Selective Unification in (Constraint) Logic Programming *

Frédéric Mesnard, Etienne Payet, Germán Vidal

Selective Unification in (Constraint) Logic Programming*

Fred Mesnard, Étienne Payet
LIM - Université de la Réunion, France
{frederic.mesnard,etienne.payet}@univ-reunion.fr

Germán Vidal
MiST, VRAIN, Universitat Politècnica de València, Spain
gvidal@dsic.upv.es

Abstract

Concolic testing is a well-known validation technique for imperative and object oriented programs. In a previous paper, we have introduced an adaptation of this technique to logic programming. At the heart of our framework lies a specific procedure that we call "selective unification". It is used to generate appropriate run-time goals by considering all possible ways an atom can unify with the heads of some program clauses. In this paper, we show that the existing algorithm for selective unification is not complete in the presence of non-linear atoms. We then prove soundness and completeness for a restricted version of the problem where some atoms are required to be linear. We also consider concolic testing in the context of constraint logic programming and extend the notion of selective unification accordingly.

1 Introduction

Concolic testing is a well-known validation technique for imperative and object oriented programs. Roughly speaking, concolic testing combines *conc*rete and symb*olic* execution (called *concolic* execution) in order to systematically produce test cases that aim at exploring all feasible execution paths of a program. Typically, one starts with an arbitrary concrete call, say $main(i_1, \ldots, i_n)$ and runs both concrete and symbolic execution on $main(i_1, \ldots, i_n)$ and $main(v_1, \ldots, v_n)$, respectively, where $v_1, \ldots, v_n$ are symbolic variables denoting unknown input values. In contrast to ordinary symbolic execution, the symbolic component of concolic execution does not explore all possible execution paths, but just mimics the steps of the concrete execution, gathering along the way constraints on the symbolic variables to follow

*This work has been partially supported by the EU (FEDER) and the *Spanish Ministerio de Ciencia, Innovación y Universidades*/AEI under grant TIN2016-76843-C4-1-R and by the *Generalitat Valenciana* under grant Prometeo/2019/098 (DeepTrust).

a particular path. Once a concolic execution terminates,[1] one uses the collected constraints to produce new concrete calls that will explore different paths. For instance, if the collected constraints are c_1, c_2, c_3, then by solving $\neg c_1$ we get values for the symbolic variables of $main(v_1, \ldots, v_n)$ so that the resulting concrete call will follow a different execution path. Other alternative initial calls can be obtained by solving $c_1 \wedge \neg c_2$ and $c_1 \wedge c_2 \wedge \neg c_3$.

We have introduced an adaptation of this technique to logic programming [1]. In contrast to the case of imperative or object oriented programming, computing the alternatives of a given execution is more complex in this setting. Consider for instance a predicate p/n defined by the set of clauses

$$\{H_1 \leftarrow B_1, H_2 \leftarrow B_2, H_3 \leftarrow B_3\}$$

and a goal where the selected atom, $p(t_1, \ldots, t_n)$, only unifies with H_1. What are then the possible alternatives? In principle, one could think that producing a goal where the selected atom only unifies with H_2 and another goal where the selected atom only unifies with H_3 is enough. However, there are five more possibilities: unifying with no clause, unifiying with both H_1 and H_2, unifying with both H_1 and H_3, unifying with both H_2 and H_3, and unifying with all three atoms H_1, H_2 and H_3.[2] Moreover, we found in [1] that producing goals that satisfy each of these conditions is far from trivial. This problem, that we call "selective unification", can be roughly expressed as follows: given an atom A, a set of *positive atoms* $\mathcal{H}^+$ and a set of *negative atoms* $\mathcal{H}^-$, we look for a substitution θ (if it exists) for the variables of A such that $A\theta$ unifies with every atom in $\mathcal{H}^+$ but it does not unify with any atom in $\mathcal{H}^-$. Observe that we want $A\theta$ to unify with each atom in $\mathcal{H}^+$ *separately*. To the best of our knowledge, this problem has not been considered before in the literature. In order to produce valid *run time* goals (*i.e.*, appropriate test cases) we also consider a *groundness* condition *i.e.*, a set G of variables that we want to be ground by θ.

Let us illustrate the notion of selective unification with a simple example. Consider, *e.g.*, an atom $p(X)$ and the sets $\mathcal{H}^+ = \{p(f(a)), p(f(Y))\}$, $\mathcal{H}^- = \{p(b)\}$ and $G = \{X\}$. A solution of this selective unification problem is $\{X/f(a)\}$, since $p(X)\{X/f(a)\} = p(f(a))$ unifies with $p(f(a))$ and also with $p(f(Y))$ (with different unifiers, though), but it does not unify with $p(b)$. Moreover, the variable $X \in G$ is ground. In contrast, the problem with atom $p(X)$ and sets $\mathcal{H}^+ = \{p(a), p(b)\}$ and $\mathcal{H}^- = \{p(c)\}$ is unfeasible since we need a renaming, *e.g.*, $\theta = \{X/Y\}$, in order for $p(X)\theta$ to unify with both $p(a)$ and $p(b)$, but then $p(X)\theta$ would also unify with $p(c)$.

In [1] we introduced a first algorithm for selective unification. Unfortunately, it was incomplete. In this paper, we further analyze this problem (see Sect. 3), identifying the potential sources of incompleteness, proving some properties, and introducing refined algorithms which are sound and complete under some circumstances. In Sect. 4, we also consider concolic testing in the context of constraint logic programming (CLP) [2, 3], and extend the notion of selective unification accordingly. Finally, Sect. 5 discusses some related work and concludes the paper.

[1] Concrete executions are assumed terminating via termination analysis or timeouts or limits on the number of inferences.

[2] In general, though, not all possibilities are feasible.

2 Preliminaries

We assume some familiarity with the standard definitions and notations for logic programming as introduced in [4] and for constraint logic programming as introduced in [3, 5]. Nevertheless, in order to make the paper as self-contained as possible, we present in this section the main concepts which are needed to understand our development.

We denote by $|S|$ the cardinality of the set S and by $\mathbb{N}$ the set of natural numbers. From now on, we fix an infinite countable set $\mathcal{V}$ of *variables* together with a *signature* Σ, *i.e.*, a pair $\langle F, \Pi_C \rangle$ where F is a finite set of *function symbols* and Π_C is a finite set of *predicate symbols* with $F \cap \Pi_C = \emptyset$ and $(F \cup \Pi_C) \cap \mathcal{V} = \emptyset$. Every element of $F \cup \Pi_C$ has an *arity* which is the number of its arguments. We write $f/n \in F$ (resp. $p/n \in \Pi_C$) to denote that f (resp. p) is an element of F (resp. Π_C) whose arity is $n \geq 0$. A *constant symbol* is an element of F whose arity is 0.

A *term* is a variable, a constant symbol or an entity $f(t_1, \ldots, t_n)$ where $f/n \in F$, $n \geq 1$ and $t_1, \ldots, t_n$ are terms. For any term t, we let $\mathcal{V}ar(t)$ denote the set of variables occurring in t. This notation is naturally extended to sets of terms. We say that t is *ground* when $\mathcal{V}ar(t) = \emptyset$. Positions are used to address the nodes of a term viewed as a tree. A *position* p in t, in symbols $p \in \mathcal{P}os(t)$, is represented by a finite sequence of natural numbers, where ε denotes the root position. We let $t|_p$ denote the *subterm* of t at position p and $t[s]_p$ the result of *replacing the subterm* $t|_p$ by the term s. The depth $depth(t)$ of t is defined as: $depth(t) = 0$ if t is a variable and $depth(f(t_1, \ldots, t_n)) = 1 + \mathsf{max}(depth(t_1), \ldots, depth(t_n))$, otherwise. We say that $t|_p$ is a subterm of t at depth k if there are k nested function symbols from the root of t to the root of $t|_p$.

An *atomic constraint* is an element $p/0$ of Π_C or an entity $p(t_1, \ldots, t_n)$ where $p/n \in \Pi_C$, $n \geq 1$ and $t_1, \ldots, t_n$ are terms. A first-order *formula* on Σ is built from atomic constraints in the usual way using the logical connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$ and the quantifiers $\exists$ and $\forall$. For any formula ϕ, we let $\mathcal{V}ar(\phi)$ denote its set of free variables and $\exists\phi$ (resp. $\forall\phi$) its existential (resp. universal) closure.

We fix a Σ-*structure* $\mathcal{D}$, *i.e.*, a pair $\langle D, [\cdot] \rangle$ which is an interpretation of the symbols in Σ. The set D is called the *domain* of $\mathcal{D}$ and $[\cdot]$ maps each $f/0 \in F$ to an element of D, each $f/n \in F$ with $n \geq 1$ to a function $[f] : D^n \rightarrow D$, each $p/0 \in \Pi_C$ to an element of $\{0, 1\}$, and each $p/n \in \Pi_C$ with $n \geq 1$ to a boolean function $[p] : D^n \rightarrow \{0, 1\}$. We assume that the binary predicate symbol $=$ is in Σ and is interpreted as identity in D. A *valuation* is a mapping from $\mathcal{V}$ to D. Each valuation v extends by morphism to terms. A valuation v induces a valuation $[\cdot]_v$ of terms to D and of formulas to $\{0, 1\}$.

Given a formula ϕ and a valuation v, we write $\mathcal{D} \models_v \phi$ when $[\phi]_v = 1$. We write $\mathcal{D} \models \phi$ when $\mathcal{D} \models_v \phi$ for all valuations v. Notice that $\mathcal{D} \models \forall\phi$ if and only if $\mathcal{D} \models \phi$, that $\mathcal{D} \models \exists\phi$ if and only if there exists a valuation v such that $\mathcal{D} \models_v \phi$, and that $\mathcal{D} \models \neg\exists\phi$ if and only if $\mathcal{D} \models \neg\phi$. We say that a formula ϕ is *satisfiable* (resp. *unsatisfiable*) in $\mathcal{D}$ when $\mathcal{D} \models \exists\phi$ (resp. $\mathcal{D} \models \neg\phi$).

We fix a set $\mathcal{L}$ of admitted formulas, the elements of which are called *constraints*. We suppose that $\mathcal{L}$ is closed under variable renaming, existential quantification and conjunction and that it contains all the atomic constraints. We assume that there is a computable function *solv* which maps each $c \in \mathcal{L}$ to one of `true` or `false` indicating whether c is satisfiable or unsatisfiable in $\mathcal{D}$. We call *solv* the *constraint solver*.

Example 1 (CLP($\mathcal{Q}_{lin}$)) *The constraint domain $\mathcal{Q}_{lin}$ has $<$, $\leq$, $=$, $\geq$, $>$ as predicate symbols, $+$, $-$, $*$, $/$ as function symbols and sequences of digits as constant symbols. $\mathcal{L}$ is the set of conjunctions of linear atomic constraints. The domain of computation is the structure with the set of rationals, denoted by $\mathbb{Q}$, as domain and where the predicate symbols and the function symbols are interpreted as the usual relations and functions over the rationals. A constraint solver for $\mathcal{Q}_{lin}$ always returning either* `true` *or* `false` *is described in [6].* □

Example 2 (Logic Programming) *The constraint domain Term has "=" as predicate symbol and strings of alphanumeric characters as function symbols. The domain of computation is the set of* finite trees *(or, equivalently, of finite terms), Tree. The interpretation of a constant is a tree with a single node labeled with the constant. The interpretation of an n-ary function symbol f is the function $f_{Tree} : Tree^n \to Tree$ mapping the trees $T_1, \ldots, T_n$ to a new tree with root labeled with f and with $T_1, \ldots, T_n$ as child nodes. A constraint solver always returning either* `true` *or* `false` *is provided by the* unification *algorithm.* □

Sequences of distinct variables are denoted by $\vec{X}$, $\vec{Y}$ or $\vec{Z}$ and are sometimes considered as sets of variables: we may write $\forall_{\vec{X}}$, $\exists_{\vec{X}}$ or $\vec{X} \cup \vec{Y}$. Sequences of (not necessarily distinct) terms are denoted by $\vec{s}$, $\vec{t}$ or $\vec{u}$. Given two sequences of n terms $\vec{s} := (s_1, \ldots, s_n)$ and $\vec{t} := (t_1, \ldots, t_n)$, we write $\vec{s} = \vec{t}$ to denote the constraint $s_1 = t_1 \wedge \ldots \wedge s_n = t_n$.

