the concurrent models of logic programming. Here an adequate operation is required in order to formalize the combination of the results obtained by two parallel computations. Again, this combination cannot be based simply on composition. Consider for instance the goal

$$G \equiv\leftarrow p(x, z), q(z)$$

and the program

$$\{ \quad p(y, y) \leftarrow . \quad q(a) \leftarrow . \quad \}$$

If we run the subgoals $G_1 \equiv\leftarrow p(x, z)$ and $G_2 \equiv\leftarrow q(z)$ in parallel, we get (equivalently) the answer $\vartheta_1 = \{x/y, z/y\}$ or $\vartheta_2 = \{x/z, y/z\}$ for G_1, and $\sigma = \{z/a\}$ for G_2. If we look now to the whole goal G, its intended semantics should contain $\vartheta_2\sigma = \{x/a, y/a, z/a\}$, but not $\vartheta_1\sigma = \{x/y, z/y\}$. The problem of compositionality in the semantics of concurrent logic languages has been addressed in [16], [2], [20], [3], [4], [11].

An approach to solve this problem is to replace the notion of substitution by the one of *equation set*. This possibility is based on the remark that substitutions are essentially used, in logic programming, as most general unifiers, i.e. for representing the general solutions to questions of the form "for which values of x and y the atoms $A[x]$ and $B[y]$ are identical?" Since the basic step of logic programming is the comparison between an atom of the goal and the head of a clause, it is sufficient to transform all the clauses of a program in their standard form. Namely, each clause

$$p(t_1, \ldots, t_m) \leftarrow B_1, \ldots, B_n$$

is replaced by

$$p(x_1, \ldots, x_m) \leftarrow x_1 = t_1, \ldots x_m = t_m, B_1, \ldots, B_n$$

where the symbol $=$ represents the syntactical identity. The operational semantics of HCL becomes much easier and elegant, the only intermediate computed substitutions are variable-pure substitutions, the composition is replaced by the union of equations, and the final answer substitution is obtained by solving all the equations. A nice theory for equation syntactic unification has been developed in [17]. This approach presents the advantage of being elegantly extendible to Equational+Logic programming (namely, HCL plus some equational theory), see [15]. Moreover, it can be generalized to the theory of Constraint Logic Programming, see [14]. Finally, when applied to Concurrent Logic Languages, it allows to use an uniform concept of guard [19], [21], [22], [10], [9]. The main disadvantage is that we have, at each step, only an *implicit* notion of result, in terms of an equation set, and this can lead to useless computations if the consistency check is done only at the end (of the computation). On the other side, the consistency check of an equation set E is as much expensive as reducing E to its *solved form*. Therefore performing the consistency check at each step is as much redundant as computing the solved form of E at one step and starting again from (a set containing) E at the next step.

We would like to preserve the advantages of this approach and to overcome this redundancy. The natural solution is to get back to the notion of substitution, that represents, in a sense, the solved form of a set of equations, and gives the intermediate results in an *explicit* form.

Our approach is based on the lattice of idempotent substitutions firstly introduced by Eder [6]. It turns out that this structure is very similar to the one based on equations, investigated in [17]. Actually, the two structures are isomorphic, apart from some particular cases. In this paper we investigate a new lattice of equations, that is always isomorphic to the lattice of idempotent substitutions, and it is more suitable for the semantics of logic programming.

The main drawback of dealing with idempotent substitutions is that the standard operation, the composition, does not preserve the property of idempotence [1]. We often find this remark, in the literature, as a motivation for not restricting to idempotent substitutions. This will not be a problem anymore, indeed we show that we can do without this operation. We define two operations on idempotent substitutions, the parallel composition and the parallel factorization, that correspond to the least upper bound and to the greatest lower bound respectively, and that inherit therefore all their properties:

- commutativity,
- associativity,
- existence of the neutral element,
- idempotence,
- existence of the null element,
- weak distributivity,
- absorption.

An algorithm to compute the least upper bound has been informally described by Eder [6], but it still makes use of the notion of standard composition. An other algorithm ("reconciliation"), presented in [13], is based on the correspondence between the poset of substitutions and the poset of terms. The algorithm we present has been introduced and used in [20], [3], [4], [5] to define an operator (the *parallel composition*) suitable to describe in a compositional way the semantics of languages based on HCL. What was still missing there was the relation between this operator and the least upper bound, and the formalization of its properties. The same algorithm was also introduced independently in [13] as an alternative way to compute the "reconciliation".

Regarding to the greatest lower bound, it could be computed by using the algorithm described in [17] to generate the more specific generalization of a set of sets of equations. Namely, we could transform the substitutions into sets of equations, compute their more specific generalization, and transform again the equation set obtained so far into its solved form. This algorithm is based on the Huet's algorithm for anti-unification [12], that usually introduces extra-variables, which have to be dropped again afterwards. We give a direct algorithm that does not require any transformation into equational form, and no introduction of extra-variables.

In order to embed this formalism in the standard theory of logic programming, we show how these operations are related to standard composition. As a side result, this provides a proof of the correctness of Eder's algorithm for computing the least upper bound. We show how the composition can be defined in terms of the parallel composition, in all the cases relevant for logic programming. Then, we can completely replace the first notion by the latter one, in the semantics of HCL. We will see that the whole theory gains a lot in terms of compositionality, intuitiveness, and easiness in making proofs. Finally, we will see that this formalization can be naturally extended to a parallel computational model, and therefore it can be regarded as a basis for a theory of concurrent logic programming.

[1] Note, however, that in the semantics of HCL the variable reniming of clauses used in the resolution steps guarrantees the presevation of idempotence in the computed substitutions.

The next section recalls some basic notions about substitutions, composition, and unification. In section 3 we define the lattice of idempotent substitutions, and we relate it to a lattice of equations. In sections 4 and 5 we give the algorithms for computing the least upper bounds and the greatest lower bounds, and we show their correctness. Finally, in section 6 we describe the relation of the least upper bound with the standard composition, we give a reformulation of the operational semantics of logic programming, and we discuss its applications to the concurrent logic programming.

2 Preliminaries

In this section we briefly recall some basic notions and results about substitutions and unification. We will use (mainly) the terminology of [1], [18], [17], and [6], to which we refer for a more detailed presentation of these topics.

2.1 Substitutions

Let Var be a non empty set of *variables*, with typical elements $x, y, z, \ldots$. Let $Cons$ be a set of *constructors*, with typical elements $a, b, c, \ldots$ (constructors with 0 arguments, or *constants*), and $f, g, h, \ldots$ (constructors with one or more arguments). We assume the presence of at least one constant.

The set $Term$, with typical elements $t, u, \ldots$, is the set of all the *terms* built on Var and on $Cons$. Examples of terms are $f(a), f(x), g(f(a), f(x)), \ldots$. The set of variables occurring in t is denoted by $var(t)$.

The set of *substitutions*, $Subst$, with typical elements $\vartheta, \sigma, \ldots$, consists of all the mappings ϑ from Var into $Term$ such that the domain of ϑ, $dom(\vartheta) = \{x \in Var \mid \vartheta(x) \not\equiv x\}$, is finite. We will use the set-theoretic notation $\{x/t \mid x \in dom(\vartheta) \wedge \vartheta(x) \equiv t\}$ to represent ϑ.

The *application* $t\vartheta$ of ϑ to t is defined as the term obtained by replacing each variable x in t by $\vartheta(x)$. The set $ran(\vartheta)$ (range of ϑ) is defined as $\bigcup_{x \in dom(\vartheta)} var(\vartheta(x))$. The substitution ϑ is *ground* iff $ran(\vartheta) = \emptyset$.

The *composition* $\vartheta\vartheta'$ of ϑ and ϑ' is defined by $(\vartheta\vartheta')(x) \equiv (\vartheta(x))\vartheta'$. We recall that the composition is associative, the empty substitution ϵ is the neutral element, and, for each term t, $t(\vartheta\vartheta') \equiv (t\vartheta)\vartheta'$ holds. A substitution ϑ is *idempotent* iff $\vartheta\vartheta = \vartheta$ (or, equivalently, iff $dom(\vartheta) \cap ran(\vartheta) = \emptyset$).

A substitution ρ is a *renaming* iff it is a bijection from Var to Var. Equivalently, ρ is a renaming iff there exists its inverse, i.e. a substitution ρ^{-1} such that $\rho\rho^{-1} = \rho^{-1}\rho = \epsilon$. In this paper we will use this notion of renaming, that has been introduced in [6], and corresponds to what is called *permutation of variables* in [17]. The definitions of renaming given in [1] and [18] are slightly different.

A substitution ϑ is *more general* than ϑ', $\vartheta \leq \vartheta'$, iff there exists σ such that $\vartheta\sigma = \vartheta'$. The associated equivalence relation will be denoted by $\sim$. An equivalent definition for $\sim$ is: $\vartheta \sim \vartheta'$ iff there exists a renaming ρ such that $\vartheta\rho = \vartheta'$. $\leq$ is a pre-ordering on $Subst$ (i.e. it is a reflexive and transitive relation) and it induces a partial ordering on $Subst_{/\sim}$ (the set of the equivalence classes of $Subst$ w.r.t. $\sim$), that will still be denoted by $\leq$.

2.2 Unification

The notion of *unification* can be given, equivalently, either with respect to a set of sets of terms [6], [18], or with respect to a set of equations [17], [1]. We choose the second approach.

An *equation* is an expression of the form $t = u$, where t and u are terms. A set of equations E is *unifiable* iff there exists ϑ such that for all $t = u$ in E, $t\vartheta \equiv u\vartheta$ holds, where $\equiv$ is the syntactic identity on terms. ϑ is then called an *unifier* of E. The set of all the unifiers of E will be denoted by $\mathit{unif}(E)$. The set of all the *most general unifiers* of E is given by $\mathit{mgu}(E) = \{\vartheta \in \mathit{unif}(E) \mid \forall \vartheta' \in \mathit{unif}(E).\vartheta \leq \vartheta'\}$. It is well known that all the most general unifiers of a set E are equivalent, and this explains why, in the literature, we often find the terminology "*the* mgu of E". Furthermore, if E is unifiable, then there exists an idempotent most general unifier of E.

3 The complete lattice of idempotent substitutions

The monoid $(\mathit{Subst}, \cdot)$ (where $\cdot$ represents here the composition of substitutions), and the partial order $(\mathit{Subst}_{/\sim}, \leq)$ have a very poor structure. In particular, the composition of substitutions is not commutative and it is not compatible with the equivalence relation (i.e., there exist $\vartheta, \vartheta', \sigma$ such that $\vartheta \sim \vartheta'$ but $\vartheta\sigma \not\sim \vartheta'\sigma$, cfr. remark 2.4 in [6]). Moreover, there exist sets of (classes of) substitutions that are bounded from above but have no least upper bound (cfr. example 2.7 in [6]). On the contrary, the set of idempotent substitutions $\mathit{ISubst}_{/\sim}$ can be provided with operations having elegant and useful properties. In particular, $(\mathit{ISubst}_{/\sim}, \leq)$ can be embedded into a complete lattice [6]. For the sake of uniformity we give here a definition that is slightly different from the original one, but equivalent to it.

