INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Subtyping constraints in quasi-lattices

Emmanuel Coquery — François Fages

N° ????

Septembre 2003

THÈME 2

Rapport de recherche

ISSN 0249-6399

Subtyping constraints in quasi-lattices

Emmanuel Coquery , François Fages

Thème 2 — Génie logiciel
et calcul symbolique
Projet Contraintes

Rapport de recherche n° ???? — Septembre 2003 — 24 pages

Abstract: In this report, we show the decidability and NP-completeness of the satisfiability problem for non-structural subtyping constraints in quasi-lattices. This problem, first introduced by Smolka in 1989, is important for the typing of logic and functional languages. The decidability result is obtained by generalizing Trifonov and Smith's algorithm over lattices, to the case of quasi-lattices. Similarly, we extend Pottier's algorithm for computing explicit solutions to the case of quasi-lattices. Finally we evoke some applications of these results to type inference in constraint logic programming and functional programming languages.

Key-words: subtyping constraints, quasi-lattices

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

Contraintes de sous-typage dans les quasi-treillis

Résumé : Dans ce rapport nous montrons la décidabilité et la NP-complétude du problème de la satisfaction des contraintes de sous-typage non-structurel dans les quasi-treillis. Ce problème posé par Smolka en 1989 est important pour le typage des langages logiques et fonctionnels. Le résultat de décidabilité est obtenu en généralisant l'algorithme de Trifonov et Smith dans les treillis, au cas des quasi-treillis. Nous étendons également l'algorithme de Pottier de calcul explicite de solutions au cas des quasi-treillis. Nous évoquons ensuite les applications de ces résultats, notamment au système TCLP de typage des programmes logiques avec contraintes.

Mots-clés : contraintes de sous-typage, quasi-treillis

1 Introduction

The search for more and more flexible type systems for programming languages goes with the search for algorithms for solving subtyping constraints in more and more complex type structures. Type checking and type inference algorithms for a program basically consist in solving systems of subtyping constraints of the form $\exists X \bigwedge_{i=1}^{n} t_i \le t'_i$ where t_i, t'_i are types and X is the set of variables appearing in the system.

In its most general form, non-structural subtyping combines subtyping and parametric polymorphisms and allows subtyping relations between type constructors of different arities. For instance, in the type system for constraint logic programming TCLP [4], the subtyping relation $list(\alpha) \le term$ allows us to see a (homogeneous) list as a term. In a lattice of type constructors, Trifonov and Smith [12] gave a simple decomposition algorithm, with a complexity in $O(n^3)$, for testing the satisfiability of non-structural subtyping constraints in a lattice of infinite or regular types. Pottier [9] extended this algorithm to compute solutions explicitly when they exist. However, the lattice structure of type constructors imposes the existence of a minimal element $\perp$ and a maximal element $\top$, and thus does not treat the typing with the empty type $\perp$ as an error. In the particular case of object types *à la* Abadi - Cardelli [1], where type constructors are defined and ordered by their invariant or covariant labels, Palsberg, Zhao and Jim [7] gave an $O(n^3)$ algorithm for solving subtyping constraints in this specific type structure.

In this paper, we are interested in the solving of non-structural subtyping constraints in more general structures than lattices. We consider quasi-lattices of types, that is partially ordered sets for which any non-empty subset having a lower bound (resp. an upper bound) has a greatest lower bound (resp. least upper bound). These structures allow the absence of the types $\top$ and $\perp$. The decidability of non-structural subtyping constraints satisfiability in quasi-lattices is an open problem mentioned in Smolka's thesis [11]. One difficulty of non structural subtyping is its capacity to forget arguments of parametric types. For example, let us consider the types $list(\alpha)$ representing homogeneous lists and $nhlist$ representing non-homogeneous lists, with $list(\alpha) \le nhlist$. Let us also consider the following constraint system: $list(nhlist) \le \alpha, list(int) \le \alpha$. In a lattice, it is equivalent to $list(\top) \le \alpha$. In a quasi-lattice without $\top$ element, the system also has a solution $\alpha = nhlist$. It is thus not correct to solve the system in the lattice obtained by completion with $\perp$ and $\top$, and then simply check the absence of $\top$ and $\perp$ in the bounds.