A structure $\mathcal{D}$ admits *quantifier elimination* if for each first-order formula ϕ there exists a quantifier-free formula ψ such that $\mathcal{D} \models \phi \leftrightarrow \psi$. $\mathcal{D}$ admits *variable elimination* if for each quantifier-free formula $\phi(\vec{X}, Y)$ there exists a quantifier-free formula $\psi(\vec{X})$ such that $\mathcal{D} \models \exists Y \phi(\vec{X}, Y) \leftrightarrow \psi(\vec{X})$. For instance, $\mathcal{Q}_{lin}$ admits variable elimination via the Fourier-Motzkin algorithm (see *e.g.*, [7]).

The signature in which all programs and queries under consideration are included is $\Sigma_L := \langle F, \Pi_C \cup \Pi_P \rangle$ where Π_P is the set of predicate symbols that can be defined in programs, with $\Pi_C \cap \Pi_P = \emptyset$. An *atom* has the form $p(t_1, \ldots, t_n)$ where $p/n \in \Pi_P$ and $t_1, \ldots, t_n$ are terms. The definitions and notations on terms ($\mathcal{V}ar$, *depth*, ground,...) are extended to atoms in the natural way. A *rule* has the form $H \leftarrow c \wedge \vec{B}$ where H is an atom called the *head* of the rule, c is a satisfiable constraint, and $\vec{B}$ is a finite sequence of atoms. A *program* is a finite set of rules. A *state* has the form $\langle d \,|\, \vec{B} \rangle$ where $\vec{B}$ is a sequence of atoms and d is a satisfiable constraint. A *constraint atom* is a state of the form $\langle d \,|\, p(\vec{t}) \rangle$. A constraint atom of the form $\langle c \,|\, p(\vec{X}) \rangle$ is *projected* when $\mathcal{V}ar(c) \subseteq \{\vec{X}\}$.

We consider the usual operational semantics given in terms of *derivations* from states to states. Let $\langle d \,|\, p(\vec{u}), \vec{B} \rangle$ be a state and $p(\vec{s}) \leftarrow c \wedge \vec{B'}$ be a fresh copy of a rule r. When $solv(\vec{s} = \vec{u} \wedge c \wedge d) =$ `true` then

$$\langle d \,|\, p(\vec{u}), \vec{B} \rangle \underset{r}{\Longrightarrow} \langle \vec{s} = \vec{u} \wedge c \wedge d \,|\, \vec{B'}, \vec{B} \rangle$$

is a *derivation step* of $\langle d \,|\, p(\vec{u}), \vec{B} \rangle$ with respect to r with $p(\vec{s}) \leftarrow c \wedge \vec{B'}$ as its *input rule*. Let S be the state $\langle d \,|\, \vec{B} \rangle$. S is *failed* if $\vec{B}$ is not empty and no derivation step is possible. S is *successful* if $\vec{B}$ is empty. We write $S \overset{+}{\underset{P}{\Longrightarrow}} S'$ to summarize a finite number (> 0) of derivation steps from S to S' where each input rule comes from program P. Let S_0 be a state. A sequence of derivation steps $S_0 \underset{r_1}{\Longrightarrow} S_1 \underset{r_2}{\Longrightarrow} \cdots$ of maximal length is called a *derivation* of $P \cup \{S_0\}$ when r_1, r_2, ... are rules from P and the *standardization apart*

condition holds, *i.e.*, each input rule used is variable disjoint from the initial state S_0 and from the input rules used at earlier steps.

Substitutions (denoted as $\theta, \sigma \ldots$) and their operations are defined as usual. In particular, for any substitution $\theta := \{X_1/t_1, \ldots, X_n/t_n\}$, the set $\mathcal{D}om(\theta) = \{X_1, \ldots, X_n\}$ is called the *domain* of θ and $\mathcal{R}an(\theta)$ is the set of variables appearing in $t_1, \ldots, t_n$. We let *id* denote the empty substitution. The application of θ to a syntactic object s (a term or an atom) is usually denoted by $s\theta$ rather than $\theta(s)$. The composition of θ and σ, written as $\theta\sigma$, is defined as: $X(\theta\sigma) = (X\theta)\sigma$ for any variable X. We say that θ is *idempotent* when $\theta\theta = \theta$. We write $\theta \leq \sigma$ iff $\sigma = \theta\eta$ for some substitution η. The *restriction* $\theta\restriction_V$ of θ to a set of variables V is defined as follows: $X\theta\restriction_V = X\theta$ if $X \in V$ and $X\theta\restriction_V = X$ otherwise. A syntactic object s_1 is *more general* than a syntactic object s_2, denoted $s_1 \leq s_2$, if there exists a substitution θ such that $s_2 = s_1\theta$. A *variable renaming* is a substitution that is a bijection on $\mathcal{V}$. We write $s_1 \sim s_2$ iff $s_1 = s_2\rho$ for some variable renaming ρ. A substitution θ is a *unifier* of s_1 and s_2 iff $s_1\theta = s_2\theta$; furthermore, θ is the *most general unifier* of s_1 and s_2, denoted by $\mathsf{mgu}(s_1, s_2)$ if, for every other unifier σ of s_1 and s_2, we have that $\theta \leq \sigma$. By abuse of notation, we also use mgu on a conjunction of equations, *i.e.*, $\mathsf{mgu}(s_1 = t_1 \wedge \ldots \wedge s_n = t_n) = \theta$ if $s_i\theta = t_i\theta$ for all $i = 1, \ldots, n$ and for every other unifier σ of s_i and t_i, $i = 1, \ldots, n$, we have $\theta \leq \sigma$. A syntactic object is *linear* if it does not contain multiple occurrences of the same variable. A substitution $\{X_1/t_1, \ldots, X_n/t_n\}$ is *linear* if $t_1, \ldots, t_n$ are linear and, moreover, they do not share variables.

3 Selective Unification in Logic Programming

In this section, we consider concolic testing and selective unification in the context of logic programming. In this setting, a *goal* is a finite sequence of atoms and the empty goal is denoted by true. Moreover, a rule has the form $H \leftarrow \vec{B}$ and is rather called a *clause*. Note that we only consider *definite* clauses *i.e.*, clauses whose head consists precisely of one atom.

3.1 Concolic Testing in Logic Programming

We first summarize the framework for concolic testing of logic programs introduced in [1]. On the positive side, in logic programming, the same principle for standard execution, SLD resolution, can also be used for symbolic execution. On the negative side, computing alternative test cases is way more complex than in the traditional setting (*e.g.*, for imperative programs) due to the non-deterministic nature of logic programming computations.

Concolic execution combines both concrete and symbolic execution. However, despite the fact that the concrete and symbolic execution mechanisms are the same, one still needs to consider *concolic* states that combine both a concrete and a symbolic (less instantiated) goal. Our concolic execution semantics deals with non-determinism and backtracking explicitly, similarly to the *linear* operational semantics of [8] for Prolog. In this context, rather than considering a goal, the semantics considers a sequence of goals that, roughly, represents a frontier of the execution tree built so far. To be precise, *concolic states* have the form $\langle S \,[\![\, S' \rangle$, where S and S' are sequences of (possibly labeled) concrete and symbolic goals, respectively. The structure of S and S' is identical, the only difference being that the atoms in S' might be less instantiated. Here, we use the vertical bar "|" as a delimiter for sequence

$$(success)\ \frac{}{\langle \mathsf{true}_\delta \,|\, S \,[\![\, \mathsf{true}_\theta \,|\, S' \rangle \rightsquigarrow_\diamond \langle \textsc{success}_\delta \,[\![\, \textsc{success}_\theta \rangle}$$

$$(failure)\ \frac{}{\langle (\mathsf{fail}, \vec{B})_\delta \,[\![\, (\mathsf{fail}, \vec{B'})_\theta \rangle \rightsquigarrow_\diamond \langle \textsc{fail}_\delta \,[\![\, \textsc{fail}_\theta \rangle}$$

$$(backtrack)\ \frac{S \neq \epsilon}{\langle (\mathsf{fail}, \vec{B})_\delta \,|\, S \,[\![\, (\mathsf{fail}, \vec{B'})_\theta \,|\, S' \rangle \rightsquigarrow_\diamond \langle S \,[\![\, S' \rangle}$$

$$(choice)\ \frac{\mathsf{clauses}(A,P) = \vec{c} \wedge \vec{c} = \{c_1, \ldots, c_n\} \wedge n > 0 \wedge \mathsf{clauses}(A', P) = \vec{d}}{\langle (A, \vec{B})_\delta \,|\, S \,[\![\, (A', \vec{B'})_\theta \,|\, S' \rangle \rightsquigarrow_{c(\ell(\vec{c}), \ell(\vec{d}))} \langle (A, \vec{B})^{c_1}_\delta \,|\, \ldots \,|\, (A, \vec{B})^{c_n}_\delta \,|\, S \,[\![\, (A', \vec{B'})^{c_1}_\theta \,|\, \ldots \,|\, (A', \vec{B'})^{c_n}_\theta \,|\, S' \rangle}$$

$$(choice\_fail)\ \frac{\mathsf{clauses}(A,P) = \emptyset \wedge \mathsf{clauses}(A', P) = \vec{c}}{\langle (A, \vec{B})_\delta \,|\, S \,[\![\, (A', \vec{B'})_\theta \,|\, S' \rangle \rightsquigarrow_{c(\emptyset, \ell(\vec{c}))} \langle (\mathsf{fail}, \vec{B})_\delta \,|\, S \,[\![\, (\mathsf{fail}, \vec{B'})_\theta \,|\, S' \rangle}$$

$$(unfold)\ \frac{\mathsf{mgu}(A, H_1) = \sigma \wedge \mathsf{mgu}(A', H_1) = \sigma'}{\langle (A, \vec{B})^{H_1 \leftarrow \vec{B_1}}_\delta \,|\, S \,[\![\, (A', \vec{B'})^{H_1 \leftarrow \vec{B_1}}_\theta \,|\, S' \rangle \rightsquigarrow_\diamond \langle (\vec{B_1}\sigma, \vec{B}\sigma)_{\delta\sigma} \,|\, S \,[\![\, (\vec{B_1}\sigma', \vec{B'}\sigma')_{\theta\sigma'} \,|\, S' \rangle}$$

Figure 1: Concolic execution semantics

elements. For instance, $(A, \vec{B})^c_\delta \,|\, S$ denotes a sequence of goals ending with the sequence S and starting with the goal $(A, \vec{B})^c_\delta$, which itself starts with the atom A, ends with the sequence $\vec{B}$ and is labeled with the substitution δ and the clause c. Given an arbitrary atom $p(\vec{u})$, an *initial concolic state* has the form $\langle p(\vec{u})_{id} \,[\![\, p(\vec{X})_{id} \rangle$, where $\vec{X}$ are different fresh variables and the labels id denote an initial (empty) computed substitution. Here, $p(\vec{u})$ can be considered a test case (a concrete goal), while $p(\vec{X})$ is the corresponding call with unknown, symbolic arguments, that we use to collect the *constraints* (here: substitutions for $\vec{X}$) using symbolic execution.