Definition 3.1 *Let τ be an arbitrary object that is not an element of ISubst. Let ISubst^{τ} be the set $\mathit{ISubst} \cup \{\tau\}$. We extend the partial ordering $\leq$ and the equivalence $\sim$ to ISubst^{τ} by requiring τ to be the greatest element of ISubst^{τ}. We denote by $\mathit{ISubst}^{\tau}_{/\sim}$ the new quotient set and by $\vartheta^{\sim}$ the equivalence class in which ϑ lies.*

Theorem 3.2 (Eder, [6]) *$(\mathit{ISubst}^{\tau}_{/\sim}, \leq)$ is a complete lattice, i.e. every subset of $\mathit{ISubst}^{\tau}_{/\sim}$ has a lub (least upper bound) and a glb (greatest lower bound). The smallest element is $\epsilon^{\sim}$ and the greatest element is $\tau^{\sim}$.*

Remark 3.3 *Here we have followed the original presentation of [6], but in order to get a complete lattice it is not always necessary to add an external element. More precisely, it is not necessary when Var is finite and the Herbrand universe is trivial, i.e. it contains only an element, say a. In fact, in this case we have that $\{x/a \mid x \in \mathit{Var}\} \in \mathit{ISubst}$ is less general than every other idempotent substitution. In the following, in this special case, we consider τ to be defined as $\{x/a \mid x \in \mathit{Var}\} \in \mathit{ISubst}$.*

There is a close relation between $(\mathit{ISubst}^{\tau}_{/\sim}, \leq)$ and the complete lattice of equations sets introduced in [17]. Actually, the two structures are isomorphic. The results showed in the next section can be considered immediate consequences of properties investigated in [17], but some of our proofs are simpler. Indeed, we deal directly with the unifiers of equations sets, instead of their *solutions* (ground unifiers), as it is done in [17]. This allows to obtain a more immediate correspondence between equations sets and idempotent substitutions. Moreover, we get an isomorphism result that holds in general, thus extending the isomorphism result stated in [17]. This last one indeed does not hold when the Herbrand universe is trivial.

3.1 Idempotent substitutions and equation sets

Each substitution ϑ "naturally" corresponds to a set of equations given by $\mathcal{E}(\vartheta) = \{x = t \mid x/t \in \vartheta\}$. We will show that this correspondence $\mathcal{E}$, when restricted to idempotent substitutions, actually gives an isomorphism (up to the equivalence relation). The next results, from proposition 3.4 to proposition 3.7, provide some useful relations between a substitution and the set of unifiers of its associated equation set.

Proposition 3.4 *$\sigma \in unif(\mathcal{E}(\vartheta))$ iff $\vartheta\sigma = \sigma$.*

Proof

if) Let $x/t \in \vartheta$. Then, $x\sigma \equiv x\vartheta\sigma \equiv t\sigma$.

only if) Let $x \in dom(\vartheta)$, say $x/t \in \vartheta$. We have $x\sigma \equiv t\sigma \equiv x\vartheta\sigma$. Consider now $x \notin dom(\vartheta)$. We have $x\vartheta \equiv x$, therefore $x\sigma \equiv x\vartheta\sigma$. □

From this proposition we derive a characterization of the idempotent substitutions.

Corollary 3.5 *ϑ is idempotent iff $\vartheta \in unif(\mathcal{E}(\vartheta))$*

Proof Immediate, by setting σ to ϑ in the proposition 3.4. □

Actually, the idempotent substitutions are characterized by a stronger property:

Corollary 3.6 *ϑ is idempotent iff $\vartheta \in mgu(\mathcal{E}(\vartheta))$.*

Proof

if) Immediate from corollary 3.5.

only if) From corollary 3.5 we have $\vartheta \in unif(\mathcal{E}(\vartheta))$. Let σ be an unifier of $\mathcal{E}(\vartheta)$. From proposition 3.4, we have $\vartheta\sigma = \sigma$ and therefore $\vartheta \leq \sigma$. □

The following proposition gives a useful characterization of the ordering relation on idempotent substitutions, in terms of the unifiers of the associated sets.

Proposition 3.7 *Let ϑ_1, ϑ_2 be idempotent substitutions. Then*

$\vartheta_1 \leq \vartheta_2$ iff $unif(\mathcal{E}(\vartheta_2)) \subseteq unif(\mathcal{E}(\vartheta_1))$.

Proof

if) Since ϑ_2 is idempotent, then, by corollary 3.5, $\vartheta_2 \in unif(\mathcal{E}(\vartheta_2))$ holds. Thus, by hypothesis, we have $\vartheta_2 \in unif(\mathcal{E}(\vartheta_1))$, and therefore, by proposition 3.4, $\vartheta_1\vartheta_2 = \vartheta_2$ holds.

only if) Let σ be a substitution such that $\vartheta_1\sigma = \vartheta_2$. Let $\vartheta \in unif(\mathcal{E}(\vartheta_2))$. We have

$$\begin{array}{ll}
\vartheta_1\vartheta & = \text{(by proposition 3.4, only if part)} \\
\vartheta_1\vartheta_2\vartheta & = \\
\vartheta_1\vartheta_1\sigma\vartheta & = \text{(by idempotency of } \vartheta_1 \text{)} \\
\vartheta_1\sigma\vartheta & = \\
\vartheta_2\vartheta & = \text{(by proposition 3.4, only if part)} \\
\vartheta &
\end{array}$$

therefore (by proposition 3.4, if part), $\vartheta \in unif(\mathcal{E}(\vartheta_1))$ holds. □

Definition 3.8 (Lassez, Maher, and Marriot, [17]) *The set of solutions of an equation set E, $soln(E)$, is the set of all the ground unifiers of E.*

This notion of $soln(E)$ is used in [17] in order to define the complete lattice of equation sets. For the sake of uniformity, we will use a terminology different from the original one. Namely, we revert the ordering and define

$$E_1 \preceq E_2 \text{ iff } soln(E_2) \subseteq soln(E_1)$$

The relation $\preceq$ is a preordering on the set Eqn of all the equation sets. Therefore, it induces an equivalence relation that we denote by $\approx$, and a partial ordering, on the quotient set, that we still represent by $\preceq$. The equivalence class of E will be denoted by $E^{\approx}$.

Theorem 3.9 (Lassez, Maher, and Marriot, [17]) *$(Eqn_{/\approx}, \preceq)$ is a complete lattice where the lub of $\{E_1^{\approx}, \ldots, E_n^{\approx}, \ldots\}$ is given by $(E_1 \cup \ldots E_n \cup \ldots)^{\approx}$. The smallest element is $\emptyset^{\approx}$ and the greatest element is $E_G^{\approx}$, where E_G is*

- *an equation without solutions, if the Herbrand universe is not trivial*
- *$\emptyset$ otherwise (in this case, the lattice is a singleton).*

We introduce here a notion of ordering on equation sets that is more convenient for our purposes, and coincides with the previous one when the Herbrand Universe is not trivial. We define:

$$E_1 \leq E_2 \text{ iff } unif(E_2) \subseteq unif(E_1)$$

and we denote by $\sim$ the corresponding equivalence relation.

Theorem 3.10 *If the Herbrand universe is not trivial then $E_1 \leq E_2$ iff $E_1 \preceq E_2$.*

Proof

if) Let $soln(E_2) \subseteq soln(E_1)$ and $\vartheta \in unif(E_2)$. We want to prove that $\vartheta \in unif(E_1)$. By definition, for all the ground substitutions σ, we have $\vartheta\sigma \in soln(E_2)$. Then, by hypothesis, $\vartheta\sigma \in soln(E_1)$ holds. Consider now an equation $t = u$ in E_1. We want to prove $t\vartheta \equiv u\vartheta$ (i.e, ϑ is a unifier of $t = u$). We proceed by structural induction. If $t\vartheta$ is a variable, say x, then also $u\vartheta$ must contain a variable. Otherwise a ground substitution including $\{x/u'\}$, with $u' \not\equiv u\vartheta$, would not unify $t\vartheta = u\vartheta$. If $u\vartheta$ contains a variable $y \not\equiv x$ then a ground substitution including $\{x/u', y/u''\}$, with $u' \not\equiv u''$ would not unify $t\vartheta = u\vartheta$. Then only the variable x occurs in $u\vartheta$. Moreover, $u\vartheta \equiv x$ must hold, otherwise a ground substitution including $\{x/u'\}$ would not be a unifier. Then we have $t\vartheta \equiv u\vartheta$. Now, assume $t\vartheta$ not to be a variable, say $t\vartheta \equiv f(t_1, \ldots, t_m)$. By the same reason as before, $u\vartheta$ cannot be a variable. Assume then $u\vartheta \equiv g(u_1, \ldots, u_n)$. Then, $f \equiv g$ (and $m = n$) must hold. Moreover, by structural induction, we have $t_1\vartheta \equiv u_1\vartheta, \ldots, t_n\vartheta \equiv u_n\vartheta$. Therefore $t\vartheta \equiv u\vartheta$ holds.

only if) Immediate by definition (note that this part is true also when the Herbrand universe is trivial). □

In theorem 3.10, the hypothesis that the Herbrand universe is not trivial is essential. The following is a simple counterexample.

Example 3.11 *Assume that the Herbrand universe consists of the constant a only. Let $E_1 = \{x = a, y = a\}$ and $E_2 = \{x = y\}$. Then, $soln(E_1) = soln(E_2)$, whilst the substitution $\{x/y\}$ is a unifier for E_2 but not for E_1.*

A consequence of theorem 3.10 is that if the Herbrand universe is not trivial, then $(Eqn_{/\approx}, \preceq)$ and $(Eqn_{/\sim}, \leq)$ are isomorphic. If the Herbrand universe is trivial, say $\{a\}$, then $Eqn_{/\approx}$ consists of one element only, $\emptyset^{\approx}$, whilst $Eqn_{/\sim}$ usually contains more elements, unless Var is empty. (In this last case $Eqn_{/\sim}$ is also trivial.) For instance, E_2 and E_1 of example 3.11 are in different equivalence classes w.r.t. $\sim$. However, $(Eqn_{/\sim}, \leq)$ is still a complete lattice. The greatest element is given by $E_T^{\sim}$, where $E_T = \{x = a \mid x \in Var\}$ (note that if Var is infinite then E_T is not unifiable). We show this by relating $(Eqn_{/\sim}, \leq)$ and $(ISubst^{\tau}_{/\sim}, \leq)$.

Definition 3.12 *We extend $\mathcal{E}$ to a mapping from $ISubst^{\tau}_{/\sim}$ to $Eqn_{/\sim}$ as follows:*

$$\mathcal{E}(\vartheta^{\sim}) = \begin{cases} \mathcal{E}(\vartheta)^{\sim} & \textit{if } \vartheta \neq \tau \\ E_T^{\sim} & \textit{otherwise} \end{cases}$$

Note that if the Herbrand universe is trivial and Var is finite then the two cases in the definition of $\mathcal{E}(\vartheta^{\sim})$ coincide. This definition is correct (i.e. it does not depend on the choice of the element in $\vartheta^{\sim}$). In fact, by proposition 3.7, $\vartheta_1 \sim \vartheta_2$ implies $unif(\mathcal{E}(\vartheta_1)) = unif(\mathcal{E}(\vartheta_2))$ and this means $\mathcal{E}(\vartheta_1)^{\sim} = \mathcal{E}(\vartheta_2)^{\sim}$.