In this paper, we bring a positive answer to the decidability problem by generalizing Trifonov and Smith's algorithm to quasi-lattices, and we prove the NP-completeness of this problem. The rest of the paper is organized as follows. In the next section, we define the ordered set of (possibly infinite) types formed upon a quasi-lattice of type constructors of different arities, and we prove that this set is a quasi-lattice. In section 3, we show that in quasi-lattices, the systems closed by Trifonov and Smith's decomposition rules are satisfiable, and we give an algorithm for testing the satisfiability of subtyping constraints with a time complexity in $O(m^v M^v n^3)$, where m (resp. M) stands for the number of minimal (resp. maximal) elements of the quasi-lattice, v is the number of unbounded variables and n is the number of constraints. The NP-completeness of constraint satisfiability is shown in this section by using the result of Pratt and Tiuryn for n-crowns [10]. In section 4, we generalize Pottier's algorithm for computing explicit solutions in quasi-lattices. Section 5 presents some applications of these results to type checking and we conclude in the last section. The proofs which do not appear in the main text are given in the appendix.

2 Types

2.1 Preliminaries

Let $(E, \leq)$ be a partially ordered set. For a nonempty subset S of E, we note $\downarrow S = \{x \in E | \forall y \in S\ x \leq y\}$ the set of lower bounds of S and $\uparrow S = \{x \in E | \forall y \in S\ y \leq x\}$ the set of upper bounds of S. For the empty set, $\downarrow\emptyset = \emptyset$ and $\uparrow\emptyset = \emptyset$. We note $\sqcap S$ (resp. $\sqcup S$) the greatest lower bound (resp. least upper bound) of S whenever it exists. A lower quasi-lattice (resp. upper quasi-lattice) is a partially ordered set where any finite subset having a lower (resp. upper) bound has a greatest lower bound (resp. a least upper bound). A quasi-lattice is an upper and a lower quasi-lattice.

Definition 1 (Complete quasi-lattice) *A partially ordered set is a* complete quasi-lattice *(in the sense of sets) if for all non empty subsets $S \subseteq E$, $\sqcap S$ exists whenever $\downarrow S \neq \emptyset$ and $\sqcup S$ exists whenever $\uparrow S \neq \emptyset$.*

2.2 Labels

As mentioned in the introduction, we are interested in type languages allowing subtyping relations between type constructors of different arities, like $list(\alpha) \leq term$ for instance. In general, such subtyping relations specify subtyping relations between specific arguments of the type constructors. For instance, by writing $k_1(\alpha, \beta) \leq k_2(\beta)$, we specify that types built with k_1 are subtypes of the ones built with k_2 when the second argument of k_1 and the argument of k_2 correspond, the first argument of k_1 being forgotten in the subtype relationship.

From a formal point of view, it is simpler (and more general) to express the relationship between arguments by working with a structure of labeled terms. In the formalism of Pottier [8], each argument of a constructor is indicated by a label instead of a position. Moreover, positive and negative labels are distinguished in order to express the covariance or the contravariance of arguments w.r.t. the subtyping relation.

So let $\mathcal{L}^+$ and $\mathcal{L}^-$ be two disjoint countable sets of *labels*, we note $\mathcal{L} = \mathcal{L}^+ \uplus \mathcal{L}^-$. Let $(\mathcal{K}, \leq_\mathcal{K})$ be a complete quasi-lattice of type constructors. Let a be the *arity* function defined from $\mathcal{K}$ into the finite parts of $\mathcal{L}$. We denote by a^+ (resp. a^-) the function which associates the positive (resp. negative) labels to a constructor. We assume that there is at least one type constructor with an empty arity, k_0.