Example 3 *Given a concrete (atomic) goal, $p(f(X))$, the corresponding initial concolic state has the form $\langle p(f(X))_{id} \,[\![\, p(N)_{id} \rangle$, where N is a fresh variable.* □

In the following, we assume that every clause c has a corresponding unique label, which we denote by $\ell(c)$. By abuse of notation, we denote by $\ell(\vec{c})$ the set of labels $\{\ell(c_1), \ldots, \ell(c_n)\}$, where $\vec{c} = c_1, \ldots, c_n$. Also, given an atom A and a logic program P, $\mathsf{clauses}(A, P)$ returns the sequence of renamed apart program clauses of P whose head unifies with A. The concolic execution semantics is given by the rules of the labeled transition relation $\rightsquigarrow$ shown in Fig. 1. Some additional notations will be explained with the corresponding rules.

- The first rules, *success* and *failure*, use fresh constants labeled with a computed substitution to denote a final state: $\textsc{success}_\delta$ and $\textsc{fail}_\delta$, respectively.[3] Note that we are interested in both sucessful and (finitely) failing derivations. Rule *backtrack* applies when the first goal in the sequence finitely fails, but there is at least one alternative

[3]We note that the semantics only considers the computation of the first solution for the initial goal. This is the way most Prolog applications are used and, thus, the semantics models this behaviour in order to consider a realistic scenario. But if one wish to test a non-deterministic predicate p, one can add a clause $top \leftarrow p(\vec{X}), \mathsf{fail}.$ and start a concolic execution with the initial state $\langle top_{id} \,[\![\, top_{id} \rangle$.

$$
\begin{array}{rl}
 & \langle p(f(X))_{id} \,[\![\, p(N)_{id} \rangle \\
\rightsquigarrow^{choice}_{c(\{\ell_1,\ell_2\},\{\ell_1,\ell_2,\ell_3\})} & \langle p(f(X))^{\ell_1}_{id} \,|\, p(f(X))^{\ell_2}_{id} \,[\![\, p(N)^{\ell_1}_{id} \,|\, p(N)^{\ell_2}_{id} \rangle \\
\rightsquigarrow^{unfold}_{\diamond} & \langle \mathsf{true}_{\{X/a\}} \,|\, p(f(X))^{\ell_2}_{id} \,[\![\, \mathsf{true}_{\{N/f(a)\}} \,|\, p(N)^{\ell_2}_{id} \rangle \\
\rightsquigarrow^{success}_{\diamond} & \langle \textsc{success}_{\{X/a\}} \,[\![\, \textsc{success}_{\{N/f(a)\}} \rangle
\end{array}
$$

Figure 2: Concolic execution for $\langle p(f(X))_{id} \,[\![\, p(N)_{id} \rangle$

choice. In these three rules, we deal with the concrete and symbolic components of the concolic state in much the same way. Also, the steps are labeled with an *empty* label "$\diamond$".

- Rule *choice* represents the first stage of an SLD resolution step. If there is at least one clause whose head unifies with the leftmost atom of the concrete goal, this rule introduces as many copies of a goal as clauses returned by function clauses. Moreover, we label each copy of the goal $(A, \vec{B})$ with a matching clause. If there is at least one matching clause, unfolding is then performed by rule *unfold* using the clause labeling the goal. Otherwise, if there is no matching clause, rule *choice_fail* returns fail so that either rule *failure* or *backtrack* applies next. A relevant point here is that the steps with rules *choice* and *choice_fail* are labeled with a term of the form $c(L_1, L_2)$, where L_1 are the labels of the clauses matching the selected atom in the concrete goal and L_2 are the labels of the clauses matching the selected atom in the corresponding symbolic goal. Note that $L_1 \subseteq L_2$ since the concrete goal is always an instance of the symbolic goal. These labels are essential to compute alternative test cases during concolic testing, as we will see below.

- Essentially, one can say that the application of rules *choice* and *unfold* amounts to an unfolding step with plain SLD resolution. However, this is only true for the concrete component of the concolic state. Note that regarding the symbolic component, we do not consider all matching clauses, $\vec{d}$, but only the clauses matching the concrete goal (i.e., $\vec{c}$). This is a well known behavior in concolic execution: symbolic execution is restricted to only mimic the steps of the corresponding concrete execution.

Example 4 *Consider the following logic program:*

$$(\ell_1) \quad p(f(a)) \leftarrow \mathsf{true}. \qquad (\ell_2) \quad p(f(b)) \leftarrow \mathsf{true}. \qquad (\ell_3) \quad p(c) \leftarrow \mathsf{true}.$$

and the initial state $\langle p(f(X))_{id} \,[\![\, p(N)_{id} \rangle$. *Concolic execution proceeds as shown in Fig. 2.* □

Concolic testing aims at computing "test cases" (concrete atomic goals in our context) that cover all execution paths. Of course, since the number of paths is often infinite, one should consider a timeout or some other method to ensure the termination of the process. Note that concolic testing methods are typically incomplete. A concolic testing algorithm should follow these steps:

1. Given a concrete goal, we construct the associated initial concolic state and run concolic execution. We assume that concrete goals are terminating and, thus, this step is always finite too.

2. Then, we consider each application of rule *choice* in this concolic execution. Consider that the step is labeled with $c(L_1, L_2)$. Here, we are interested in looking for instances of the symbolic goal that match the clauses of every set in $\wp(L_2) \setminus L_1$ since the set L_1 is already considered by the current execution.[4]

3. Checking the feasibility for each set in $\wp(L_2) \setminus L_1$ is done as follows. Let A be the selected atom in the *symbolic* goal and let $L \in \wp(L_2) \setminus L_1$ be the considered set of clauses. Let $\mathcal{H}^+$ be the atoms in the heads of the clauses in L (*i.e.*, the clauses we want to unify with) and let $\mathcal{H}^-$ be the atoms in the heads of the clauses in $L_2 \setminus L$ (*i.e.*, the clauses we do not want to unify with). Then, we are looking for a substitution, θ, such that $A\theta$ unifies with each atom in $\mathcal{H}^+$ but it does not unify with any atom in $\mathcal{H}^-$. This is what we call a *selective unification problem* (see Sect. 3.2). As mentioned before, this is the first time such a unification problem has been considered in the literature. Usually, we also add another constraint: some variables must become ground by θ. This last requirement is needed to ensure that $A\theta$ is indeed a valid *concrete* (run time) goal and, thus, its execution terminates.[5]

4. Finally, for each selective unification problem which is solvable, we have a new concrete goal (*i.e.*, a new test case) and the process starts again. Moreover, one should keep track of the concrete goals already considered and the paths already explored in order to avoid computing the same test case once and again.

Let us now illustrate the concolic testing procedure with a simple example.

Example 5 *Consider again the program of Ex. 4, together with the initial goal $p(f(X))$. For simplicity, we will not consider a groundness condition in this example. Let us start with the concolic execution shown in Fig. 2. Given the label $c(\{\ell_1, \ell_2\}, \{\ell_1, \ell_2, \ell_3\})$, we have to consider the sets in $\wp(\{\ell_1, \ell_2, \ell_3\}) \setminus \{\ell_1, \ell_2\}$,* i.e.,

$$\{\emptyset, \{\ell_1\}, \{\ell_2\}, \{\ell_3\}, \{\ell_1, \ell_3\}, \{\ell_2, \ell_3\}, \{\ell_1, \ell_2, \ell_3\}\} .$$

Therefore, our first selective unification problem, associated to the empty set, considers the atom $p(N)$ and the sets $\mathcal{H}^+ = \emptyset$ and $\mathcal{H}^- = \{p(f(a)), p(f(b)), p(c)\}$. A solution is, e.g., *$\{N/a\}$ and, thus, $p(N)\{N/a\} = p(a)$ is another test case to consider.*

As for the second set, $\{\ell_1\}$, the selective unification problem considers the atom $p(N)$ and the sets $\mathcal{H}^+ = \{p(f(a))\}$ and $\mathcal{H}^- = \{p(f(b)), p(c)\}$. Here, the only solution is $\{N/f(a)\}$ and, thus, the atom $p(N)\{X/f(a)\} = p(f(a))$ is another test case to consider.

The process goes on producing the test cases $p(f(b))$ (for the set $\{\ell_2\}$), $p(c)$ (for the set $\{\ell_3\}$), and $p(N)$ (for the set $\{\ell_1, \ell_2, \ell_3\}$), the remaining problems being unfeasible. □

[4] Here, we denote by $\wp(S)$ the powerset of a set S. Moreover, for simplicity, we often use the label of a clause to refer to the clause itself.

[5] Which variables should be ground can be selected by instantiation mode declarations or by termination analysis, as some analysers for logic programming can infer subsets of argument positions such that if these arguments are ground, the computation is finite.

3.2 The Selective Unification Problem (SUP)

We write $A_1 \approx A_2$ to denote that the atoms A_1 and A_2 unify for some substitution.

Definition 6 (Selective Unification Problem, $\mathcal{P}$) *Let A be an atom, G be a set of variables with $G \subseteq \mathcal{V}ar(A)$ and $\mathcal{H}^+$ and $\mathcal{H}^-$ be finite sets of atoms such that: the elements of $A \cup \mathcal{H}^+ \cup \mathcal{H}^-$ are pairwise variable disjoint and $A \approx B$ for all $B \in \mathcal{H}^+ \cup \mathcal{H}^-$. The* selective unification problem *for A with respect to $\mathcal{H}^+$, $\mathcal{H}^-$ and G consists in determining whether the following set of substitutions is empty:*

$$\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G) = \left\{ \sigma\restriction_{\mathcal{V}ar(A)} \;\middle|\; \begin{array}{c} \forall H \in \mathcal{H}^+ : A\sigma \approx H \\ \wedge\, \forall H \in \mathcal{H}^- : \neg(A\sigma \approx H) \\ \wedge\, G\sigma \text{ is ground} \end{array} \right\} .$$

The substitutions in $\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$ (if any) are the solutions *of the problem. We say that the problem is* satisfiable *when it has a solution,* i.e.*, $\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G) \neq \emptyset$.*

Example 7 *We illustrate the notion of selective unification with several examples below.*

- *Let $A = p(X)$, $\mathcal{H}^+ = \{p(a), p(b)\}$, $\mathcal{H}^- = \emptyset$ and $G = \emptyset$. Then, the empty substitution is a solution,* i.e.*, $id \in \mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$, as $p(X)$ unifies with $p(a)$ and $p(b)$.*
- *Let $A = p(X)$, $\mathcal{H}^+ = \{p(a), p(b)\}$, $\mathcal{H}^- = \{p(f(Z))\}$ and $G = \emptyset$. This problem has no solution,* i.e.*, $\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G) = \emptyset$. Indeed, one cannot find an instance of A that unifies with both atoms in $\mathcal{H}^+$ and does not unify with $p(f(Z))$.*
- *Let $A = p(X)$, $\mathcal{H}^+ = \{p(s(Y))\}$, $\mathcal{H}^- = \{p(s(0))\}$ and $G = \{X\}$. There are infinitely many solutions, among them we find $\{X/s^{n+2}(0)\}$ for $n \in \mathbb{N}$. For instance, let us check that $\sigma = \{X/s(s(0))\}$ is a solution. We have $A\sigma = p(s(s(0)))$. Moreover, $A\sigma$ and $p(s(Y))$ unify while $A\sigma$ and $p(s(0))$ do not unify, and $X\sigma$ is ground.*
- *Let $A = p(X, Y)$, $\mathcal{H}^+ = \{p(a, b), p(Z, Z)\}$ and $\mathcal{H}^- = \emptyset = G$. Then, id, $\{X/a\}$ and $\{Y/b\}$ are solutions. For instance, let us check that $\sigma = \{X/a\}$ is a solution. We have $A\sigma = p(a, Y)$, hence $A\sigma$ unifies with $p(a, b)$ and with $p(Z, Z)$. Moreover, as $\mathcal{H}^-$ and G are empty, the last two conditions of $\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$ are trivially true.* □

3.3 A Sound Algorithm for the SUP

When the considered signature is finite, the following algorithm is sound and complete for solving a selective unification problem for an atom A with respect to $\mathcal{H}^+$, $\mathcal{H}^-$ and G: first, bind the variables of A with all the terms of depth 0. If all the corresponding substitutions are not members of $\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$, then try with all the terms of depth 1. We keep increasing the considered term depth until a solution is found. Here, we prove that there exists a finite number n such that, if a solution has not been found when considering the terms of depth n, then the problem is not satisfiable.