The following theorem states the isomorphism between the lattice $(ISubst^{\tau}_{/\sim}, \leq)$ and the lattice $(Eqn_{/\sim}, \leq)$. This property extends the one stated in [17], concerning the isomorphism between $(ISubst^{\tau}_{/\sim}, \leq)$ and $(Eqn_{/\approx}, \preceq)$. Indeed, the second one does not hold when the Herbrand universe is trivial. Note that in [17] this correspondence is shown by making use of the following property (cfr. theorem 6 in [17])

$$\vartheta \in mgu(E) \text{ iff } \mathcal{E}(\vartheta) \approx E$$

that does not hold in the trivial case. Example 3.14 will provide a counterexample to it. The general correspondence is

$$\vartheta \in mgu(E) \text{ iff } \mathcal{E}(\vartheta) \sim E$$

Theorem 3.13 *$(Eqn_{/\sim}, \leq)$ and $(ISubst^{\tau}_{/\sim}, \leq)$ are isomorphic. Namely, there exists a bijective mapping between $Eqn_{/\sim}$ and $ISubst^{\tau}_{/\sim}$ that preserves the orderings.*

Proof We show that $\mathcal{E}$ is a bijection from $ISubst^{\tau}_{/\sim}$ to $Eqn_{/\sim}$ and that $\mathcal{E}(\vartheta_1) \leq \mathcal{E}(\vartheta_2)$ iff $\vartheta_1 \leq \vartheta_2$.

Injectivity If $\vartheta_1^{\sim} \neq \vartheta_2^{\sim}$ then $\vartheta_1 \not\sim \vartheta_2$, and therefore, by proposition 3.7, $\mathcal{E}(\vartheta_1)^{\sim} \neq \mathcal{E}(\vartheta_2)^{\sim}$ holds.

Surjectivity Consider an element $E^{\sim} \in Eqn_{/\sim}$. If E is unifiable, then take an idempotent $\vartheta \in mgu(E)$. We show that $\mathcal{E}(\vartheta) \sim E$.

$\mathbf{unif(\mathcal{E}(\vartheta)) \subseteq unif(E)}$ Let $\sigma \in unif(\mathcal{E}(\vartheta))$. Then, by proposition 3.4, we have $\vartheta \leq \sigma$. Since $\vartheta \in mgu(E)$, we have that $\sigma \in unif(E)$.

$\mathbf{unif(E) \subseteq unif(\mathcal{E}(\vartheta))}$ Consider now $\sigma \in unif(E)$. Since $\vartheta \in mgu(E)$, we have that $\exists \sigma' : \sigma = \vartheta\sigma'$. Then

$$\begin{array}{ll} \vartheta\sigma & = \\ \vartheta\vartheta\sigma' & = \text{(by idempotency of } \vartheta \text{)} \\ \vartheta\sigma' & = \\ \sigma. & \end{array}$$

By proposition 3.4 we have then $\sigma \in unif(\mathcal{E}(\vartheta))$.

If E is not unifiable, then $E \sim E_T$. Therefore $E^{\sim}$ is given by $\mathcal{E}(\tau^{\sim})$.

Figure 1: correspondence

Ordering-preservation Let $\vartheta_1 \leq \vartheta_2$. If $\vartheta_2 \neq \tau$, then, by proposition 3.7 we have $\mathcal{E}(\vartheta_1)^{\sim} \leq \mathcal{E}(\vartheta_2)^{\sim}$. If $\vartheta_2 = \tau$ it follows by definition. $\square$

Figure 1 visualizes the correspondence stated in theorem 3.13 and it gives an overview of the proof.

Example 3.14 *Let $Var = \{x, y\}$ and assume that the Herbrand universe contains the constant $\{a\}$ only. We have five elements in $ISubst_{/\sim}$:*

- $\vartheta_1^{\sim} = \epsilon^{\sim}$,
- $\vartheta_2^{\sim} = \{x/a\}^{\sim}$,
- $\vartheta_3^{\sim} = \{x/y\}^{\sim}$,
- $\vartheta_4^{\sim} = \{y/a\}^{\sim}$,
- $\vartheta_5^{\sim} = \{x/a, y/a\}^{\sim}$,

The ordering is represented in figure 2.

Consider the equation set $E = \emptyset$. We have $soln(\vartheta_5) = \{\vartheta_5\} = soln(E)$, whilst $\vartheta_5 \notin mgu(E) = \{\epsilon\}$. This is a counterexample to theorem 6 in [17]. More in general, the equations sets are all equivalent with respect to $\approx$. In fact, $\{\vartheta_5\}$ is *the only possible solution set and it is not possible to build an equation set without solutions. Thus, $Eqn_{/\approx}$ contains only one element. This is a counterexample to the isomorphism between $(Eqn_{/\approx}, \preceq)$ and $(ISubst_{/\sim}, \leq)$.*

On the other side, this is not a counterexample when $(Eqn_{/\sim}, \preceq)$ is considered. In fact, $unif(\vartheta_5) = \{\vartheta_5\}$ and $unif(E) = \{\epsilon\}$, Therefore $\mathcal{E}(\vartheta) \not\sim E$. Also the set $Eqn_{/\sim}$ consists of five elements, given by the $\mathcal{E}(\vartheta_i)$'s.

Figure 2: example

We want to analyze now the effectiveness properties of the structure $(ISubst^{\mathcal{T}}_{/\sim}, \leq)$. The operations of *lub* and *glb* are both computable in finite time, when performed on finite sets[2]. In the following sections we give the algorithms to compute them and we prove their correctness.

Eder [6] has already presented a way to determine the *lub*. Namely, $lub(\vartheta, \sigma)$ can be obtained as the composition of ϑ and the *mgu* of the set $\{x\vartheta = t\vartheta \mid x/t \in \sigma\}$. We will give a definition that makes no use of the composition, and that deals with ϑ and σ in a symmetric way. Therefore we consider our formulation slightly more elegant. Moreover, Eder does not give a formal correctness proof. We will show in the last section how his proof can be derived from the correctness of our algorithm.

We end this section with a remark [3].

Remark 3.15 *The lub on $(ISubst^{\mathcal{T}}_{/\sim}, \leq)$ obviously coincides with the lub on $(Subst^{\mathcal{T}}_{/\sim}, \leq)$, whilst this is not the case for the glb.*

Example 3.16 *Let $\vartheta = \{x/f(a,a)\}$ and $\sigma = \{x/f(a,b)\}$. Then $glb(\vartheta, \sigma) = \{x/f(a,x)\}$ in $(Subst^{\mathcal{T}}_{/\sim}, \leq)$ whilst $glb(\vartheta, \sigma) = \epsilon$ in $(ISubst^{\mathcal{T}}_{/\sim}, \leq)$. Notice that $glb(\vartheta, \sigma) \neq \{x/f(a,y)\}$ since $\{x/f(a,y)\}\{y/a\} = \{x/f(a,a), y/a\} \neq \vartheta$.*

4 Least upper bound as parallel composition

We introduce the notion of *parallel composition* on substitutions, denoted by $\uparrow$. Intuitively, the parallel composition is meant to be the formalization of one of the basic operations performed by the *parallel execution model* of logic programs. When two atoms A_1 and A_2 (in the same goal) are run in parallel, the associated computed

[2] Actually, the *glb* is finitely computable also when performed on an infinite set of substitutions. This can be derived from an analogous result concerning anti-unification of terms (cfr. proposition 18 in [17]). However, although very interesting, this topic is out of the scope of this paper.

[3] This remark, and the following example, are due to Jean-Marie Jacquet.

answer substitutions ϑ_1 and ϑ_2 have to be combined, afterwards, in order to get the final result. This operation can be performed in the following way. Consider the set of all the pairs corresponding to the bindings of both ϑ_1 and ϑ_2. Then, compute the most general unifier of such a set. Note that the *consistency check* corresponds to a verification that such a set is unifiable.

Definition 4.1 (Parallel composition) *We define the operator* $\uparrow$: $\mathit{ISubst}_{/\sim} \times \mathit{ISubst}_{/\sim} \to \mathit{ISubst}_{/\sim}$ *as follows*

$$\vartheta_1^{\sim} \uparrow \vartheta_2^{\sim} = \mathit{mgu}(\mathcal{E}(\vartheta_1) \cup \mathcal{E}(\vartheta_2)).$$

(here the mgu is meant to be restricted to the idempotent substitutions).

Definition 4.1 is correct (i.e. it does not depend on the choice of the elements in the equivalence classes). In fact, by proposition 3.7, if $\vartheta_1 \sim \sigma_1$ and $\vartheta_2 \sim \sigma_2$, then

$$\mathit{unif}(\mathcal{E}(\vartheta_1)) = \mathit{unif}(\mathcal{E}(\sigma_1)) \text{ and } \mathit{unif}(\mathcal{E}(\vartheta_2)) = \mathit{unif}(\mathcal{E}(\sigma_2)).$$

This implies

$$\begin{array}{ll} \mathit{unif}(\mathcal{E}(\vartheta_1) \cup \mathcal{E}(\vartheta_2)) & = \\ \mathit{unif}(\mathcal{E}(\vartheta_1)) \cap \mathit{unif}(\mathcal{E}(\vartheta_2)) & = \\ \mathit{unif}(\mathcal{E}(\sigma_1)) \cap \mathit{unif}(\mathcal{E}(\sigma_2)) & = \\ \mathit{unif}(\mathcal{E}(\sigma_1) \cup \mathcal{E}(\sigma_2)). & \end{array}$$

And therefore

$$\mathit{mgu}(\mathcal{E}(\vartheta_1) \cup \mathcal{E}(\vartheta_2)) = \mathit{mgu}(\mathcal{E}(\sigma_1) \cup \mathcal{E}(\sigma_2))$$

from which it follows

$$\sigma_1 \uparrow \sigma_2 = \vartheta_1 \uparrow \vartheta_2.$$

From now on, for the sake of simplicity, we will indicate the operation $\vartheta_1^{\sim} \uparrow \vartheta_2^{\sim}$ simply by $\vartheta_1 \uparrow \vartheta_2$.

The following proposition shows that the parallel composition exactly corresponds to the least upper bound on (classes of) substitutions.