Definition 2 *$(\mathcal{K}, \leq_\mathcal{K}, \mathcal{L}^+, \mathcal{L}^-, a)$ is a* signature *if:*

1. *for all $k_1 \leq_\mathcal{K} k_2 \leq_\mathcal{K} k_3$, $a(k_1) \cap a(k_3) \subseteq a(k_2)$.*
2. *for all $S \subseteq \mathcal{K}$, if $\sqcap S$ exists, then $a(\sqcap S) \subseteq \bigcup_{s \in S} a(s)$.*
3. *for all $S \subseteq \mathcal{K}$, if $\sqcup S$ exists, then $a(\sqcup S) \subseteq \bigcup_{s \in S} a(s)$.*
4. *for all $k_1 \leq_\mathcal{K} k_2$, there exists k s.t. $k_1 \leq_\mathcal{K} k \leq_\mathcal{K} k_2$ and $a(k) = a(k_1) \cap a(k_2)$.*

Conditions 1, 2, 3 express the coherence of labels w.r.t. the order relation and are similar to the ones found in [8] for lattices. Condition 4 is specific to quasi-lattices, its purpose is to forbid signatures like $k_1(\alpha) \leq_{\mathcal{K}} k_2(\beta)$ which do not induce a quasi-lattice structure for types. For example, if k_3 and k_4 are not comparable, then $k_2(k_3)$ and $k_2(k_4)$ have common lower bounds, like $k_1(k_3)$ and $k_1(k_4)$, but don't have a greatest common lower bound.

For a signature $(\mathcal{K}, \leq_{\mathcal{K}}, \mathcal{L}^+, \mathcal{L}^-, a)$, we note $\mathcal{L}^*$ the set of finite strings of labels, ϵ the empty string, "." the string concatenation and $|w|$ the length of w. We are interested in (possibly infinite) types formed upon $\mathcal{K}$, where the positions of subterms are defined by strings of labels.

Definition 3 *Let $(\mathcal{K}, \leq_{\mathcal{K}}, \mathcal{L}^+, \mathcal{L}^-, a)$ be a signature. A (possibly infinite)* type *is a partial mapping t from $\mathcal{L}^*$ into $\mathcal{K}$ such that:*

1. *Its domain is prefix closed: $\forall w = w_1.w_2 \in dom(t), w_1 \in dom(t)$.*

2. *$\epsilon \in dom(t)$.*

3. *For all positions $w \in dom(t)$, for all $l \in \mathcal{L}$, $w.l \in dom(t)$ if and only if $l \in a(t(w))$.*

We note $\mathcal{T}(\mathcal{S})$ the set of (possibly infinite) types built upon the signature $\mathcal{S}$. In the following, we assume a fixed signature $\mathcal{S} = (\mathcal{K}, \leq_{\mathcal{K}}, \mathcal{L}^+, \mathcal{L}^-, a)$ and we note $\mathcal{T} = \mathcal{T}(\mathcal{S})$ the set of types built upon $\mathcal{S}$. We note t/w the type $t' : v \mapsto t(w.v)$, that is the subterm occuring at position w in t. We note U/l, the set of subterms of types in $U \subseteq \mathcal{T}$ occuring at position $l \in \mathcal{L}$, that is $U/l = \{t/l \mid t \in U \wedge l \in a(t(\epsilon))\}$.

Example 1 *We shall use the following example of quasi-lattice of type constructors given with their labels, $\{k_0,\ k_1,\ k_2(l_1, l_2, l_3),\ k_3(l_2, l_3),\ k_4(l_2),\ k_5(l_3)\}$, where $\mathcal{L}^+ = \{l_1, l_2, l_3\}$, $\mathcal{L}^- = \emptyset$, and the subtyping relation is pictured out as follows:*

$$\begin{array}{ccccc} & & & k_2(l_1,l_2,l_3) & \\ & & & | & \\ k_0 & & k_1 & k_3(l_2,l_3) & \\ & & & / \quad \backslash & \\ & & k_4(l_2) & & k_5(l_3) \end{array}$$

Definition 4 *A type constructor $k' \in \mathcal{K}$ is* a lower (resp. upper) bound of another constructor $k \in \mathcal{K}$ w.r.t. a set of labels $L \subseteq \mathcal{L}$ *if $k' \leq_{\mathcal{K}} k$ (resp. $k \leq_{\mathcal{K}} k'$) and $a(k) \cap a(k') \subseteq L$.*