For simplicity, in the next result we consider that both A and $\mathcal{H}^+$ are linear.

Theorem 8 (Decidability) *Let A be a linear atom with $G \subseteq \mathcal{V}ar(A)$, $\mathcal{H}^+$ be a finite set of linear atoms and $\mathcal{H}^-$ be a finite set of atoms such that all atoms are pairwise variable disjoint and $A \approx B$ for all $B \in \mathcal{H}^+ \cup \mathcal{H}^-$. Then, checking that $\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G) \neq \emptyset$ is decidable.*

Proof: Here, we assume the naive algorithm sketched above. Let us first consider that all atoms in $\{A\} \cup \mathcal{H}^+ \cup \mathcal{H}^-$ are linear. Let k be the maximum depth of the atoms in $\{A\} \cup \mathcal{H}^+ \cup \mathcal{H}^-$. Consider the set

$$\Theta' = \{\theta \mid \mathcal{D}om(\theta) \subseteq \mathcal{V}ar(A),\ depth(A\theta) \leq k+1\}\ .$$

On Θ', we define the binary relation $\theta_1 \simeq \theta_2$ iff $A\theta_1 \sim A\theta_2$. The relation $\simeq$ is an equivalence relation. Let $\Theta = \Theta'/\simeq$. The set Θ is usually large but finite. Now, we proceed by contradiction and assume that the problem is satisfiable but there is no solution in Θ *i.e.*, $\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G) \neq \emptyset$ and $\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G) \cap \Theta = \emptyset$.

Let σ be one of such solutions *i.e.*, $\sigma \in \mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$ and $\sigma \notin \Theta$. Let k' be the maximum depth of the atoms in $\mathcal{H}^+$. We have $k' \leq k$. Let $s_1, \ldots, s_n$ be the non-variable terms at depth $k'+1$ or higher in $A\sigma$ and let $p_1, \ldots, p_n$ be their respective position. Trivially, all atoms in $\mathcal{H}^+$ should have a variable at depth k' or lesser in order to still unify with $A\sigma$. Therefore, replacing $s_1, \ldots, s_n$ by any term in $A\sigma$, one gets and an atom that still unifies with all atoms in $\mathcal{H}^+$. Formally, $(\ldots(A\sigma[t_1]_{p_1})\ldots)[t_n]_{p_n} \approx H$ for all $H \in \mathcal{H}^+$ and for all terms $t_1, \ldots, t_n$.

Now, let us consider the negative atoms. Let us focus on the worst case, where the maximum depth of the atoms in $\mathcal{H}^-$ is $k \geq k'$. Since $\neg(A\sigma \approx H)$ for all $H \in \mathcal{H}^-$ and $(\ldots(A\sigma[t_1]_{p_1})\ldots)[t_n]_{p_n} \approx H$ for all $H \in \mathcal{H}^+$ and for all terms $t_1, \ldots, t_n$, let us choose terms

$$t'_1, \ldots, t'_n \text{ such that } \begin{cases} \neg((\ldots(A\sigma[t'_1]_{p_1})\ldots)[t'_n]_{p_n} \approx H) & \text{for all } H \in \mathcal{H}^- \text{ and} \\ (\ldots(A\sigma[t'_1]_{p_1})\ldots)[t'_n]_{p_n} & \text{has depth } k+1. \end{cases}$$

Note that this is always possible since, in the worst case, for each term in the atoms of $\mathcal{H}^-$ at depth k, we might need a term at depth $k+1$ (when the term in the atom of $\mathcal{H}^-$ is the only constant of the signature, so we need to introduce a function symbol and another constant if the argument should be ground). Let σ' be a substitution such that $\mathcal{D}om(\sigma') \subseteq \mathcal{V}ar(A)$ and $A\sigma' = (\ldots(A\sigma[t'_1]_{p_1})\ldots)[t'_n]_{p_n}$. Then, $\sigma' \in \mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$ with $\sigma' \in \Theta$ and, thus, we get a contradiction. □ Extending the above result to non-linear atoms is not difficult but it is tedious since we have to consider a higher depth that may depend on the multiple occurrences of the same variables. For instance, given a non-linear atom $A = p(X, f(f(X)))$ with $\mathcal{H}^+ = \{p(f(a), f(Y))\}$, considering solutions up to depth 2 (the maximum depth of the considered atoms) is not enough. Here, one would need a substitution $\sigma = \{X/f(a)\}$ so that the depth of $p(X, f(X))\sigma = p(f(a), f(f(f(a))))$ is 4. In general, when considering non-linear atoms, there still exists a finite depth k such that the set Θ (as in the proof above) is finite, but the considered depth might be higher. On the other hand, we conjecture that the above naive algorithm would also be complete for infinite signatures (*e.g.*, integers) since the number of symbols in the considered atoms is finite. Nonetheless, such algorithms may be so inefficient that they are impractical in the context of concolic testing.

We note that the set $\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$ is usually infinite. Moreover, even when considering only the *most general* solutions in this set, there may still exist more than one.

Example 9 *Consider* $A = p(X, Y)$, $\mathcal{H}^+ = \{p(Z, Z), p(a, b)\}$, $\mathcal{H}^- = \{p(c, c)\}$ *and* $G = \emptyset$. *Then, both substitutions* $\{X/a, Y/U\}$ *and* $\{X/U, Y/b\}$ *are most general solutions in* $\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$. *In principle, any of them is equally good in our context.* □

In [1], we have introduced a stepwise method that, intuitively speaking, proceeds as follows:

- First, we produce some "maximal" substitutions θ for A such that $A\theta$ still unifies with the atoms in $\mathcal{H}^+$. Here, we use a special set $\mathcal{U}$ of fresh variables with $\mathcal{V}ar(\{A\} \cup \mathcal{H}^+ \cup \mathcal{H}^-) \cap \mathcal{U} = \emptyset$. The elements of $\mathcal{U}$ are denoted by $U, U', U_1 \ldots$ Then, in θ, the variables from $\mathcal{U}$ (if any) denote positions where further binding *might* prevent $A\theta$ from unifying with some atom in $\mathcal{H}^+$.
- In a second stage, we look for another substitution η such that $\theta\eta$ is a solution of the selective unification problem, *i.e.*, $\theta\eta \in \mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$. Here, we basically follow a generate and test algorithm (as in the naive algorithm above), but it is now much more restricted thanks to the bindings in θ and the fact that binding variables from $\mathcal{U}$ is not allowed.

In the following, we recall the selective unification algorithm from [1] that was conjectured to be complete. Here, we prove that it is indeed incomplete and we identify the two sources of incompleteness.

3.3.1 Dealing with the Positive Atoms

In the first stage, we use the variables from the special set $\mathcal{U}$ to replace *disagreement pairs* (see [4] p. 27). Roughly speaking, given terms s and t, a subterm s' of s and a subterm t' of t form a disagreement pair if the root symbols of s' and t' are different, but the symbols from s' up to the root of s and from t' up to the root of t are the same. For instance, $X, g(a)$ and $b, h(Y)$ are disagreement pairs of the terms $f(X, g(b))$ and $f(g(a), g(h(Y)))$. A disagreement pair t, t' is called *simple* if one of the terms is a variable that does not occur in the other term and no variable of $\mathcal{U}$ occurs in t, t'.

Definition 10 (Positive Unification Algorithm, $\mathcal{SU}^+$) **Input:** *an atom A and a set of atoms $\mathcal{H}^+$ such that all atoms are pairwise variable disjoint and $A \approx B$ for all $B \in \mathcal{H}^+$.*

Output: *a substitution θ.*

1. *Let $\mathcal{B} := \{A\} \cup \mathcal{H}^+$.*
2. *While simple disagreement pairs occur in $\mathcal{B}$ do*
   (a) *non-deterministically choose a simple disagreement pair X, t (respectively, t, X) in $\mathcal{B}$;*
   (b) *$\mathcal{B} := \mathcal{B}\eta$, where $\eta = \{X/t\}$.*[6]
3. *While $|\mathcal{B}| \neq 1$ do*
   (a) *non-deterministically choose a disagreement pair t, t' in $\mathcal{B}$;*
   (b) *replace t, t' with a fresh variable from $\mathcal{U}$.*

[6] I.e., we construct a new set by applying η to each atom of $\mathcal{B}$ and we assign this new set to $\mathcal{B}$.

4. *Return $\theta\gamma$, where $\mathcal{B} = \{B\}$, $A\theta = B$, $\mathcal{D}om(\theta) \subseteq \mathcal{V}ar(A)$, and γ is a variable renaming for the variables of $\mathcal{V}ar(A\theta)\backslash\mathcal{U}$ with fresh variables from $\mathcal{V}\backslash\mathcal{U}$.*

We denote by $\mathcal{SU}^+(A, \mathcal{H}^+)$ the set of non-deterministic substitutions computed by this algorithm.

Observe that the step (2a) involves two types of non-determinism:

- *Don't care* non-determinism, when there are several disagreement pairs X, t (or t, X) for *different* variables. In this case, we can select any of them and continue with the next step. The final solution would be the same no matter the selection. This is also true for step (3a), since the order in which the non-simple disagreement pairs are selected will not affect the final result.
- *Don't know* non-determinism, when there are several disagreement pairs X, t (or t, X) for the same variable X. In this case, we should consider all possibilities since they may give rise to different solutions.