Proposition 4.2 *Let ϑ_1, ϑ_2 be idempotent substitutions. Then $\vartheta_1 \uparrow \vartheta_2 = \mathit{lub}\{\vartheta_1^{\sim}, \vartheta_2^{\sim}\}$.*

Proof

$$\begin{array}{ll} \mu \in \vartheta_1 \uparrow \vartheta_2 & \text{iff (by definition)} \\ \mu \in \mathit{mgu}(\mathcal{E}(\vartheta_1) \cup \mathcal{E}(\vartheta_2)) & \text{iff (by corollary 3.6)} \\ \mathit{mgu}(\mathcal{E}(\mu)) = \mathit{mgu}(\mathcal{E}(\vartheta_1) \cup \mathcal{E}(\vartheta_2)) & \text{iff} \\ \mathit{unif}(\mathcal{E}(\mu)) = \mathit{unif}(\mathcal{E}(\vartheta_1) \cup \mathcal{E}(\vartheta_2)) & \text{iff} \\ \mathit{unif}(\mathcal{E}(\mu)) = \mathit{unif}(\mathcal{E}(\vartheta_1)) \cap \mathit{unif}(\mathcal{E}(\vartheta_2)) & \text{iff (by proposition 3.7)} \\ \mathit{mgu}(\mathcal{E}(\mu)) = \mathit{lub}\{\vartheta_1^{\sim}, \vartheta_2^{\sim}\} & \text{iff (by corollary 3.6)} \\ \mu \in \mathit{lub}\{\vartheta_1^{\sim}, \vartheta_2^{\sim}\}. & \end{array}$$

□

Example 4.3

1. *Let $\vartheta_1 = \{x/f(y, a), z/g(b)\}$ and $\vartheta_2 = \{x/f(b, w), z/g(y)\}$. Then*

$$\begin{array}{rl} \vartheta_1 \uparrow \vartheta_2 = & \mathit{mgu}\{x = f(y, a), z = g(b), x = f(b, w), z = g(y)\} \\ = & \{\{x/f(b, a), z/g(b), y/b, w/a\}\}. \end{array}$$

2. Let ϑ_1 be the same as before, and $\vartheta_2 = \{x/f(a,w), z/g(y)\}$. Then

$$\begin{aligned}\vartheta_1 \uparrow \vartheta_2 &= mgu\{x = f(y,a), z = g(b), x = f(a,w), z = g(y)\}\\ &= \emptyset.\end{aligned}$$

The following properties of $\uparrow$ are a consequence of proposition 4.2 and of the usual properties of lattices.

commutativity	$\vartheta_1 \uparrow \vartheta_2 = \vartheta_2 \uparrow \vartheta_1$
associativity	$\vartheta_1 \uparrow (\vartheta_2 \uparrow \vartheta_3) = (\vartheta_1 \uparrow \vartheta_2) \uparrow \vartheta_3$
idempotence	$\vartheta \uparrow \vartheta = \vartheta$
identical element	$\vartheta \uparrow \epsilon = \vartheta$
null element	$\vartheta \uparrow \tau = \tau$

5 Greatest lower bound as parallel factorization

In this section we introduce a binary operator $\downarrow$ on substitutions and we show that it corresponds to the greatest lower bound. We call this operator *parallel factorization* since it is the counterpart of the parallel composition $\uparrow$ defined in the previous section. We will give a constructive definition of $\downarrow$, i.e. via an algorithm. Actually, an algorithm to compute the greatest lower bound of equation sets, w.r.t. $\preceq$, has been already defined in [17] (cfr. the *generalization algorithm*). Their method is based on the anti-unification algorithm of Huet [12], and it works in the following way. Given two equation sets E_1, E_2

1. consider the *most general solutions* (*mgs*) L_1, L_2 of E_1, E_2
2. compute the least common anti-instances set L of L_1, L_2 (seen as sets of terms), by using the anti-unification algorithm,
3. apply a special procedure to further generalize L. The equation set L' obtained so far is called the *most specific generalization* (*msg*) of E_1, E_2, and it is proved to be the greatest lower bound of E_1, E_2 w.r.t. $\preceq$.

Thus, we could define $\downarrow$ by $\vartheta \downarrow \sigma = mgu(msg(\mathcal{E}(\vartheta), \mathcal{E}(\sigma)))$. By the results of section 3.1 this definition is correct, at least when $\leq$ coincides with $\preceq$ (i.e., when the Herbrand universe is not trivial).

We propose here an algorithm that is more direct than the one above described. The basic idea is the following: given two substitutions ϑ and σ with different bindings for the same variable, say $x/t \in \vartheta$ and $x/u \in \sigma$, we eliminate this difference by replacing t and u by x in ϑ and σ respectively. This replacement will possibly establish a link between x and other variables. For instance, starting from $\vartheta = \{x/a, y/f(a)\}$ and $\sigma = \{x/b, y/f(b)\}$, we obtain $\vartheta' = \{y/f(x)\} = glb(\vartheta, \sigma)$ by replacing a and b by x. Notice that ϑ and σ are obtainable from ϑ' by composing it with the eliminated bindings $\gamma = \{x/a\}$ and $\delta = \{x/b\}$, respectively. To formalize this, we need the following definitions.

Definition 5.1 (Context) *A* context $c[\,]$ *is a term "with a hole", defined as follows*

- $[\,]$ *is a context (the* empty context*)*
- $x \in Var$ *is a context*
- *if* $c_1[\,], \ldots, c_n[\,]$ *are context, then* $f(c_1[\,], \ldots, c_n[\,])$ *is a context.*

$c[t]$ is the term obtained by "filling the hole" of $c[\,]$ with t.

- *if* $c[\,] \equiv [\,]$*, then* $c[t] \equiv t$
- *if* $c[\,] \equiv x$*, then* $c[t] \equiv x$
- *if* $c[\,] \equiv f(c_1[\,], \ldots, c_n[\,])$*, then* $c[t] \equiv f(c_1[t], \ldots, c_n[t])$.

Example 5.2

- $c[\,] \equiv f(g(x, [\,]), a, h([\,]))$ *is a context*
- *the term* $c[b]$ *is given by* $f(g(x, b), a, h(b))$.

Definition 5.3 (Subterm replacement) *$t'[t/x]$ is the term obtained by replacing every subterm t in t' by x, i.e.*

$$t'[t/x] \equiv \begin{cases} x & \text{if } t' \equiv t \\ y & \text{if } t' \not\equiv t \text{ and } t' \equiv y \in Var \\ f(t_1[t/x], \ldots, t_n[t/x]) & \text{if } t' \not\equiv t \text{ and } t' \equiv f(t_1, \ldots, t_n) \end{cases}$$

When replacing subterms we must take care to do it only when necessary. For instance, if $\vartheta = \{x/a, y/f(a,a)\}$ and $\sigma = \{x/b, y/f(a,b)\}$, then the occurrences of a in ϑ corresponding to occurrences of a in σ must not be replaced by x. Indeed, since $glb(\vartheta, \sigma) = \{y/f(a,x)\}$, the first occurrence of a in $f(a,a)$ (in ϑ) must be left unchanged. To this pourpose, we must compare binding to binding ϑ and σ when performing the replacement. To formalize this, we introduce the following definition.

Definition 5.4 (Comparative subterms replacement)

- *For $t \not\equiv u$, the pair of terms $\langle t', u' \rangle[t/x, u/x]$ is defined as*

$$\langle t', u' \rangle[t/x, u/x] = \begin{cases} \langle x, x \rangle & \text{if } t' \equiv t \text{ and } u' \equiv u \\ \langle y, y \rangle & \text{if } t' \equiv u' \equiv y \in Var \\ \langle f(t'_1, \ldots, t'_n), f(u'_1, \ldots, u'_n) \rangle & \text{if } t' \equiv f(t_1, \ldots, t_n) \text{and } u' \equiv f(u_1, \ldots, u_n), \\ & \text{where } \langle t'_i, u'_i \rangle = \langle t_i, u_i \rangle[t/x, u/x] \\ \langle t'[t/x], u'[u/x] \rangle & \text{otherwise} \end{cases}$$

- *The pair of substitutions $\langle \vartheta, \sigma \rangle[t/x, u/x]$ is defined as the pair (of substitutions) $\langle \vartheta', \sigma' \rangle$ such that for all $y \in Var$, $\langle y\vartheta', y\sigma' \rangle = \langle y\vartheta, y\sigma \rangle[t/x, u/x]$*

Example 5.5 *Let $t \equiv a, u \equiv b, t' \equiv f(a,a), u' \equiv f(a,b)$ and $u'' \equiv f(b,b)$. We have:*

- $t'[t/x] \equiv f(x,x)$, $u'[u/x] \equiv f(a,x)$ *and* $u''[u/x] \equiv f(x,x)$,
- $\langle t', u' \rangle[t/x, u/x] \equiv \langle f(a,x), f(a,x) \rangle$ *and* $\langle t', u'' \rangle[t/x, u/x] \equiv \langle f(x,x), f(x,x) \rangle$
- *Let* $\vartheta = \{x/a, y/f(a,a)\} = \{x/t, y/t'\}$ *and* $\sigma = \{x/b, y/f(a,b)\} = \{x/u, y/u'\}$. *We have* $\langle \vartheta, \sigma \rangle[t/x, u/x] = \langle \{y/f(a,x)\}, \{y/f(a,x)\} \rangle$.

We can describe now the factorization algorithm, that will be the basis for the definition of the parallel factorization operator. Essentially, this algorithm consists of repeatingly eliminating the different bindings for the same variables, until no differences are left. This elimination is done by inverting the bindings (step (**a**)) (when they are equivalent) and by comparative subterm replacement (step (**b**)) (when they are not equivalent). The eliminated bindings will constitute, at the end, the *side* substitutions, i.e. the substitutions that, composed with the result, give the original substitutions. Note that in step (**b**) the subterms choosen for the replacement must contain no subterms that could be replaced as well. For instance, if $\vartheta = \{x/a, y/f(a)\}$, and $\sigma = \{x/b, y/f(b)\}$, the first subterms to be replaced are a and b and not $f(a)$ and $f(b)$ (otherwise we would obtain, after the first step, $\vartheta' = \{x/a\}, \sigma' = \{x/b\}$, and, after the second, $\vartheta'' = \sigma'' = \epsilon$).

Definition 5.6 (Factorization algorithm)

- *Let $\langle\vartheta, \sigma\rangle$ be a pair of idempotent substitutions. Define $\langle\vartheta, \sigma\rangle \stackrel{\langle\gamma_1,\delta_1\rangle}{\longrightarrow} \langle\vartheta_1, \sigma_1\rangle$ if $\vartheta_1, \sigma_1, \gamma_1, \delta_1$ are obtained by choosing one of the following alternatives*

(**a**) $x\vartheta \equiv y, y\sigma \equiv x$ *and* $x \not\equiv y$	$\langle\gamma_1, \delta_1\rangle := \langle\epsilon, \{x/y, y/x\}\rangle$, $\langle\vartheta_1, \sigma_1\rangle := \langle\vartheta, \sigma\delta_1\rangle$
(**b**) $x\vartheta \equiv t, x\sigma \equiv u$ *and* $t \not\equiv u$ *where* $(\not\exists y \exists c_1[\,] \not\equiv [\,], c_2[\,] \not\equiv [\,]:$ $t \equiv c_1[y\vartheta] \wedge u \equiv c_2[y\sigma] \wedge y\vartheta \not\equiv y\sigma)$ *and previous case does not apply*	$\langle\gamma_1, \delta_1\rangle := \langle\{x/t\}, \{x/u\}\rangle$, $\langle\vartheta_1, \sigma_1\rangle := \langle\vartheta, \sigma\rangle[t/x, u/x]$

- *Let $\stackrel{\langle\gamma,\delta\rangle}{\longrightarrow}{}^{*}$ be the reflexive-transitive closure of the relation $\stackrel{\langle\gamma_1,\delta_1\rangle}{\longrightarrow}$, i.e.*

$$\begin{aligned}\langle\vartheta, \sigma\rangle \stackrel{\langle\gamma,\delta\rangle}{\longrightarrow}{}^{*} \langle\vartheta', \sigma'\rangle \text{ iff } & \exists\langle\vartheta_0, \sigma_0\rangle, \ldots, \langle\vartheta_n, \sigma_n\rangle \exists\langle\gamma_1, \delta_1\rangle, \ldots, \langle\gamma_n, \delta_n\rangle : \\ & \langle\vartheta, \sigma\rangle = \langle\vartheta_0, \sigma_0\rangle \stackrel{\langle\gamma_1,\delta_1\rangle}{\longrightarrow} \ldots \stackrel{\langle\gamma_n,\delta_n\rangle}{\longrightarrow} \langle\vartheta_n, \sigma_n\rangle = \langle\vartheta', \sigma'\rangle \\ & \text{and } \gamma = \gamma_n \ldots \gamma_1, \delta = \delta_n \ldots \delta_1\end{aligned}$$

  Note the reverse order of the composition in $\gamma_n \ldots \gamma_1$ and in $\delta_n \ldots \delta_1$.