We note $\downarrow_L k$ (resp. $\uparrow_L k$) the set of lower (resp. upper) bounds of k w.r.t. L. In example 1, we have $\downarrow_{\{l_2,l_3\}} k_2 = \{k_3, k_4, k_5\}$ and $\uparrow_{\{l_3\}} k_3 = \emptyset$. Next, we define the subset of labels of k occuring in $\downarrow_L k$:

Definition 5 *For a set of labels $L \subseteq \mathcal{L}$, the subset of* significant labels *of L under (resp. over) k is the set*

$$SL{\downarrow_L}k = a(k) \cap \bigcup_{k' \in \downarrow_L k} a(k')$$

(resp. $SL{\uparrow_L}k = a(k) \cap \bigcup_{k' \in \uparrow_L k} a(k')$).

In example 1, we have $SL{\downarrow}_{\{l_1,l_2\}}k_2 = \{l_2\}$ and $SL{\uparrow}_{\{l_1,l_2\}}k_2 = \emptyset$. One can easily check using the conditions of the definition 2 of a signature the following:

Proposition 1 *If $\downarrow_L k \neq \emptyset$, then $\downarrow_L k$ has a maximum $\sqcup\downarrow_L k$ and $a(\sqcup\downarrow_L k) = SL{\downarrow}_L k$. If $\uparrow_L k \neq \emptyset$, then $\uparrow_L k$ has a minimum $\sqcap\uparrow_L k$ and $a(\sqcap\uparrow_L k) = SL{\uparrow}_L k$.*

2.3 Subtype ordering

The subtyping relation $\leq$ is defined over types, as the intersection of a sequence $(\leq_n)$ of preorders over types defined by:

- $\leq_0 = \mathcal{T} \times \mathcal{T}$
- $t \leq_{n+1} t'$ if $t(\epsilon) \leq_{\mathcal{K}} t'(\epsilon)$ and for all labels $l \in a(t(\epsilon)) \cap a(t'(\epsilon))$:
  - either $l \in \mathcal{L}^+$ and $t/l \leq_n t'/l$
  - or $l \in \mathcal{L}^-$ and $t'/l \leq_n t/l$
- $\leq\, = \bigcap_{n \in \mathbb{N}} \leq_n$

Proposition 2 *$\leq$ is an order over $\mathcal{T}$.*

Proof . We show by induction that for all $n \in \mathbb{N}$, $\leq_n$ is a preorder and we deduce the reflexivity and the transitivity of $\leq$. To show the antisymmetry, we first show by induction that for all $n \in \mathbb{N}$, if $t_1 \leq_{n+1} t_2$ and $t_2 \leq_{n+1} t_1$ then for all positions $w \in dom(t_1)$ of length $|w| \leq n$, we have $w \in dom(t_2)$ and $t_1(w) = t_2(w)$. Now let us consider $t_1 \leq t_2 \leq t_1$: if $t_1 \neq t_2$, then there exists a w of minimal size such that $t_1(w) \neq t_2(w)$. However $t_1 \leq_{|w|+1} t_2 \leq_{|w|+1} t_1$, so $t_1(w) = t_2(w)$, a contradiction. □

Similarly, the subtype ordering can be characterized by:

Proposition 3 *Let $t_1, t_2 \in \mathcal{T}$. $t_1 \leq t_2$ if and only if $t_1(\epsilon) \leq_{\mathcal{K}} t_2(\epsilon)$ and for all labels $l \in a(t_1(\epsilon)) \cap a(t_2(\epsilon))$:*

- *either $l \in \mathcal{L}^+$ and $t_1/l \leq t_2/l$*
- *or $l \in \mathcal{L}^-$ and $t_2/l \leq t_1/l$*

Now, our goal is to show that this ordered set of types forms a quasi-lattice. First we define the set of usable labels under a set of types S as the set of labels l such that S/l has a lower bound:

Definition 6 *The* set of usable labels under a set of types $S \subseteq \mathcal{T}$ *is the set*

$$UL{\downarrow}S = \{l \in \mathcal{L}^+ \mid \downarrow(S/l) \neq \emptyset\} \cup \{l \in \mathcal{L}^- \mid \uparrow(S/l) \neq \emptyset\}$$

The set of usable labels above S *is the set*

$$UL{\uparrow}S = \{l \in \mathcal{L}^+ \mid \uparrow(S/l) \neq \emptyset\} \cup \{l \in \mathcal{L}^- \mid \downarrow(S/l) \neq \emptyset\}$$

For example, with the types $t = k_2(k_0, k_1, k_4(k_0))$ and $t' = k_3(k_1, k_5(k_1))$ formed over the constructors of example 1, we have $UL{\downarrow}\{t, t'\} = \{l_1, l_2\}$ and $UL{\uparrow}\{t, t'\} = \{l_1, l_2, l_3\}$. The head constructor of greatest lower bounds and least upper bounds in $\mathcal{T}$ is given by:

Definition 7 *For a set of types $S \subseteq \mathcal{T}$, the* greatest lower bound constructor of S *is the constructor noted* $\sqcap_\epsilon S = \sqcup\downarrow_{(UL\downarrow S)}(\sqcap\{s(\epsilon) \mid s \in S\})$*, the* least upper bound constructor of S *is the constructor noted* $\sqcup_\epsilon S = \sqcap\uparrow_{(UL\uparrow S)}(\sqcup\{s(\epsilon) \mid s \in S\})$*.*

Now, we define a sequence of types that approximates the greatest lower bound of a set of types up to a given depth. The first type of the sequence is an arbitrary type constant of arity $\emptyset$, k_0, which simply plays the role of a place holder[1].

Definition 8 *The* greatest lower (resp. least upper) bound of rank n *of a non empty set $S \subseteq \mathcal{T}$ of types, noted $\sqcap_n S$ (resp. $\sqcup_n S$), is defined by:*

- $\sqcap_0 S = \sqcup_0 S = k_0$
- $(\sqcap_{n+1} S)(\epsilon) = \sqcap_\epsilon S$ *and for all labels $l \in a(\sqcap_\epsilon S)$:*
  - *if $l \in \mathcal{L}^+$ then $(\sqcap_{n+1} S)/l = \sqcap_n(S/l)$*
  - *if $l \in \mathcal{L}^-$ then $(\sqcap_{n+1} S)/l = \sqcup_n(S/l)$*
- $(\sqcup_{n+1} S)(\epsilon) = \sqcup_\epsilon S$ *and for all labels $l \in a(\sqcup_\epsilon S)$:*
  - *if $l \in \mathcal{L}^+$ then $(\sqcup_{n+1} S)/l = \sqcup_n(S/l)$*
  - *if $l \in \mathcal{L}^-$ then $(\sqcup_{n+1} S)/l = \sqcap_n(S/l)$*

For example, let $t = k_4(k_4(k_1))$ and $t' = k_4(k_5(k_1))$ be two types formed over the constructors of example 1. We have $t \sqcup_0 t' = k_0$, $t \sqcup_1 t' = k_4(k_0)$, $t \sqcup_2 t' = k_4(k_3(k_0, k_0))$ and for all $n \geq 3$, $t \sqcup_n t' = k_4(k_3(k_1, k_1))$.

This provides the following construction of the greatest lower bound and the least upper bound of a set of types, showing that $(\mathcal{T}, \leq)$ is a quasi-lattice (Theorem 1).

Definition 9 *Let $\sqcap_\mathcal{T} : \wp(\mathcal{T}) \to (\mathcal{L}^* \to \mathcal{K})$ (resp. $\sqcup_\mathcal{T}$) be a partial mapping defined by:*

$$(\sqcap_\mathcal{T} S)(w) = (\sqcap_{n+1} S)(w) \quad (\text{resp. } (\sqcup_\mathcal{T} S)(w) = (\sqcup_{n+1} S)(w)\ )$$

for all non empty sets of types $S \subseteq \mathcal{T}$, for all $n \in \mathbb{N}$, for all positions $w \in dom(\sqcap_{n+1}(S))$ (resp. $\sqcup_{n+1}$) such that $|w| = n$.