Example 11 *Let $A = p(X, Y)$ and $\mathcal{H}^+ = \{p(a, b), p(Z, Z)\}$. Therefore, we start with $\mathcal{B} := \{p(X, Y), p(a, b), p(Z, Z)\}$. The algorithm then considers the simple disagreement pairs in $\mathcal{B}$. From X, a, we get $\eta_1 := \{X/a\}$ and the action (2b) sets $\mathcal{B}$ to $\mathcal{B}\eta_1 = \{p(a, Y), p(a, b), p(Z, Z)\}$. The substitution $\eta_2 := \{Y/b\}$ results from Y, b and the action (2b) sets $\mathcal{B}$ to $\mathcal{B}\eta_2 = \{p(a, b), p(Z, Z)\}$. Now, we have two* don't know *non-deterministic possibilities:*

- *If we consider the disagreement pair a, Z, we have a substitution $\eta_3 := \{Z/a\}$ and action (2b) then sets $\mathcal{B}$ to $\mathcal{B}\eta_3 = \{p(a, b), p(a, a)\}$. Now, no simple disagreement pair occurs in $\mathcal{B}$, hence the algorithm jumps to the loop at line 3. Action (3b) replaces the disagreement pair b, a with a fresh variable $U \in \mathcal{U}$, hence $\mathcal{B}$ is set to $\{p(a, U)\}$. As $|\mathcal{B}| = 1$ the while loop of line 3 stops and the algorithm returns the substitution $\{X/a, Y/U\}$.*
- *If we consider the disagreement pair b, Z instead, we have a substitution $\eta_3' := \{Z/b\}$, and action (2b) sets $\mathcal{B}$ to $\mathcal{B}\eta_3' = \{p(a, b), p(b, b)\}$. Now, by proceeding as in the previous case, the algorithm returns $\{X/U', Y/b\}$.*

Therefore, $\mathcal{SU}^+(A, \mathcal{H}^+) = \{\{X/a, Y/U\}, \{X/U', Y/b\}\}$. □

We note that the algorithm in Def. 10 assumes that the input atom A is always more general than the final atom B so that the last step (line 4) is well defined. An invariant proving that this is indeed the case can be stated as follows:

Proposition 12 *The following statement is an invariant of the loops at lines 2 and 3 of the algorithm: $A \approx B$ for all $B \in \mathcal{B}$ and $A \leq B'$ for some $B' \in \mathcal{B}$.*

Proof: By induction on the iteration n of each loop. □

Now we prove termination and soundness of the algorithm. To this end, we prove termination of each loop together with some invariants, see Prop. 14 and Prop. 15 below. The proof of these propositions relies on the following technical result.

Lemma 13 *Suppose that $A\theta = B\theta$ for some atoms A and B and some substitution θ. Then we have $A\theta\eta = B\eta\theta\eta$ for any substitution η with $[\mathcal{D}om(\eta) \cap \mathcal{V}ar(B)] \cap \mathcal{D}om(\theta) = \emptyset$ and $\mathcal{R}an(\eta) \cap \mathcal{D}om(\theta\eta) = \emptyset$.*

Proof: For any $X \in \mathcal{V}ar(B)$,

- either $X \notin \mathcal{D}om(\eta)$ and then $X\eta\theta\eta = X\theta\eta$
- or $X \in \mathcal{D}om(\eta)$ and then $X\eta\theta\eta = (X\eta)\theta\eta = X\eta$ because $\mathcal{R}an(\eta) \cap \mathcal{D}om(\theta\eta) = \emptyset$. Moreover, $X \notin \mathcal{D}om(\theta)$ because $[\mathcal{D}om(\eta) \cap \mathcal{V}ar(B)] \cap \mathcal{D}om(\theta) = \emptyset$, so $X\theta\eta = X\eta$. Finally, $X\eta\theta\eta = X\theta\eta$.

Consequently, $B\eta\theta\eta = B\theta\eta$. As $A\theta = B\theta$, we have $A\theta\eta = B\theta\eta$ *i.e.*, $A\theta\eta = B\eta\theta\eta$. □

Proposition 14 *The loop at line 2 always terminates and the following statement is an invariant for it: for each $A' \in \{A\} \cup \mathcal{H}^+$ there exists $B \in \mathcal{B}$ and a substitution θ such that $A'\theta = B\theta$ and $\mathcal{V}ar(\mathcal{B}) \cap \mathcal{D}om(\theta) = \emptyset$.*

Proof: Action (2b) reduces the number of simple disagreement pairs in $\mathcal{B}$, which implies termination. The invariant can be proved by induction on the iteration n of the loop using Lemma 13. □

Proposition 15 *The loop at line 3 always terminates and the following statement is an invariant for it: for each $A' \in \{A\} \cup \mathcal{H}^+$ there exists $B \in \mathcal{B}$ and a substitution θ such that $A'\theta = B\theta$, $\mathcal{D}om(\theta) \subseteq (\mathcal{V}ar(\mathcal{H}^+ \cup \{A\}) \cup \mathcal{U})$ and $\mathcal{V}ar(\mathcal{B}) \cap \mathcal{D}om(\theta) \subseteq \mathcal{U}$.*

Proof: Action (3b) reduces the number of disagreement pairs in $\mathcal{B}$, which implies termination. The invariant can be proved by induction on the iteration n of the loop using Prop. 14 in the base case. □

The next theorem states termination and soundness of the Positive Unification Algorithm 10. Note that this result was incomplete in [1] since the condition on $\mathcal{R}an(\eta)$ was missing.

Theorem 16 *Let A be an atom and $\mathcal{H}^+$ be a set of atoms such that all atoms are pairwise variable disjoint and $A \approx B$ for all $B \in \mathcal{H}^+$. The algorithm in Def. 10 with input A and $\mathcal{H}^+$ always terminates. Moreover, for all $\theta \in \mathcal{SU}^+(A, \mathcal{H}^+)$, we have that $A\theta\eta \approx H$ for all $H \in \mathcal{H}^+$ and for any idempotent substitution η with $\mathcal{D}om(\eta) \subseteq \mathcal{V}ar(A\theta)\backslash\mathcal{U}$ and $\mathcal{R}an(\eta) \cap (\mathcal{V}ar(\mathcal{H}^+ \cup \{A\}) \cup \mathcal{U}) = \emptyset$.*

Proof: Termination of the algorithm results from Prop. 14 and Prop. 15.

Upon termination of the loop at line 3 we have $|\mathcal{B}| = 1$. Let B be the element of $\mathcal{B}$ with $A\theta = B$ and let $\theta' \in \mathcal{SU}^+(A, \mathcal{H}^+)$ be a renaming of θ for the variables of $A\theta\backslash\mathcal{U}$. By Prop. 15, we have that for all $H \in \mathcal{H}^+$ there exists a substitution μ such that $A\theta\mu = H\mu$ and the following conditions hold:

- $\mathcal{D}om(\mu) \subseteq (\mathcal{V}ar(\mathcal{H}^+ \cup \{A\}) \cup \mathcal{U})$ and
- $\mathcal{V}ar(A\theta) \cap \mathcal{D}om(\mu) \subseteq \mathcal{U}$.

Trivially, there exists a unifier μ' for $A\theta'$ and H too, and the same conditions hold: $\mathcal{D}om(\mu') \subseteq (\mathcal{V}ar(\mathcal{H}^+ \cup \{A\}) \cup \mathcal{U})$ and $\mathcal{V}ar(A\theta') \cap \mathcal{D}om(\mu') \subseteq \mathcal{U}$.

Now, in order to apply Lemma 13, we need to prove the following conditions:

- $[\mathcal{D}om(\eta) \cap \mathcal{V}ar(A\theta')] \cap \mathcal{D}om(\mu') = \emptyset$. This is trivially implied by the fact that $\mathcal{D}om(\eta) \subseteq \mathcal{V}ar(A\theta') \backslash \mathcal{U}$ and $\mathcal{V}ar(A\theta') \cap \mathcal{D}om(\mu') \subseteq \mathcal{U}$.
- $\mathcal{R}an(\eta) \cap \mathcal{D}om(\mu'\eta) = \emptyset$. First, since $\mathcal{D}om(\mu'\eta) \subseteq \mathcal{D}om(\mu') \cup \mathcal{D}om(\eta)$, we prove a stronger claim: $\mathcal{R}an(\eta) \cap \mathcal{D}om(\mu') = \emptyset$ and $\mathcal{R}an(\eta) \cap \mathcal{D}om(\eta) = \emptyset$. The second condition is trivially implied by the idempotency of η. Regarding the first condition, it is implied by $\mathcal{R}an(\eta) \cap (\mathcal{V}ar(\mathcal{H}^+ \cup \{A\}) \cup \mathcal{U}) = \emptyset$ since $\mathcal{D}om(\mu') \subseteq (\mathcal{V}ar(\mathcal{H}^+ \cup \{A\}) \cup \mathcal{U})$, which is true.

Therefore, by Lemma 13, we have that $A\theta'\eta\mu'\eta = H\mu'\eta$ and, thus, $A\theta'\eta$ unifies with H. Hence, we have proved that $A\theta'\eta$ unifies with every atom in $\mathcal{H}^+$. □

3.3.2 Dealing with the Negative Atoms

Now we deal with the negative atoms and the groundness constraints.

Definition 17 (Selective Unification Algorithm, $\mathcal{SU}$) Input: *an atom A with $G \subseteq \mathcal{V}ar(A)$ a set of variables, and two finite sets $\mathcal{H}^+$ and $\mathcal{H}^-$ such that all atoms are pairwise variable disjoint and $A \approx B$ for all $B \in \mathcal{H}^+ \cup \mathcal{H}^-$.*

Output: fail *or a substitution $\theta\eta$ (restricted to the variables of A).*

1. *Generate – using a fair algorithm – pairs (θ, η) with $\theta \in \mathcal{SU}^+(A, \mathcal{H}^+)$ and η an idempotent substitution such that $G\theta\eta$ is ground, $\mathcal{D}om(\eta) \subseteq \mathcal{V}ar(A\theta) \backslash \mathcal{U}$ and $\mathcal{R}an(\eta) \cap (\mathcal{V}ar(\mathcal{H}^+ \cup \{A\}) \cup \mathcal{U}) = \emptyset$; otherwise, return* fail.
2. *Check that for each $H^- \in \mathcal{H}^-$, $\neg(A\theta\eta \approx H^-)$; otherwise, return* fail.
3. *Return $\theta\eta\gamma$ (restricted to the variables of A), where γ is a variable renaming for $A\theta\eta$ with fresh variables from $\mathcal{V} \backslash \mathcal{U}$.*

We denote by $\mathcal{SU}(A, \mathcal{H}^+, \mathcal{H}^-, G)$ the set of non-deterministic (non-failing) substitutions computed by the above algorithm.

Note that step (1) above is *don't know* non-deterministic and, thus, all substitutions in $\mathcal{SU}^+(A, \mathcal{H}^+)$ should in principle be considered. On the other hand, computing the first solution of the above algorithm is enough for concolic testing.

The soundness of the selective unification algorithm is a straightforward consequence of Theorem 16 and the fact that the algorithm in Def. 17 is basically a fair generate-and-test procedure. Unfortunately, the selective unification algorithm is not complete in general, as illustrated below. Ex. 18 shows that the algorithm cannot always compute all the solutions while Ex. 19 shows that it may even find no solution at all for a satisfiable instance of the problem.

Example 18 *Consider the atom $A = p(X_1, X_2)$ with $G = \{X_1\}$ and the sets*

$$\mathcal{H}^+ = \{p(X, g(X)), p(Z, Z)\} \quad \textit{and} \quad \mathcal{H}^- = \{p(g(b), W)\} .$$

Here, we have

$$\mathcal{SU}^+(A, \mathcal{H}^+) = \{\underbrace{\{X_1/X', X_2/U\}}_{\theta_1}, \underbrace{\{X_1/U, X_2/g(X')\}}_{\theta_2}\} .$$

The algorithm is able to compute the solution $\{X_1/g(a), X_2/U\}$ *from* θ_1, $\eta = \{X'/g(a)\}$ *and* $\gamma = id$. *However, it cannot compute* $\{X_1/g(a), X_2/g(X')\} \in \mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$. □

The algorithm fails here because the instantiation of variables from $\mathcal{U}$ is not allowed. In [1], it was incorrectly assumed that *any* binding of a variable from $\mathcal{U}$ will result in a substitution θ' such that $A\theta'$ does not unify will all the atoms in $\mathcal{H}^+$ anymore. However, the universal quantification was not right. For each variable from $\mathcal{U}$, we can only ensure that *there exists some* term t such that binding this variable to t will result in a substitution that prevents A from unifying with some atom in $\mathcal{H}^+$. Therefore, since the algorithm of Def. 17 forbids the bindings of the variables in $\mathcal{U}$, completeness is lost. We will propose a solution to this problem in the next section.