The following results show the termination of the factorization algorithm and a property of the normal form.

Lemma 5.7 *Let ϑ, σ be idempotent and*

$$\langle\vartheta, \sigma\rangle = \langle\vartheta_0, \sigma_0\rangle \stackrel{\langle\gamma_1,\delta_1\rangle}{\longrightarrow} \ldots \stackrel{\langle\gamma_n,\delta_n\rangle}{\longrightarrow} \langle\vartheta_n, \sigma_n\rangle = \langle\vartheta', \sigma'\rangle.$$

Then

1. *$\vartheta_1, \sigma_1, \ldots, \vartheta_n, \sigma_n$ are substitutions (i.e. functions from Var to Term with finite domain)*
2. *$\vartheta_1, \sigma_1, \ldots, \vartheta_n, \sigma_n$ are idempotent*
3. *$\gamma_1, \delta_1, \ldots, \gamma_n, \delta_n$ are substitutions.*

Proof We show that these properties hold after the first step $\langle\vartheta, \sigma\rangle \stackrel{\langle\gamma_1,\delta_1\rangle}{\longrightarrow} \langle\vartheta_1, \sigma_1\rangle$. The rest follows by induction.

1. Obvious.

2. **step (a))** Assume $\{x/y\} \in \vartheta$ and $\{y/x\} \in \sigma$. Then $\gamma_1 = \epsilon, \vartheta_1 = \vartheta, \delta_1 = \{x/y, y/x\}$ and $\sigma_1 = \sigma\delta_1$. We show that σ_1 is idempotent.
   - $x\sigma_1\sigma_1 \equiv x\sigma\delta_1\sigma\delta_1 \equiv x\delta_1\sigma\delta_1 \equiv y\sigma\delta_1 \equiv x\delta_1 \equiv x\sigma\delta_1 \equiv x\sigma_1$.
   - $y\sigma_1\sigma_1 \equiv y\sigma\delta_1\sigma\delta_1 \equiv x\delta_1\sigma\delta_1 \equiv y\sigma\delta_1 \equiv y\sigma_1$.
   - Let $z/t \in \sigma$, where $z \not\equiv x$ and $z \not\equiv y$. Since, by idempotence of σ, t does not contain occurrences of y, we have $t\delta_1 = t[x/y]$ and $t[x/y]\sigma = t$. Then $z\sigma_1\sigma_1 \equiv z\sigma\delta_1\sigma\delta_1 \equiv t\delta_1\sigma\delta_1 \equiv t[x/y]\sigma\delta_1 \equiv t\delta_1 \equiv t\sigma\delta_1 \equiv z\sigma\delta_1 \equiv z\sigma_1$.

   step (b)) Assume $\langle\vartheta_1, \sigma_1\rangle = \langle\vartheta, \sigma\rangle[t/x, u/x]$. We show that ϑ_1 is idempotent, for σ_1 the proof is similar. We distinguish two cases

   ***t* $\not\in$ *Var*)** In this case we observe that, by definition $ran(\vartheta_1) \subseteq ran(\vartheta) \cup \{x\}$ and $dom(\vartheta_1) = dom(\vartheta) - \{x\}$. By idempotency of ϑ, we have $ran(\vartheta) \cap dom(\vartheta) = \emptyset$. Therefore $ran(\vartheta_1) \cap dom(\vartheta_1) = \emptyset$.

   ***t* $\equiv$ *y* $\in$ *Var*)** We have two subcases

   ***y* $\not\in$ *dom*(σ))** Also in this case we have, as before, $ran(\vartheta_1) \subseteq ran(\vartheta) \cup \{x\}$ and (by idempotency of ϑ) $dom(\vartheta_1) = dom(\vartheta) - \{x\}$.

   ***y* $\in$ *dom*(σ))** In this case we have (by idempotency of σ) $ran(\vartheta_1) \subseteq (ran(\vartheta) - \{y\}) \cup \{x\}$ and $dom(\vartheta_1) = (dom(\vartheta) \cup \{y\}) - \{x\}$. Therefore $ran(\vartheta_1) \cap dom(\vartheta_1) = \emptyset$ still holds.

3. Obvious. □

Proposition 5.8 *Let ϑ, σ be two idempotent substitutions. Then*

1. *the factorization algorithm always terminates, i.e. there exist $\vartheta', \sigma', \gamma, \delta$ such that*

$$\langle\vartheta, \sigma\rangle \stackrel{\langle\gamma,\delta\rangle}{\longrightarrow}{}^{*} \langle\vartheta', \sigma'\rangle, \text{ and}$$

$$\langle\vartheta', \sigma'\rangle \stackrel{\langle\gamma',\delta'\rangle}{\not\longrightarrow} \text{ for no } \gamma', \delta'.$$

   In other words, the relation $\stackrel{\langle\gamma,\delta\rangle}{\longrightarrow}$ *is* noetherian. *We say that $\langle\vartheta', \sigma'\rangle$ is in* normal form *and that it is the normal form of $\langle\vartheta, \sigma\rangle$.*

2. *$\langle\vartheta, \sigma\rangle$ is in normal form iff $\vartheta = \sigma$.*

Proof

1. Step (**a**) can be performed only a finite number of times, in fact
   - for every pair of variables x, y, step (**a**) can be performed at most once,
   - the algorithm does not introduces new variables, and
   - $dom(\vartheta) \cup ran(\vartheta) \cup dom(\sigma) \cup ran(\sigma)$ is finite.

   Also step (**b**) can be performed only a finite number of times, indeed
   - if $\langle\vartheta, \sigma\rangle \stackrel{\langle\gamma,\delta\rangle}{\longrightarrow} \langle\vartheta', \sigma'\rangle$ by step (**b**), then $\vartheta' \cup \sigma'$ contains strictly less elements than $\vartheta \cup \sigma$.
   - $\vartheta \cup \sigma$ is finite.

2. **if)** If $\vartheta = \sigma$, then by lemma 5.7 (1 and 2), step (**a**) and (**b**) does not apply.

   only if) If step (**a**) and step (**b**) do not apply, then for every variable x we must have $x\vartheta = x\sigma$. Therefore $\vartheta = \sigma$. □

We can now define the operation $\vartheta^{\sim} \downarrow \sigma^{\sim}$. We consider the pair $\langle \vartheta, \sigma \rangle$ and we repeatedly apply the algorithm 5.6 until we get a pair in which the left and the right components are the same substitution. The result of $\vartheta^{\sim} \downarrow \sigma^{\sim}$ is defined as the set of these substitutions. Notice that we obtain different substitutions depending upon the different choices of bindings in steps **(a)** and **(b)** of definition 5.6.

Definition 5.9 (Parallel factorization) *We define the operator* $\downarrow$: $ISubst_{/\sim} \times ISubst_{/\sim} \to ISubst_{/\sim}$ *as follows*

$$\vartheta^{\sim} \downarrow \sigma^{\sim} = \{\eta \mid \langle \vartheta, \sigma \rangle \stackrel{\langle \gamma, \delta \rangle}{\longrightarrow}{}^{*} \langle \eta, \eta \rangle\}$$

The following theorem shows that $\vartheta^{\sim} \downarrow \sigma^{\sim}$ is the greatest lower bound of $\vartheta^{\sim}$ and $\sigma^{\sim}$, and that γ and δ are exactly the substitutions that we have to apply to $\vartheta^{\sim} \downarrow \sigma^{\sim}$ in order to get back ϑ and σ. As in the case of $\uparrow$, the definition of $\downarrow$ is correct (i.e. it does not depends upon the choice of ϑ and σ in $\vartheta^{\sim}$ and in $\sigma^{\sim}$). The proof is an immediate consequence of next theorem (part 2). For the sake of simplicity we will therefore denote $\vartheta^{\sim} \downarrow \sigma^{\sim}$ by $\vartheta \downarrow \sigma$.

Theorem 5.10

1. *If* $\langle \vartheta, \sigma \rangle \stackrel{\langle \gamma, \delta \rangle}{\longrightarrow}{}^{*} \langle \eta, \eta \rangle$, *then* $\eta\gamma = \vartheta$ *and* $\eta\delta = \sigma$. *We call* ϑ *and* σ side substitutions.
2. $\vartheta \downarrow \sigma = glb\{\vartheta^{\sim}, \sigma^{\sim}\}$.

Proof

1. We prove that if $\langle \vartheta, \sigma \rangle \stackrel{\langle \gamma_1, \delta_1 \rangle}{\longrightarrow} \langle \vartheta_1, \sigma_1 \rangle$, then $\vartheta_1\gamma_1 = \vartheta$ and $\sigma_1\delta_1 = \sigma$. The rest follows by induction.

   step (a)) Assume $\langle \gamma_1, \delta_1 \rangle = \langle \epsilon, \{x/y, y/x\} \rangle$, and $\langle \vartheta_1, \sigma_1 \rangle = \langle \vartheta, \sigma\delta_1 \rangle$. Then we have $\vartheta_1\gamma_1 = \vartheta_1 = \vartheta$ and, since $\delta_1\delta_1 = \epsilon$, $\sigma_1\delta_1 = \sigma\delta_1\delta_1 = \sigma$.

   step (b)) Assume $\langle \vartheta_1, \sigma_1 \rangle = \langle \vartheta, \sigma \rangle[t/x, u/x]$, and $\gamma_1 = \{x/t\}, \delta_1 = \{x/u\}$. We prove that for all $y \in vars$, $y\vartheta_1\{x/t\} \equiv y\vartheta$ holds. The proof for σ_1 is similar. We have four cases, given by the definition of $\langle y\vartheta, y\sigma \rangle[t/x, u/x]$:

   $y\vartheta \equiv t, y\sigma \equiv u$) In this case, $y\vartheta_1 \equiv x$ and then $y\vartheta_1\{x/t\} \equiv t$.