Example 19 *Consider* $A = p(X_1, X_2)$, $\mathcal{H}^+ = \{p(X, a), p(b, Y)\}$, $\mathcal{H}^- = \{p(b, a)\}$ *and* $G = \emptyset$. *Here, we have* $\mathcal{SU}^+(A, \mathcal{H}^+) = \{\{X_1/b, X_2/a\}\}$ *so the algorithm in Def. 17 fails. However, the following substitution* $\{X_1/Z, X_2/Z\}$ *is a solution,* i.e., $\{X_1/Z, X_2/Z\} \in \mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$. □

Unfortunately, we do not know how to generate such non-linear solutions except with the naive semi-algorithm mentioned at the beginning of this section, which is not generally useful in practice. Therefore, in the next section we will rule out these solutions.

3.4 The Linear Case

In this section, we introduce an alternative to recover the completeness of the selective unification algorithm. In the following, we only consider a subset of the solutions, namely those which are *linear*:

$$\mathcal{P}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G) = \{\sigma \in \mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G) \mid \sigma \text{ is linear}\} .$$

Hence, we rule out solutions like those in Ex. 19 since we do not know how to produce them using a constructive algorithm. We refer to $\mathcal{P}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G)$ as the solutions to the *linear selective unification problem*. Below, we introduce an algorithm for solving the linear problem which is sound and complete when the atoms in A and $\mathcal{H}^+$ are linear.

3.4.1 Dealing with the Positive Atoms

Formally, we are concerned with the following unification problem:

Definition 20 (Positive Linear Unification Problem, $\mathcal{P}^+_{\mathsf{lin}}$) *Let* A *be a linear atom and let* $\mathcal{H}^+$ *be a finite set of linear atoms such that all atoms are pairwise variable disjoint and* $A \approx B$ *for all* $B \in \mathcal{H}^+$. *Then, the* positive linear unification problem *for* A *with respect to* $\mathcal{H}^+$ *consists in determining whether the following set is empty:*

$$\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+) = \{\sigma\restriction_{\mathcal{V}ar(A)} \mid (\forall H \in \mathcal{H}^+ : A\sigma \approx H) \text{ and } \sigma \text{ is linear}\} .$$

Let us recall that we do not want to find a unifier between A and *all* the atoms in $\mathcal{H}^+$, but a substitution θ such that $A\theta$ still unifies with *each* atom in $\mathcal{H}^+$ independently. So this problem is different from the usual unification problems found in the literature.

Clearly, $|\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)| \geq 1$ since the identity substitution is always a solution to the positive linear unification problem. In general, though, the set $\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$ is infinite.

Example 21 *Let us consider* $A = p(X)$ *and* $\mathcal{H}^+ = \{p(f(Y)), p(f(g(Z)))\}$. *Then, we have* $\{id, \{X/f(X')\}, \{X/f(g(X'))\}, \{X/f(g(a))\}, \{X/f(g(f(X')))\}, \ldots\} \subseteq \mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$, *which is clearly infinite.* □

Therefore, in the following, we restrict our attention to so called *maximal* solutions:

Definition 22 (Maximal Solution) *Let* A *be a linear atom and* $\mathcal{H}^+$ *be a finite set of linear atoms such that all atoms are pairwise variable disjoint and* $A \approx B$ *for all* $B \in \mathcal{H}^+$. *We say that a substitution* $\theta \in \mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$ *is* maximal *when the following conditions hold:*

1. *for any idempotent substitution* γ *with* $\mathcal{D}om(\gamma) \subseteq \mathcal{V}ar(A\theta)\backslash\mathcal{U}$ *and* $\mathcal{R}an(\gamma) \cap (\mathcal{V}ar(\mathcal{H}^+ \cup \{A\}) \cup \mathcal{U}) = \emptyset$, $(\theta\gamma)\restriction_{\mathcal{V}ar(A)}$ *is still an element of* $\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$,
2. *for any variable* $U \in \mathcal{V}ar(A\theta) \cap \mathcal{U}$, *we have that* $(\theta\{U/t\})\restriction_{\mathcal{V}ar(A)}$ *is not an element of* $\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$ *for all non-variable term* t, *and*
3. *for any* $X/t \in \theta$ *and for all non-variable term* $t|_p$, *replacing it by a non-variable term rooted by a different symbol will result in a substitution which is not an element of* $\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$ *anymore.*

We let $max(A, \mathcal{H}^+)$ *denote the set of maximal solutions in* $\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$.

Intuitively speaking, given a maximal solution θ, the first condition implies that $(\theta\gamma)\restriction_{\mathcal{V}ar(A)}$ is still a solution of the positive linear unification problem as long as no variables from $\mathcal{U}$ are bound. The second and third conditions mean that the rest of the symbols in θ cannot be changed, *i.e.*, binding a variable from $\mathcal{U}$ with a non-variable term or changing any constant or function symbol by a different one will always result in a substitution which is not a solution anymore.

Example 23 *Consider,* e.g., $A = p(X_1, X_2)$ *and* $\mathcal{H}^+ = \{p(f(Y), a), p(f(g(Z)), b)\}$. *Here, we have* $\{X_1/X', X_2/X''\} \in \mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$ *but it is not a maximal solution,* i.e., $\{X_1/X', X_2/X''\} \notin max(A, \mathcal{H}^+)$ *since binding* X'' *to,* e.g., a, *will result in a substitution which is not in* $\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$. *In contrast,* $\{X_1/f(g(Z')), X_2/U\}$ *is a maximal solution. However, any substitution of the form* $\{X_1/f(g(t)), X_2/U\}$ *for any non-variable term* t *is not a maximal solution since the third condition will not hold anymore (one can change the symbols introduced by* t *and still get a solution in* $\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$*). The substitution* $\{X_1/f(Y'), X_2/U\}$ *is not a maximal solution as well since binding* Y' *to,* e.g., a, *will result in a substitution which is not in* $\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$, *hence the first condition does not hold. And the same applies to* $\{X_1/f(U'), X_2/U\}$, *which is not a maximal solution either since we can bind* U' *to* $g(X')$ *and still get a substitution in* $\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$. □

In contrast to $\mathcal{P}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$, the set $max(A, \mathcal{H}^+)$ is finite, since it is bounded by the depth of the terms in $\mathcal{H}^+$. Actually, for linear atoms in $\{A\} \cup \mathcal{H}^+$, there is only *one* maximal solution.

Proposition 24 *Let* A *be a linear atom and* $\mathcal{H}^+$ *be a finite set of linear atoms such that all atoms are pairwise variable disjoint and* $A \approx B$ *for all* $B \in \mathcal{H}^+$. *Then, the set* $max(A, \mathcal{H}^+)$ *is a singleton (up to variable renaming).*

Proof: We proceed by contradiction. Assume that there are two maximal solutions $\sigma, \theta \in max(A, \mathcal{H}^+)$, where $X/s \in \sigma$ and $X/t \in \theta$ for some variable $X \in \mathcal{V}ar(A)$. Consider that s and t differ at position p such that $s|_p$ and $t|_p$ are rooted by a different symbol. We distinguish the following cases:

- If $s|_p$ and $t|_p$ are rooted by different constant or function symbols, we get a contradiction by condition (3) of maximal solution.
- If $s|_p$ is rooted by a constant or function symbol, while $t|_p$ is rooted by a variable from $\mathcal{U}$ (or viceversa), we get a contradiction by condition (2) of maximal solution.
- If $s|_p$ is rooted by a constant or function symbol, while $t|_p$ is rooted by a variable from $\mathcal{V}\backslash\mathcal{U}$ (or viceversa), we get a contradiction either by condition (1) or (3) of maximal solution.
- Finally, if $s|_p$ is rooted by a variable from $\mathcal{U}$, while $t|_p$ is rooted by a variable from $\mathcal{V}\backslash\mathcal{U}$ (or viceversa), we get a contradiction either by condition (1) or (2) of maximal solution.

Therefore, the set $max(A, \mathcal{H}^+)$ is necessarily a singleton. □ Moreover, the following key property holds: a maximal solution can always be *completed* in order to get a solution to the linear unification problem when it is satisfiable. In order to prove this result, we need to recall the definition of *parallel composition* of substitutions, denoted by $\Uparrow$ in [9].

Definition 25 (Parallel Composition [9]) *Let θ_1 and θ_2 be two idempotent substitutions. Then, we define $\Uparrow$ as follows:*

$$\theta_1 \Uparrow \theta_2 = \begin{cases} \mathsf{mgu}(\widehat{\theta}_1 \wedge \widehat{\theta}_2) & \text{if } \widehat{\theta}_1 \wedge \widehat{\theta}_2 \text{ has a solution (a unifier)} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

where $\widehat{\theta}$ denotes the equational representation *of a substitution θ,* i.e.*, if $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ then $\widehat{\theta} = (X_1 = t_1 \wedge \cdots \wedge X_n = t_n)$.*

Proposition 26 *Let A be a linear atom and $\mathcal{H}^+$ be a finite set of linear atoms such that all atoms are pairwise variable disjoint and $A \approx B$ for all $B \in \mathcal{H}^+$. Let $\theta \in max(A, \mathcal{H}^+)$ be the maximal solution for A and $\mathcal{H}^+$. If $\mathcal{P}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G) \neq \emptyset$ then there exists some substitution γ such that $\theta\gamma \in \mathcal{P}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G)$.*

Proof: For simplicity, we consider that $A = p(X)$, $\mathcal{H}^+ = \{p(t_1), \ldots, p(t_n)\}$ and $\mathcal{H}^- = \{p(s_1), \ldots, p(s_m)\}$. Since the atoms are linear, the claim would follow by a similar argument. Let $\theta = \{X/t\} \in max(A, \mathcal{H}^+)$ be the maximal solution. Hence, we have $t \approx t_i$ for all $i = 1, \ldots, n$. Let $\sigma \in \mathcal{P}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G)$ be a solution to the selective unification problem. By definition of maximal solution, there may be other solutions to the positive unification problem, but every introduced symbol cannot be different if we want to still unify with all terms $t_1, \ldots, t_n$ by condition (3) in the definition of maximal solution. Therefore, both substitutions must be compatible, *i.e.*, we have $\theta \Uparrow \sigma = \delta \neq \mathsf{fail}$. Furthermore, taking into account the negative atoms in $\mathcal{H}^-$ as well as the groundness constraints with respect to G, δ can only introduce further bindings, but would never require to generalize any term

introduced by θ and, thus, δ can be decomposed as $\theta\gamma$, with $\theta\gamma \in \mathcal{P}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G)$. □
Therefore, computing the maximal solution suffices to check for satisfiability. Here, we use again the algorithm in Def. 10 for computing the maximal solution, with the following differences: i) first, both A and the atoms in $\mathcal{H}^+$ are now linear; ii) step (2a) is now *don't care* non-deterministic, so the algorithm will return a single solution, which we denote by $\mathcal{SU}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$.

Proposition 27 *Let A be a linear atom and $\mathcal{H}^+$ be a finite set of linear atoms such that all atoms are pairwise variable disjoint and $A \approx B$ for all $B \in \mathcal{H}^+$. Then, $\mathcal{SU}^+_{\mathsf{lin}}(A, \mathcal{H}^+) = max(A, \mathcal{H}^+)$.*

Proof: The fact that $\mathcal{SU}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$ returns a singleton is trivial by definition, since the algorithm has no *don't know* non-determinism and no step admits a failure.