   $y\vartheta \equiv y\sigma$) In this case, $y\vartheta_1 \equiv y\vartheta$ and then, by idempotency of ϑ, $y\vartheta_1\{x/t\} \equiv y\vartheta\{x/t\} \equiv y\vartheta$ holds.

   $y\vartheta \equiv f(t_1, \ldots, t_n), y\sigma \equiv f(u_1, \ldots, u_n)$) In this case, $y\vartheta_1 \equiv f(t_1\vartheta_1, \ldots, t_n\vartheta_1)$ and then, by structural induction,

   $$\begin{array}{ll} y\vartheta_1\{x/t\} & \equiv \\ f(t_1\vartheta_1, \ldots, t_n\vartheta_1)\{x/t\} & \equiv \\ f(t_1\vartheta_1\{x/t\}, \ldots, t_n\vartheta_1\{x/t\}) & \equiv \\ f(t_1\vartheta, \ldots, t_n\vartheta) & \equiv \\ y\vartheta & \end{array}$$

   all the previous cases do not apply) We have $y\vartheta_1 \equiv y\vartheta[t/x]$. By idempotency of ϑ, x does not occur in $y\vartheta$. Therefore $y\vartheta_1\{x/t\} \equiv y\vartheta[t/x]\{x/t\} \equiv y\vartheta$.

2. By part 1 of this theorem we have that $\vartheta \downarrow \sigma$ is a lower bound of $\{\vartheta^{\sim}, \sigma^{\sim}\}$. We have just to prove that it is the greatest one. We show that for any lower bound $\psi^{\sim}$, and for $\langle \vartheta, \sigma \rangle \stackrel{\langle \gamma_1, \delta_1 \rangle}{\longrightarrow} \langle \vartheta_1, \sigma_1 \rangle$, we have $\psi \leq \vartheta_1$ and $\psi \leq \sigma_1$. The rest follows by induction. Assume that, for appropriate ψ', ψ'' we have $\psi\psi' = \vartheta$ and $\psi\psi'' = \sigma$.

step (a)) Assume $\{x/y\} \in \vartheta$ and $\{y/x\} \in \sigma$. Then $\gamma_1 = \epsilon, \vartheta_1 = \vartheta = \vartheta_1\gamma_1, \delta_1 = \{x/y, y/x\}$ and $\sigma_1 = \sigma\delta_1 \sim \sigma$ (since δ_1 is a renaming). The rest follows by the hypothesis.

step (b)) Assume $\langle \vartheta_1 \sigma_1 \rangle = \langle \vartheta, \sigma \rangle[t/x, u/x] = \langle \psi\psi', \psi\psi'' \rangle[t/x, u/x]$. We prove that for $\langle \eta', \eta'' \rangle = \langle \psi', \psi'' \rangle[t/x, u/x]$ we have

$$\psi\eta' = \vartheta_1 \text{ and } \psi\eta'' = \sigma_1 \tag{1}$$

- First we prove that, for $\langle \vartheta_1, \sigma_1 \rangle = \langle \vartheta, \sigma \rangle[t/x, u/x]$, we have

$$\forall t' \in \mathit{Term}. \langle t'\vartheta_1, t'\sigma_1 \rangle = \langle t'\vartheta, t'\sigma \rangle[t/x, u/x] \tag{2}$$

  By induction.

  $\boldsymbol{t' \equiv z \in \mathit{Var}}$) We have

$$\begin{array}{ll} \langle t'\vartheta_1, t'\sigma_1 \rangle & = \\ \langle z\vartheta_1, z\sigma_1 \rangle & = \text{(by definition)} \\ \langle z\vartheta, z\sigma \rangle[t/x, u/x] & = \\ \langle t'\vartheta, t'\sigma \rangle[t/x, u/x] & \end{array}$$

  $\boldsymbol{t' \equiv f(t_1, \ldots, t_n)}$) We have

$$\begin{array}{ll} \langle t'\vartheta_1, t'\sigma_1 \rangle & = \\ \langle f(t_1, \ldots, t_n)\vartheta_1, f(t_1, \ldots, t_n)\sigma_1 \rangle & = \\ \langle f(t_1\vartheta_1, \ldots, t_n\vartheta_1), f(t_1\sigma_1, \ldots, t_n\sigma_1) \rangle & \end{array}$$

  Now consider $\langle t'\vartheta, t'\sigma \rangle[t/x, u/x]$. We have

$$\begin{array}{ll} \langle t'\vartheta, t'\sigma \rangle[t/x, u/x] & = \\ \langle f(t_1, \ldots, t_n)\vartheta, f(t_1, \ldots, t_n)\sigma \rangle[t/x, u/x] & = \\ \langle f(t_1\vartheta, \ldots, t_n\vartheta), f(t_1\sigma, \ldots, t_n\sigma) \rangle[t/x, u/x] & \end{array}$$

  We observe that $f(t_1\vartheta, \ldots, t_n\vartheta) \equiv t$ and $f(t_1\sigma, \ldots, t_n\sigma)\rangle \equiv u$ imply $t \equiv c[w\vartheta], u \equiv c[w\sigma]$, for an appropriate variable w and a non-empty context $c[\,]$. Therefore one of the two identities above does not hold (otherwise we would have applied step (**b**) with respect to w/t' and w/u'). Hence we have:

$$\langle t'\vartheta, t'\sigma \rangle[t/x, u/x] = \langle f(t'_1, \ldots, t'_n), f(u'_1, \ldots, u'_n) \rangle$$

  where

$$\begin{array}{ll} \langle t'_i, u'_i \rangle & = \\ \langle t_i\vartheta, t_i\sigma \rangle[t/x, u/x] & = \text{(by inductive hypothesis)} \\ \langle t_i\vartheta_1, t_i\sigma_1 \rangle & \end{array}$$

  Thus, we have

$$\langle t'\vartheta, t'\sigma \rangle[t/x, u/x] = \langle t'\vartheta_1, t'\sigma_1 \rangle$$

- Now we prove that

$$\forall t' \in \mathit{Term}. \langle t'\psi\vartheta_1, t'\psi\sigma_1 \rangle = \langle t'\psi\eta', t'\psi\eta'' \rangle \tag{3}$$

  By structural induction.

  $\boldsymbol{t' \equiv z \in \mathit{Var}}$)

$$\begin{array}{ll} \langle t'\psi\vartheta_1, t'\psi\sigma_1 \rangle & = \\ \langle z\psi\vartheta_1, z\psi\sigma_1 \rangle & = \text{(by (2))} \\ \langle z\psi\vartheta, z\psi\sigma \rangle[t/x, u/x] & = \\ \langle z\psi\psi\psi', z\psi\psi\psi'' \rangle[t/x, u/x] & = \text{(by idempotence of } \psi) \\ \langle z\psi\psi', z\psi\psi'' \rangle[t/x, u/x] & = \\ \langle z\psi\eta', z\psi\eta'' \rangle & = \\ \langle t'\psi\eta', t'\psi\eta'' \rangle & \end{array}$$

$\boldsymbol{t' \equiv f(t_1, \ldots, t_n))}$

$$\begin{array}{lcl} \langle t'\psi\vartheta_1, t'\psi\sigma_1\rangle & = & \\ \langle f(t_1,\ldots,t_n)\psi\vartheta_1, f(t_1,\ldots,t_n)\psi\sigma_1\rangle & = & \\ \langle f(t_1\psi\vartheta_1,\ldots,t_n\psi\vartheta_1), f(t_1\psi\sigma_1,\ldots,t_n\psi\sigma_1)\rangle & = & \text{(by inductive hypothesis)} \\ \langle f(t_1\psi\eta',\ldots,t_n\psi\eta'), f(t_1\psi\eta'',\ldots,t_n\psi\eta'')\rangle & = & \\ \langle f(t_1,\ldots,t_n)\psi\eta', f(t_1,\ldots,t_n)\psi\eta''\rangle & = & \\ \langle t'\psi\eta', t'\psi\eta''\rangle & & \end{array}$$

- Finally, we can prove (1). Consider $y \in Var$. We have

$$\begin{array}{lcl} \langle y\psi\eta', y\psi\eta''\rangle & = & \text{(by (3))} \\ \langle y\psi\vartheta_1, y\psi\sigma_1\rangle & = & \text{(by (2))} \\ \langle y\psi\vartheta, y\psi\sigma\rangle[t/x, u/x] & = & \\ \langle y\psi\psi\psi', y\psi\psi\psi''\rangle[t/x, u/x] & = & \text{(by idempotence of } \psi) \\ \langle y\psi\psi', y\psi\psi''\rangle[t/x, u/x] & = & \\ \langle y\vartheta, y\sigma\rangle[t/x, u/x] & = & \text{(by definition)} \\ \langle y\vartheta_1, y\sigma_1\rangle & & \end{array}$$

□

Example 5.11 *Let $\vartheta = \{x/a, x'/a, y/f(a,a), z/w\}$ and $\sigma = \{x/b, x'/b, y/f(a,b), w/z\}$. Consider the application of the factorization algorithm to $\langle\vartheta, \sigma\rangle$. Initially we have to apply step* (**a**) *and we obtain*

- $\langle\vartheta_1, \sigma_1\rangle = \langle\{x/a, x'/a, y/f(a,a), z/w\}, \{x/b, x'/b, y/f(a,b), z/w\}\rangle$ *and*
- $\langle\gamma_1, \delta_1\rangle = \langle\epsilon, \{w/z, z/w\}\rangle$.

Now we can apply only step (**b**), *and we consider the elements x/a and x/b. We obtain then*

- $\langle\vartheta_2, \sigma_2\rangle = \langle\{x'/x, y/f(a,x), z/w\}, \{x'/x, y/f(a,x), z/w\}\rangle$ *and*
- $\langle\gamma_2, \delta_2\rangle = \langle\{x/a\}, \{x/b\}\rangle$.

Since $\vartheta_2 = \sigma_2$, then $\langle\vartheta_2, \sigma_2\rangle$ is in normal form, and therefore the algorithm stops. We have $\vartheta = \vartheta_2\gamma_2\gamma_1$ and $\sigma = \sigma_2\delta_2\delta_1$.

Notice that at the second step of example 5.11 we could consider the elements x'/a and x'/b. In this case we obtain

$$\langle\vartheta'_2, \sigma'_2\rangle = \langle\{x/x', y/f(a,x'), z/w\}, \{x/x', y/f(a,x'), z/w\}\rangle$$

On the contrary, we cannot consider $y/f(a,a)$ and $y/f(a,b)$, since $f(a,a)$ and $f(a,b)$ are strictly greater than a and b. Note that $\vartheta'_2 \sim \vartheta_2$ and $\sigma'_2 \sim \sigma_2$. In general, different choices of elements in steps (**a**) and (**b**) lead to different (but, of course, equivalent) least upper bounds.

The following properties are a consequence of theorem 5.10 (2) and the usual properties of lattices.

commutativity	$\vartheta_1 \downarrow \vartheta_2 = \vartheta_2 \downarrow \vartheta_1$
associativity	$\vartheta_1 \downarrow (\vartheta_2 \downarrow \vartheta_3) = (\vartheta_1 \downarrow \vartheta_2) \downarrow \vartheta_3$
idempotence	$\vartheta \downarrow \vartheta = \vartheta$
identical element	$\vartheta \downarrow \tau = \vartheta$
null element	$\vartheta \downarrow \epsilon = \epsilon$

Moreover, from theorem 5.10, proposition 4.2, and the usual properties of lattices, we get the following relations between $\uparrow$ and $\downarrow$

weak distributivity	$\vartheta_1 \uparrow (\vartheta_2 \downarrow \vartheta_3) \leq (\vartheta_1 \uparrow \vartheta_2) \downarrow (\vartheta_1 \uparrow \vartheta_3)$
	$(\vartheta_1 \downarrow \vartheta_2) \uparrow (\vartheta_1 \downarrow \vartheta_3) \leq \vartheta_1 \downarrow (\vartheta_2 \uparrow \vartheta_3)$
absorption	$\vartheta_1 \uparrow (\vartheta_2 \downarrow \vartheta_1) = \vartheta_1$
	$\vartheta_1 \downarrow (\vartheta_2 \uparrow \vartheta_3) = \vartheta_1$

6 Applications

In this section we discuss some applications of our theory.

6.1 Semantics of HCL

We show how the notions introduced in the previous sections can be usefully exploited in the semantics of logic programming. Let first briefly recall some basic notions about HCL. We refer to [1], [18] for a detailed introduction to it. Let *Pred* be a set of *predicates*, with typical elements $p, q, r, \ldots$. The set *Atom*, with typical elements $A, B, H, G \ldots$, is the set of all the *atoms* (or *atomic formulas*) built on *Pred* and on *Term*. Examples of atoms are $p(a), p(x), q(f(a), f(x))$ etc. A *definite clause* C is a formula of the form

$$A \leftarrow B_1, \ldots, B_n \quad (n \geq 0)$$

where the intended meaning of "$\leftarrow$" is the right-to-left logic implication, and the intended meaning of "," is the logical conjunction. Clauses are seen as universally quantified. A *goal statement* G is a formula of the form

$$\leftarrow A_1, \ldots, A_m \quad (m \geq 0)$$

(the missing argument at the left of $\leftarrow$ is meant to be the logical value *false*). If $m = 0$, the goal is called *empty goal* and is denoted by $\Box$. Goal statements are also seen as universally quantified. A HCL program W is a (finite) set of definite clauses. Since a clause (as well as a goal statement) is a particular First Order Logic formula, HCL programs can be seen as a particular subset of the First Order Logic theories. In the following, we use the abbreviations $\bar{A}, \bar{B}$ etc. to denote conjunctions of atoms, like $A_1, \ldots, A_m$ and $B_1, \ldots, B_n$. For instance, $H \leftarrow \bar{B}$ represents a clause, and $\leftarrow \bar{A}$ represents a goal statement.

The *resolution step* is defined by

$$\leftarrow \bar{A}', A, \bar{A}'' \overset{\vartheta}{\longmapsto} \leftarrow (\bar{A}', \bar{B}, \bar{A}'')\vartheta \text{ iff } \exists H \leftarrow \bar{B} \in W \text{ (renamed)} : \vartheta \in mgu(A = H).$$

The renaming of the clauses is done in order to avoid variable clash with the goal.

A *derivation* is defined by

$$G \overset{\vartheta}{\longmapsto}{}^{*} G' \text{ iff } \exists \vartheta_1, \ldots, \vartheta_n : G \overset{\vartheta_1}{\longmapsto} \ldots \overset{\vartheta_n}{\longmapsto} G' \text{ and } \vartheta = \vartheta_1 \ldots \vartheta_n.$$

The operational semantics of a goal G in a program W is defined by the set (*not ground success set*)

$$\mathcal{O}(G) = \{\vartheta \mid G \overset{\vartheta}{\longmapsto}{}^{*} \Box\}$$

6.1.1 Compositionality

We give now a compositional characterization of the operational semantics of HCL, following [5]. We define a transition relation from goals to goals, labeled on substitutions, via the following axiom and rule:

$$\textbf{(1)}\quad \leftarrow A \xrightarrow{\vartheta} \leftarrow \bar{B} \text{ iff } \exists H \leftarrow \bar{B} \in W(\text{appropriately renamed}) : \vartheta \in mgu(A = H).$$

$$\textbf{(2)}\quad \frac{\leftarrow \bar{A}_1 \xrightarrow{\vartheta_1} \bar{B}_1, \quad \leftarrow \bar{A}_2 \xrightarrow{\vartheta_2} \bar{B}_2}{\leftarrow \bar{A}_1, \bar{A}_2 \xrightarrow{\vartheta_1 \uparrow \vartheta_2} \leftarrow (\bar{B}_1, \bar{B}_2)(\vartheta_1 \uparrow \vartheta_2)}$$

The (compositional) operational semantics of HCL can now be defined by:

$$\mathcal{O}'(G) = \{\vartheta_1 \uparrow \ldots \uparrow \vartheta_n \mid G \xrightarrow{\vartheta_1} \ldots \xrightarrow{\vartheta_n} \Box\}$$

This semantics is compositional. We can indeed extend the operator $\uparrow$ to sets of substitutions in the following way. Given $\Theta_1, \Theta_2 \subseteq \mathit{Subst}$ we define:

$$\Theta_1 \uparrow \Theta_2 = \begin{cases} \bigcup_{\vartheta_1 \in \Theta_1, \vartheta_2 \in \Theta_2} \vartheta_1 \uparrow \vartheta_2 & \text{if it is different from } \{\tau\} \\ \emptyset & \text{otherwise} \end{cases}$$

We have then

$$\mathcal{O}'(\leftarrow \bar{A}_1, \bar{A}_2) = \mathcal{O}'(\leftarrow \bar{A}_1) \uparrow \mathcal{O}'(\leftarrow \bar{A}_2) \tag{4}$$

For the proof of (4) we refer [5].

We now prove that this new semantics exactly corresponds to the standard one. For a different proof, more complicate but much more general, because it deals also with the finite failure and the infinite failure sets, we refer again [5]. Observe that in the definition of $\mathcal{O}'$ we don't make use of the standard notion of composition. This justifies the assertion, made in the introduction, that we can redefine the whole theory of logic programming without this operator.

Proposition 6.1 *Let ϑ_1, ϑ_2 be idempotent substitutions. Then*

$$\vartheta_1 \uparrow \vartheta_2 = \vartheta_1 mgu(\mathcal{E}(\vartheta_2)\vartheta_1) = \vartheta_2 mgu(\mathcal{E}(\vartheta_1)\vartheta_2).$$

Proof It is sufficient to prove that $\vartheta_1 \uparrow \vartheta_2 = \vartheta_1 mgu(\mathcal{E}(\vartheta_2)\vartheta_1)$ (the other equation is symmetrical). We prove that for all $\mu \in mgu(\mathcal{E}(\vartheta_2)\vartheta_1)$ we have $\mathit{unif}(\mathcal{E}(\vartheta_1 \uparrow \vartheta_2)) = \mathit{unif}(\mathcal{E}(\vartheta_1\mu))$. The rest follows by proposition 3.7.

$\subseteq$) Let $\vartheta \in \mathit{unif}(\mathcal{E}(\vartheta_1 \uparrow \vartheta_2))$. Then $\vartheta \in \mathit{unif}(\mathcal{E}(\vartheta_1))$ and $\vartheta \in \mathit{unif}(\mathcal{E}(\vartheta_2))$. Then for each $x \in \mathit{Var}$

$$\begin{aligned} x\vartheta_2\vartheta_1\vartheta &= \text{(by proposition 3.4)} \\ x\vartheta_2\vartheta &= \text{(by proposition 3.4)} \\ x\vartheta & \end{aligned}$$

Therefore $\vartheta \in \mathit{unif}(\mathcal{E}(\vartheta_2)\vartheta_1))$. Then

$$\begin{aligned} \vartheta_1\mu\vartheta &= \text{(by propositions 3.7 and 3.4)} \\ \vartheta_1\vartheta &= \text{(by proposition 3.4)} \\ \vartheta & \end{aligned}$$

Again by proposition 3.4 we have that $\vartheta \in \mathit{unif}(\mathcal{E}(\vartheta_1\mu))$.

$\supseteq$) Let $\vartheta \in \mathit{unif}(\mathcal{E}(\vartheta_1\mu))$. Then by proposition 3.4, $\vartheta_1\mu\vartheta = \vartheta$. Moreover $\vartheta_2\vartheta = \vartheta_2\vartheta_1\mu\vartheta = \vartheta_1\mu\vartheta = \vartheta$. By proposition 3.7 we have that $\vartheta \in \mathit{unif}(\mathcal{E}(\vartheta_1))$ and $\vartheta \in \mathit{unif}(\mathcal{E}(\vartheta_2))$. Therefore $\vartheta \in \mathit{unif}(\mathcal{E}(\vartheta_1 \uparrow \vartheta_2))$. $\Box$

Note that the definition given by Eder [6] for the least upper bound is just $lub(\vartheta_1^{\sim}, \vartheta_2^{\sim}) = \vartheta_1 mgu(\mathcal{E}(\vartheta_2)\vartheta_1)$. Therefore this proposition, together with proposition 4.2, shows the correctness of his definition.

Corollary 6.2 *$G \overset{\vartheta}{\longmapsto}^* G'$ iff $G \xrightarrow{\vartheta_1} \ldots \xrightarrow{\vartheta_n} G'$ and $\vartheta \in \vartheta_1 \uparrow \ldots \uparrow \vartheta_n$*

6.1.2 Proofs

Same proofs of the standard theory of logic programming get much easier when we use the results of previous section. We give here an example.