Regarding the fact that θ is a maximal solution, let us prove that all three conditions in Def. 22 hold. The first condition of maximal solution follows by Theorem 16, which is proved for the more general case of arbitrary (possibly non-linear) atoms. The third condition holds from the fact that in step (2) of $\mathcal{SU}^+_{\mathsf{lin}}$ only symbols from the atoms A and $\mathcal{H}^+$ are introduced following a *mgu*-like algorithm; therefore they are possibly not necessary, but cannot be replaced by different symbols and still unify with all the atoms in $\mathcal{H}^+$. Finally, the second condition derives from step (3) of $\mathcal{SU}^+_{\mathsf{lin}}$ where non-simple disagreement pairs are replaced by fresh variables from $\mathcal{U}$ and, thus, any binding to a non-variable term would result in $A\theta$ not unifying with some atom of $\mathcal{H}^+$. □

3.4.2 Dealing with the Negative Atoms

The algorithm $\mathcal{SU}$ in Def. 17 is now redefined as follows.

Definition 28 (Linear Selective Unification Algorithm, $\mathcal{SU}_{\mathsf{lin}}$) Input: *a linear atom A with $G \subseteq \mathcal{V}ar(A)$ a set of variables, and two finite sets $\mathcal{H}^+$ and $\mathcal{H}^-$ such that the atoms in $\mathcal{H}^+$ are linear and all atoms are pairwise variable disjoint and $A \approx B$ for all $B \in \mathcal{H}^+ \cup \mathcal{H}^-$.*

Output: fail *or a substitution $\theta\eta$ (restricted to the variables of A).*

1. *Let $\{\theta\} = \mathcal{SU}^+_{\mathsf{lin}}(A, \mathcal{H}^+)$. Generate – using a fair algorithm – linear idempotent substitutions η such that $G\theta\eta$ is ground, $\mathcal{D}om(\eta) \subseteq \mathcal{V}ar(A\theta)\backslash\mathcal{U}$ and $\mathcal{R}an(\eta) \cap (\mathcal{V}ar(\mathcal{H}^+ \cup \{A\}) \cup \mathcal{U}) = \emptyset$, otherwise return* fail.
2. *Check that for each $H^- \in \mathcal{H}^-$, $\neg(A\theta\eta \approx H^-)$, otherwise return* fail.
3. *Return $\theta\eta\gamma$ (restricted to the variables of A), where γ is a variable renaming for $A\theta\eta$ with fresh variables from $\mathcal{V}\backslash\mathcal{U}$.*

We denote by $\mathcal{SU}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G)$ the set of non-deterministic (non-failing) substitutions computed by the above algorithm.

Example 29 *Consider again $A = p(X_1, X_2)$ and $\mathcal{H}^+ = \{p(f(Y), a), p(f(g(Z)), b)\}$, together with $\mathcal{H}^- = \{p(f(g(a)), c)\}$ and $G = \{X_1\}$. The algorithm for linear positive unification returns the maximal substitution $\{X_1/f(g(Z')), X_2/U\}$. Therefore, the algorithm for*

linear selective unification would eventually produce a solution of the form $\theta = \{X_1/f(g(b)), X_2/X'\}$ *since* $A\theta = p(f(g(b)), X')$ *unifies with* $p(f(Y), a)$ *and* $p(f(g(Z)), b)$ *but not with* $p(f(g(a)), c)$ *and, moreover,* X_1 *is ground. However, if we consider a non-maximal solution, the algorithm in Def. 17 may fail, even if there exists some solution to the linear selective unification problem. This is the case,* e.g.*, if we consider the non-maximal solution* $\{X_1/f(g(a)), X_2/U\}$*.* □

Theorem 30 (Soundness) *Let* A *be a linear atom with* $G \subseteq \mathcal{V}ar(A)$*,* $\mathcal{H}^+$ *be a finite set of linear atoms and* $\mathcal{H}^-$ *be a finite set of atoms such that all atoms are pairwise variable disjoint and* $A \approx B$ *for all* $B \in \mathcal{H}^+ \cup \mathcal{H}^-$*. Then, we have* $\mathcal{SU}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G) \subseteq \mathcal{P}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G)$*.*

Proof: The claim follows from Prop. 27 by assuming that the *don't know* non-deterministic substitutions considered in step (1) of the algorithm of Def. 28 are obtained by a fair generate-and-test algorithm which produces substitutions systematically starting with terms of depth 0, then depth 1, etc., as in the naive algorithm described at the beginning of Sect. 3.3. □ The following result states the completeness of our algorithm. In principle, we do not guarantee that *all* solutions are computed using our algorithms, even for the linear case. However, we can ensure that if the linear selective unification problem is satisfiable, our algorithm will find at least one solution (which is sufficient in the context of concolic testing).

Theorem 31 (Completeness) *Let* A *be a linear atom with* $G \subseteq \mathcal{V}ar(A)$*,* $\mathcal{H}^+$ *be a finite set of linear atoms and* $\mathcal{H}^-$ *be a finite set of atoms such that all atoms are pairwise variable disjoint and* $A \approx B$ *for all* $B \in \mathcal{H}^+ \cup \mathcal{H}^-$*. If* $\mathcal{P}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G) \neq \emptyset$ *then* $\mathcal{SU}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G) \neq \emptyset$*.*

Proof: By Prop. 26, if $\mathcal{P}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G) \neq \emptyset$ and θ is the computed maximal solution, then there exists a substitution γ such that $(\theta\gamma)\restriction_{\mathcal{V}ar(A)} \in \mathcal{P}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G)$. Moreover, such a substitution γ can be obtained by a fair generate-and-test algorithm such as that considered in Def. 28. Finally, the claim follows by Prop. 27 which ensures that the algorithm in Def. 10 will always produce the maximal solution for A and $\mathcal{H}^+$. □

Example 32 *Consider again* $A = p(X_1, X_2)$ *and* $\mathcal{H}^+ = \{p(f(Y), a), p(f(g(Z)), b)\}$*, together with* $\mathcal{H}^- = \{p(g(W), c)\}$ *and* $G = \emptyset$*. The algorithm for linear positive unification returns the maximal substitution* $\{X_1/f(g(Z')), X_2/U\}$*. Therefore, it is impossible that the algorithm in Def. 17 could produce a solution like* $\{X_1/f(X'), X_2/X''\} \in \mathcal{P}_{\mathsf{lin}}(A, \mathcal{H}^+, \mathcal{H}^-, G)$*.* □

In the next section, we extend the notion of selective unification to constraint logic programs.

4 Selective Unification in Constraint Logic Programming

4.1 Concolic Testing in Constraint Logic Programming

Extending the concolic testing framework from logic programs to CLP is not difficult. In this section, we focus on the main differences and also show an example that illustrates the

technique in this setting.

First, the concolic execution semantics for the CLP case is basically equivalent to that in Fig. 1 by replacing goals with states of the form $\langle c \,|\, \vec{B} \rangle$ and by considering the usual unfolding rule for CLP programs. Furthermore, the function clauses is now redefined as follows. Given a state $\langle d \,|\, p(\vec{u}) \rangle$ and a set of rules P, we have

$$\mathsf{clauses}(\langle d \,|\, p(\vec{u}) \rangle, P) = \{ p(\vec{s}) \leftarrow c \wedge \vec{B} \in P \mid solv(\vec{s} = \vec{u} \wedge c \wedge d) = \mathtt{true} \} \; .$$

Concolic testing then proceeds basically as in the logic programming case. The main difference, though, is that the selective unification problems now deal with states and CLP programs rather than goals and logic programs. Let us consider a choice step labeled with $c(L_1, L_2)$. Here, given a set $L \in \wp(L_2) \setminus L_1$, the sets $\mathcal{H}^+$ and $\mathcal{H}^-$ are built as follows:

$$\begin{array}{l} \mathcal{H}^+ = \{ \langle c \,|\, H \rangle \mid H \leftarrow c \wedge \vec{B} \in L \} \\ \mathcal{H}^- = \{ \langle c \,|\, H \rangle \mid H \leftarrow c \wedge \vec{B} \in L_2 \setminus L \} \; . \end{array}$$

The groundness condition, if any, will now require some variables to have a fixed value in a given constraint (see Sect. 4.2).

Example 33 *Consider the following CLP($\mathcal{Q}_{lin}$) program:*

$$(\ell_1)\; p(X) \leftarrow X \leq 0. \qquad (\ell_2)\; p(X) \leftarrow X \geq 0 \wedge X < 10.$$

and a choice step labeled with $c(\{\ell_1\}, \{\ell_1, \ell_2\})$, where the symbolic state is $\langle \mathsf{true} \,|\, p(N) \rangle$. Hence, we have to consider the sets in $\wp(\{\ell_1, \ell_2\}) \setminus \{\ell_1\} = \{\emptyset, \{\ell_2\}, \{\ell_1, \ell_2\}\}$.

Our first selective unification problem, associated to the empty set, considers the state $\langle \mathsf{true} \,|\, p(N) \rangle$ and the sets $\mathcal{H}^+ = \emptyset$ and $\mathcal{H}^- = \{\langle X \leq 0 \,|\, p(X) \rangle, \langle X \geq 0 \wedge X < 10 \,|\, p(X) \rangle\}$. A solution is, e.g.*, $N \geq 10$. Thus, $\langle N \geq 10 \,|\, p(N) \rangle$ is another test case to consider.*

As for the second set, $\{\ell_2\}$, the selective unification problem considers the state $\langle \mathsf{true} \,|\, p(N) \rangle$ and the sets $\mathcal{H}^+ = \{\langle X \geq 0 \wedge X < 10 \,|\, p(X) \rangle\}$ and $\mathcal{H}^- = \{\langle X \leq 0 \,|\, p(X) \rangle\}$. Here, a possible solution is $N > 0 \wedge N < 10$. Thus, the state $\langle N > 0 \wedge N < 10 \,|\, p(N) \rangle$ is another test case to consider.

Finally, for the set $\{\ell_1, \ell_2\}$, the selective unification problem considers the state $\langle \mathsf{true} \,|\, p(N) \rangle$ and the sets $\mathcal{H}^+ = \{\langle X \leq 0 \,|\, p(X) \rangle, \langle X \geq 0 \wedge X < 10 \,|\, p(X) \rangle\}$ and $\mathcal{H}^- = \emptyset$, where the only solution is $N = 0$. Thus, our final alternative test case is the state $\langle N = 0 \,|\, p(N) \rangle$. □

4.2 The Constraint Selective Unification Problem (CSUP)

In this section, we generalize Def. 6 to CLP. Let $A_1 = \langle c_1 \,|\, p(\vec{u}) \rangle$ and $A_2 = \langle c_2 \,|\, p(\vec{s}) \rangle$ be two constraint atoms with no common variable. We write $A_1 \approx A_2$ or A_1 *unifies with* A_2 to denote that $\mathcal{D} \models \exists(\vec{u} = \vec{s} \wedge c_1 \wedge c_2)$. For simplicity, in the following definition, we consider that the constraint atom has the form $\langle c_A \,|\, p(\vec{X}) \rangle$. There is no loss of generality since any arbitrary constraint atom $\langle c \,|\, p(\vec{u}) \rangle$ can be trivially transformed into $\langle \vec{u} = \vec{X} \wedge c \,|\, p(\vec{X}) \rangle$.

Definition 34 (Constraint Selective Unification Problem, $\mathcal{P}$) *Let A be a constraint atom of the form $\langle c_A \,|\, p(\vec{X}) \rangle$ with $G \subseteq \mathcal{V}ar(A)$. Let $\mathcal{H}^+$ and $\mathcal{H}^-$ be finite sets of constraint atoms such that all constraint atoms, including A, are pairwise variable disjoint and $A \approx B$*

for all $B \in \mathcal{H}^+ \cup \mathcal{H}^-$. Then, the constraint selective unification problem *for A with respect to $\mathcal{H}^+$, $\mathcal{H}^-$ and G consists in determining whether the following set of constraints is empty:*

$$\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G) = \left\{ c_A \wedge c \;\middle|\; \begin{array}{l} c_A \wedge c \text{ is satisfiable} \\ \wedge\ c \text{ is variable disjoint with } \mathcal{H}^+ \cup \mathcal{H}^- \\ \wedge\ \forall H \in \mathcal{H}^+ : \langle c_A \wedge c \,|\, p(\vec{X}) \rangle \approx H \\ \wedge\ \forall H \in \mathcal{H}^- : \neg(\langle c_A \wedge c \,|\, p(\vec{X}) \rangle \approx H) \\ \wedge\ \text{each } X \in G \text{ is fixed within } c_A \wedge c \end{array} \right\}.$$

The constraints c in $\mathcal{P}(A, \mathcal{H}^+, \mathcal{H}^-, G)$ (if any) are the solutions *of the problem.*

Intuitively, to solve a CSUP, we consider any constraint c such that $\langle c_A \wedge c \,|\, p(\vec{X}) \rangle$ still unifies with all the positive constraint atoms while preventing any unification with the negative constraint atoms and ensuring that the variables in G have a fixed value.

Note that $X \in G$ *is fixed within* $c_A \wedge c$ is the equivalent of the *groundness* condition of the LP case and we keep calling it the *groundness* condition in the CLP case. Some constraint solvers might give to X the required value, but it is not mandatory as it can be expressed in first order logic by stating that, within $c_A \wedge c$, X has exactly one value. For instance, X is fixed within $c(X, \vec{X})$ is equivalent to $\mathcal{D} \models \exists X \left(\exists \vec{X} c(X, \vec{X}) \wedge \forall Y \forall \vec{Y} \big(c(Y, \vec{Y}) \rightarrow X = Y \big) \right)$.

Example 35 (CLP($\mathcal{Q}_{lin}$)) *We illustrate the notion of selective unification in the context of CLP($\mathcal{Q}_{lin}$) with the next examples.*

- *Let $A := \langle 0 \leq X \wedge X \leq 5 \,|\, p(X) \rangle$, $\mathcal{H}^+ := \{\langle 4 \leq Y \,|\, p(Y) \rangle\}$, $\mathcal{H}^- := \{\langle Z < 2 \,|\, p(Z) \rangle\}$ and $G = \{X\}$. There is an infinite number of solutions, among which one can find the constraint $c \equiv (X = 9/2)$. It is equivalent to the satisfiable constraint $(0 \leq X \wedge X \leq 5 \wedge X = 9/2)$ and it entails that X is ground. Moreover, the constraint $(X = Y \wedge 4 \leq Y \wedge X = 9/2)$ is satisfiable while $(X = Z \wedge Z < 2 \wedge X = 9/2)$ is unsatisfiable.*

- *Let $A := \langle 0 \leq X \wedge X \leq 5 \,|\, p(X) \rangle$, $\mathcal{H}^+ := \{\langle 4 \leq Y_1 \,|\, p(Y_1) \rangle, \langle Y_2 \leq 1 \,|\, p(Y_2) \rangle\}$, $\mathcal{H}^- := \{\langle 2 < Z \wedge Z < 3 \,|\, p(Z) \rangle\}$ and $G = \emptyset$. There is no solution here. Intuitively, if there is a solution on the left of 2, then it excludes the first positive atom or if there is a solution to the right of 3, then it excludes the second positive atom. So one cannot find a conjunction of atomic constraints that include the elements of $\mathcal{H}^+$ and exclude the element of $\mathcal{H}^-$ because the set of points described by such a conjunction is convex.* □

The following general result holds in CLP.

Theorem 36 (Undecidability) *It is undecidable whether an arbitrary instance of the CSUP has a solution.*

Proof: We provide a complete proof in [10] which can be summarized as follows. We consider any Turing machine M and any word w and we define an instance $\mathcal{P}_{M,w}$ of the generic CSUP in a constraint logic programming language CLP($\mathcal{A}$), the class of constraints of which is a strict subclass of the *array property* fragment introduced in [11]. We encode the tape of M as an array and we define $\mathcal{P}_{M,w}$ so that M accepts w if and only if $\mathcal{P}_{M,w}$

has a solution. Then the theorem results from the fact that it is undecidable whether an arbitrary Turing machine accepts an arbitrary word. □

However, we have identified some decidable classes of CSUP instances. Their definition rely on the next two properties of the constraint structure $\mathcal{D}$ under consideration.

Definition 37 *We let **(A1)** and **(A2)** denote the following properties of $\mathcal{D}$.*

- ***(A1)** $\mathcal{D}$ admits variable elimination.*
- ***(A2)** The negation of any atomic constraint is equivalent to a finite disjunction of atomic constraints.*

Example 38 *$\mathcal{Q}_{lin}$ satisfies these properties. It admits variable elimination. The set of predefined predicate symbols is $\Pi_C = \{</2, \leq/2, =/2, \geq/2, >/2\}$. The negation of each atomic constraint is an atomic constraint, except for $=/2$ whose negation is defined by a disjunction of atomic constraints,* i.e., $\neg(X = Y) \equiv (X < Y \vee X > Y)$. □

(A1) and **(A2)** are sufficient conditions on the constraint domain to solve the CSUP without the groundness condition (*i.e.*, when $G = \emptyset$). The following results can be found in [10]:

Theorem 39 *If **(A1)** and **(A2)** hold, then it is decidable whether an arbitrary instance of the CSUP with $G = \emptyset$ has a solution.*

By relying on specific properties of CLP($\mathcal{Q}_{lin}$) which allow to pick a ground value satisfying a constraint, one can even extend Theorem 39 to any instance of the CSUP (*i.e.*, when $G \neq \emptyset$):

Theorem 40 *In CLP($\mathcal{Q}_{lin}$), it is decidable whether an arbitrary instance of the CSUP has a solution.*

5 Related Work and Conclusion

In this paper, we have studied the soundness and completeness of selective unification, a relevant operation in the context of concolic testing of logic programs. We have reconsidered the algorithm provided in [1]: we have improved its correctness result (a condition was missing in [1]) and we have identified its main sources of incompleteness. Then, we have introduced several refinements so that the procedure is now sound and complete with respect to linear solutions. For the non-linear case, the decidability of the selective unification problem is an open problem.

We are not aware of any other work that deals with the kind of unification problems that we study in this paper. We have also considered concolic testing in the framework of constraint logic programming. We have extended the notion of selective unification accordingly and we have sketched that the selective unification problem is generally undecidable for CLP. This paper is both an extended version of [12] and a summary of [10], where the extension to CLP is fully described. We have included all the proofs that were missing in [12] about soundness and termination of the algorithm for positive unification ([12] only provides a portion of the soundness proof).

Constructive negation in LP [13] and CLP [14] is related to our work. Its starting point stems from the desire to extract constructive information from the proof of a negative subgoal, in contrast to the usual negation as failure rule. In [14], Stuckey introduced *admissible closedness* as a sufficient condition over constraint structures for a practical use of constructive negation. This property is weaker than quantifier elimination. Stuckey showed that properties **(A1)** and **(A2)** of Sect. 4.2 imply admissible closedness. Moreover, CLP($\mathcal{H}$) (the constraint domain of finite trees with equality and quantified disequality) does not admit quantifier elimination but is admissible closed. More recently, Dovier *et al.* proved that admissible closedness was also a necessary condition for constructive negation in CLP [15]. So this concept could be a promising tool for studying concolic testing for logic programming from a CLP perspective.

Finally, we are also working on an improved concolic testing scheme [16] which supports negative constraints and defines selective unification problems as constraints on Herbrand terms. This approach opens the door to an SMT-based implementation of a concolic testing tool that, hopefully, will scale better to larger applications.

References

[1] Mesnard F, Payet E, Vidal G. Concolic Testing in Logic Programming. *Theory and Practice of Logic Programming*, 2015. **15**(4-5):711–725. doi: 10.1017/S1471068415000332.

[2] Jaffar J, Lassez JL. Constraint Logic Programming. In: Proc. of the 14th Annual ACM Symposium on Principles of Programming Languages (POPL'87). ACM Press, 1987 pp. 111–119. doi:10.1145/41625.41635.

[3] Jaffar J, Maher MJ. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 1994. **19,20**:503–581. doi:10.1016/0743-1066(94)90033-7.

[4] Apt KR. From Logic Programming to Prolog. Prentice Hall, 1997. ISBN 978-0-13-230368-2.

[5] Jaffar J, Maher MJ, Marriott K, Stuckey PJ. The Semantics of Constraint Logic Programs. *Journal of Logic Programming*, 1998. **37**(1-3):1–46. doi:10.1016/S0743-1066(98)10002-X.

[6] Refalo P, Van Hentenryck P. CLP($\mathcal{R}_{lin}$) Revised. In: Maher M (ed.), Proc. of the Joint International Conference and Symposium on Logic Programming. The MIT Press, 1996 pp. 22–36.

[7] Marriott K, Stuckey PJ. Programming with Constraints: An Introduction. The MIT Press, 1998.

[8] Ströder T, Emmes F, Schneider-Kamp P, Giesl J, Fuhs C. A Linear Operational Semantics for Termination and Complexity Analysis of ISO Prolog. In: Vidal G (ed.), Proc. of the 21st International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'11), volume 7225 of *Lecture Notes in Computer Science*. Springer, 2012 pp. 237–252. doi:10.1007/978-3-642-32211-2_16.

[9] Palamidessi C. Algebraic Properties of Idempotent Substitutions. In: Paterson MS (ed.), Proc. of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90), volume 443 of *Lecture Notes in Computer Science*. Springer, 1990 pp. 386–399. doi:10.1007/BFb0032046.

[10] Mesnard F, Payet E, Vidal G. Selective Unification in Constraint Logic Programming. In: Vanhoof W, Pientka B (eds.), Proc. of the 19th International Symposium on Principles and Practice of Declarative Programming (PPDP'17). ACM, 2017 pp. 115–126. doi:10.1145/3131851.3131863.

[11] Bradley AR, Manna Z, Sipma HB. What's Decidable About Arrays? In: Emerson EA, Namjoshi KS (eds.), Proc. of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'06), volume 3855 of *Lecture Notes in Computer Science*. Springer, 2006 pp. 427–442. doi:10.1007/11609773_28.

[12] Mesnard F, Payet E, Vidal G. On the Completeness of Selective Unification in Concolic Testing of Logic Programs. In: Hermenegildo MV, López-García P (eds.), Proc. of the 26th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'16), volume 10184 of *Lecture Notes in Computer Science*. Springer, 2016 pp. 205–221. doi:10.1007/978-3-319-63139-4_12.

[13] Chan D. Constructive Negation Based on the Completed Database. In: Kowalski RA, Bowen KA (eds.), Proc. of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP'88). MIT Press, 1988 pp. 111–125.

[14] Stuckey PJ. Negation and Constraint Logic Programming. *Information and Computation*, 1995. **118**(1):12–33. doi:10.1006/inco.1995.1048.

[15] Dovier A, Pontelli E, Rossi G. A Necessary Condition for Constructive Negation in Constraint Logic Programming. *Information Processing Letters*, 2000. **74**(3-4):147–156. doi:10.1016/S0020-0190(00)00046-6.

[16] Fortz S, Mesnard F, Payet E, Perrouin G, Vanhoof W, Vidal G. An SMT-Based Concolic Testing Tool for Logic Programs. Technical report, Universitat Politècnica de València, 2019. Submitted for publication, URL `http://personales.upv.es/gvidal/german/SMTConcolicTool/paper.pdf`.