Example 6.3 (Switching lemma) *The property stated by the switching lemma (cfr. lemma 9.1 in [18]) is the following: Assume*

$$\begin{array}{lll} \leftarrow A_1, A_2 & \overset{\vartheta'}{\longmapsto} \leftarrow (\bar{B}_1, A_2)\vartheta' & \textit{with } H_1 \leftarrow \bar{B}_1 \in W \textit{ (renamed)}, \vartheta' \in mgu(A_1 = H_1) \\ & \overset{\vartheta''}{\longmapsto} \leftarrow (\bar{B}_1, \bar{B}_2)\vartheta'\vartheta'' & \textit{with } H_2 \leftarrow \bar{B}_2 \in W \textit{ (renamed)}, \vartheta'' \in mgu(A_2\vartheta' = H_2) \end{array}$$

and

$$\begin{array}{lll} \leftarrow A_1, A_2 & \overset{\sigma'}{\longmapsto} \leftarrow (A_1, \bar{B}_2)\sigma' & \sigma' \in mgu(A_2 = H_2) \\ & \overset{\sigma''}{\longmapsto} \leftarrow (\bar{B}_1, \bar{B}_2)\sigma'\sigma'' & \sigma'' \in mgu(A_1\sigma' = H_1) \end{array}$$

Then

$$(A_1, A_2)\sigma'\sigma'' = (A_1, A_2)\vartheta'\vartheta''$$

and

$$(\bar{B}_1, \bar{B}_2)\sigma'\sigma'' = (\bar{B}_1, \bar{B}_2)\vartheta'\vartheta''$$

We have a very simple proof:
Proof

$$\begin{array}{lll} \langle A_1, A_2, \bar{B}_1, \bar{B}_2\rangle\sigma'\sigma'' & = & \\ \langle A_1, A_2, \bar{B}_1, \bar{B}_2\rangle\sigma' mgu(A_1\sigma' = H_1) & = & \textit{(by the renaming assumption)} \\ \langle A_1, A_2, \bar{B}_1, \bar{B}_2\rangle\sigma' mgu(A_1\sigma' = H_1\sigma') & = & \textit{(by proposition 6.1)} \\ \langle A_1, A_2, \bar{B}_1, \bar{B}_2\rangle\sigma' \uparrow mgu(A_1 = H_1) & = & \\ \langle A_1, A_2, \bar{B}_1, \bar{B}_2\rangle\sigma' \uparrow \vartheta' & = & \\ \langle A_1, A_2, \bar{B}_1, \bar{B}_2\rangle\vartheta' \uparrow \sigma' & = & \\ \langle A_1, A_2, \bar{B}_1, \bar{B}_2\rangle\vartheta' \uparrow mgu(A_2 = H_2) & = & \textit{(by proposition 6.1)} \\ \langle A_1, A_2, \bar{B}_1, \bar{B}_2\rangle\vartheta' mgu(A_2\vartheta' = H_2\vartheta') & = & \textit{(by the renaming assumption)} \\ \langle A_1, A_2, \bar{B}_1, \bar{B}_2\rangle\vartheta' mgu(A_2\vartheta' = H_2) & = & \\ \langle A_1, A_2, \bar{B}_1, \bar{B}_2\rangle\vartheta'\vartheta'' & & \end{array}$$

6.2 Parallel models for logic programming and concurrent logic programming

The considerations made in the previous section about the compositional semantics of HCL apply also to the concurrent logic programming. In particular, relation 4 provides a definition of the combination of the sets of results Θ_1, Θ_2 computed by the two subgoals, $\leftarrow A_1$, $\leftarrow A_2$,of the goal $\leftarrow A_1, A_2$, run in parallel. We can define such a result as

$$\Theta_1 \uparrow \Theta_2$$

Note that the *consistency check* corresponds to check that $\vartheta_1 \uparrow \vartheta_2$ ($\vartheta_1 \in \Theta_1, \vartheta_2 \in \Theta_2$) is not τ.

In the field of the parallel models for logic programming we also find an interesting application for the operator $\downarrow$. As well as $\uparrow$ is related to the AND-parallelism, so $\downarrow$ is related to the OR-parallelism. Consider, in the previous example, the OR nodes N_1 and to N_2 associated to $\leftarrow A_1$ and to $\leftarrow A_2$. Assume that one of the two, say N_1, intermediately computes some subset Θ'_1 of Θ_1 (note that the elements of Θ_1 corresponds to different OR computations). Then N_1 could compute $\downarrow(\Theta'_1)$. The father AND node N, associated to $\leftarrow A_1, A_2$, could use $\downarrow(\Theta'_1)$ to make a preliminary consistency check with the results computed by N_2: if, for $\vartheta_2 \in \Theta_2$, we have $\vartheta_2 \uparrow (\downarrow(\Theta'_1)) = \tau$ then (by the weak distributivity) it is useless to compare ϑ_2 with the single elements of Θ'_1. (A similar method to speed up the detection of inconsistencies has been proposed in [13].) If, on the other side, $\vartheta_2 \uparrow (\downarrow(\Theta'_1)) = \psi \neq \tau$, then we can use the *side substitutions* of theorem 5.10 to compute the real results. In fact, if $\Theta'_1 = \{\vartheta_1, \sigma_1\}$, and γ and δ are the side substitutions for ϑ_1 and σ_1, then, by commutativity and associativity of $\uparrow$, $\vartheta_1 \uparrow \vartheta_2 = ((\vartheta_1 \downarrow \sigma_1) \uparrow \vartheta_2) \uparrow \gamma$ and $\sigma_1 \uparrow \vartheta_2 = ((\vartheta_1 \downarrow \sigma_1) \uparrow \vartheta_2) \uparrow \delta$. Since $((\vartheta_1 \downarrow \sigma_1) \uparrow \vartheta_2)$ has to be computed only once, and γ and δ are in general less complex than ϑ_1 and σ_1, this method should speed up the computation of the result done by N.

7 Conclusions and future works

Starting from the observation that the notion of substitution is so complicate to deal with because it has been provided only with the operation of composition (that has very few properties), we have proposed an algebra of idempotent substitutions whose operations have many properties: commutativity, associativity, existence of the neutral element, idempotence, existence of the null element, weak distributivity and absorption. We have given the algorithms to compute them, and we have proved their correctness. We have shown that the operational semantics of Logic Programming could be rewritten in terms of one of these new operations: the parallel composition. We have shown that, by making use of the properties of parallel composition, and of its relation with the standard composition, some standard properties get obvious. Finally, we have suggested a possible use of the other operation, the parallel factorization, in the framework of the parallel models for logic programming. We are going to develop, in the next future, an extension and a formalization of this idea.

An other promising application of the operator $\downarrow$ seems to be related to the field of abstract interpretations. The parallel factorization, indeed, gives the *general form* of the substitutions that can be computed by a certain goal. Therefore many properties of a program, like the one that a certain argument in a goal will remain unbound, can be derived by simply looking at the parallel factorization of the substitutions considered when applying the standard techniques of abstract interpretations (both in the top-down and in the bottom-up approaches).

Finally, we are going to extend the lattice of idempotent substitutions (and its correspondence with the universally quantified equation sets) to the *existentially quantified* (idempotent) substitutions, corresponding to the existentially quantified equation sets.

8 Acknowledgements.

I am much indebted to Frank S. de Boer who has suggested me the importance of having a well structured theory for substitutions, and who has encouraged me to write this work. Discussions with Jean-Marie Jacquet, Joost N. Kok, Jan J. M. M. Rutten, Jan-Willem Klop and Krzysztof R. Apt have also been very helpful in this

sense. In particular, I want to express my gratitude to Giorgio Levi, who has helped me several times by comments, suggestions and encouragements.

References

[1] K. Apt. Introduction to logic programming (revised and extended version). Technical Report CS-R8826, Centre for Mathematics and Computer Science, Amsterdam, 1988. To appear as a chapter in *Handbook of Theoretical Computer Science*, North-Holland (J. van Leeuwen, editor).

[2] J. de Bakker and J. Kok. Uniform abstraction, atomicity and contractions in the comparative semantics of Concurrent Prolog. In *Proc. of the International Conference on Fifth Generation Computer Systems*, pages 347–355, Tokyo, Japan, 1988. OHMSHA, LTD. Extended Abstract, full version available as CWI report CS-8834. To appear on Theoretical Computer Science.

[3] F. de Boer, J. Kok, C. Palamidessi, and J. Rutten. Control flow versus logic: a denotational and a declarative model for Guarded Horn Clauses. In A. Kreczmar and G. Mirkowska, editors, *Proc. of the Symposium on Mathematical Foundations of Computer Science*, volume 379 of *Lecture Notes in Computer Science*, pages 165–176. Springer Verlag, 1989.

[4] F. de Boer, J. Kok, C. Palamidessi, and J. Rutten. Semantic models for a version of PARLOG. In G. Levi and M. Martelli, editors, *Proc. of the Sixth International Conference on Logic Programming*, pages 621–636, Lisboa, 1989. MIT Press. Extended version to appear in Theoretical Computer Science.

[5] F. de Boer, J. Kok, C. Palamidessi, and J. Rutten. From failure to success: Comparing a denotational and a declarative semantics for Horn Clause Logic. In *Proc. of the International Workshop on Semantics for Concurrency*, Lecture Notes in Computer Science. Springer Verlag, 1990.

[6] E. Eder. Properties of substitutions and unifications. *Journal of Symbolic Computation*, 1:31–46, 1985.

[7] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A new declarative semantics for logic languages. In R. A. Kowalski and K. A. Bowen, editors, *Proc. of the Fifth International Conference on Logic Programming*, Series in Logic Programming, pages 993–1005, Seattle, USA, 1988. MIT press, Cambridge, Mass.

[8] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative modeling of the operational behaviour of logic languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[9] M. Gabbrielli and G. Levi. An unfolding reactive semantics for concurrent constraint programming. Technical Report TR ../90, Dipartimento di Informatica, Università di Pisa, Pisa, 1990.

[10] H. Gaifman, M. J. Maher, and E. Shapiro. Reactive Behaviour semantics for Concurrent Constraint Logic Programs. In E. Lusk and R. Overbeck, editors, *North American Conference on Logic Programming*, 1989.

[11] R. Gerth, M. Codish, Y. Lichtenstein, and E. Shapiro. Fully abstract denotational semantics for Concurrent Prolog. In *Proc. of the Third IEEE Symposium on Logic In Computer Science*, pages 320–335. IEEE Computer Society Press, New York, 1988.

[12] G. Huet. *Resolution d'Equations dans des Langages d'Order 1, 2,...,ω*. PhD thesis, Univ. Paris VII, 1976.

[13] J.-M. Jacquet. *Conclog: a Methodological Approach to Concurrent Logic Programming*. PhD thesis, Facultés Universitaires Notre-Dame de la Paix, University of Namur, Belgium, 1989.

[14] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *14th ACM Principles of Programming Languages Conference*, pages 111–119, Munich, F.R.G., 1987. ACM, New York.

[15] J. Jaffar, J.-L. Lassez, and M. Maher. A logic programming language scheme. In D. de Groot and G. Lindstrom, editors, *Logic Programming, Functions, Relations and Equations*, pages 441–468. Prentice Hall, 1986.

[16] J. Kok. A compositional semantics for Concurrent Prolog. In R. Cori and M. Wirsing, editors, *Proc. Fifth Symposium on Theoretical Aspects of Computer Science*, volume 294 of *Lecture Notes in Computer Science*, pages 373–388. Springer Verlag, 1988.

[17] J. L. Lassez, M. J. Maher, and K. Marriot. Unification Revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgann Kaufmann, Los Altos, 1988.

[18] J. Lloyd. *Foundations of logic programming*. Springer-Verlag, second edition, 1987.

[19] M. J. Maher. Logic semantics for a class of committed-choice programs. In J.-L. Lassez, editor, *Proc. of the Fourth International Conference on Logic Programming*, Series in Logic Programming, pages 858–876, Melbourne, 1987. MIT Press.

[20] C. Palamidessi. A fixpoint semantics for Guarded Horn Clauses. Technical Report CS-R8833, Centre for Mathematics and Computer Science, Amsterdam, 1988.

[21] V. Saraswat. *Concurrent Constraint Programming Languages*. PhD thesis, january 1989. To be published by the MIT Press.

[22] V. Saraswat and M. Rinard. Concurrent constraint programming. In *Conference Record of the Annual ACM Symposium on Principles of Programming Languages*. ACM, New York, 1990.

[23] M. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.