Using the types of the previous example, we have $t \sqcup_\mathcal{T} t' = k_4(k_3(k_1, k_1))$.

Proposition 4 *Let $S \neq \emptyset \subseteq \mathcal{T}$. If ${\downarrow}S \neq \emptyset$ (resp. ${\uparrow}S \neq \emptyset$) then $\sqcap_\mathcal{T} S$ (resp. $\sqcup_\mathcal{T} S$) is well defined and is a type.*

[1] In the proofs, k_0 is compared to other types through the relation $\leq_0$ which is equal to $\mathcal{T} \times \mathcal{T}$. This means that the type k_0 does not need to be a subtype or a supertype of any other type and that it may be replaced in the definition by any arbitrary type.

Proposition 5 *Let $S \neq \emptyset \subseteq \mathcal{T}$ such that $\downarrow S \neq \emptyset$ (resp. $\uparrow S \neq \emptyset$). Then for all $s \in S, \sqcap_{\mathcal{T}} S \leq s$ (resp. $s \leq \sqcup_{\mathcal{T}} S$).*

Proposition 6 *Let $S \neq \emptyset \subseteq \mathcal{T}$ such that $\downarrow S \neq \emptyset$ (resp. $\uparrow S \neq \emptyset$). For all $t \in \downarrow S, t \leq \sqcap_{\mathcal{T}} S$ (resp. for all $t \in \uparrow S, \sqcup_{\mathcal{T}} S \leq t$).*

Theorem 1 *$(\mathcal{T}, \leq)$ is a complete quasi-lattice, where $\sqcap_{\mathcal{T}}$ denotes greatest lower bounds and $\sqcup_{\mathcal{T}}$ denotes least upper bounds.*

Proof. Let $S \neq \emptyset \subseteq \mathcal{T}$. If S has a lower bound then, by proposition 4, $\sqcap_{\mathcal{T}} S$ exists. By proposition 5, for all $s \in S, \sqcap_{\mathcal{T}} S \leq s$ and by proposition 6, for all $t \in \downarrow S, t \leq \sqcap_{\mathcal{T}} S$. So $\sqcap_{\mathcal{T}} S$ is the greatest lower bound of S. Similarly, we show that if S has an upper bound, then $\sqcup_{\mathcal{T}} S$ is defined and is the least upper bound of S. So $(\mathcal{T}, \leq)$ is a quasi-lattice. $\square$

Concerning the subset $\mathcal{R} \subseteq \mathcal{T}$ of regular types (*i.e.* types having a finite number of subterms), we have the following:

Proposition 7 *Let t_1 and t_2 be two regular types. If $t_1 \sqcap_{\mathcal{T}} t_2$ is defined, then it is a regular type. If $t_1 \sqcup_{\mathcal{T}} t_2$ is defined, then it is a regular type.*

Theorem 2 *$(\mathcal{R}, \leq)$ is a quasi-lattice.*

Proof. By theorem 1, $(\mathcal{T}, \leq)$ is a quasi-lattice. By proposition 7, if $r_1, r_2 \in \mathcal{R}$ and $\exists r, r \leq r_1 \wedge r \leq r_2$ (resp. $r_1 \leq r \wedge r_2 \leq r$) then $r_1 \sqcap_{\mathcal{T}} r_2 \in \mathcal{R}$ (resp. $r_1 \sqcup_{\mathcal{T}} r_2 \in \mathcal{R}$). So $(\mathcal{R}, \leq)$ is a quasi-lattice where $\sqcap_{\mathcal{T}}$ denotes greatest lower bounds and $\sqcup_{\mathcal{T}}$ denotes least upper bounds. $\square$

It is worth noting however that $(\mathcal{R}, \leq)$ may not be a *complete* quasi-lattice. For example, let $\mathcal{K} = \{a, b\}$ with $a \leq_{\mathcal{K}} b$ and $a(a) = a(b) = \{l\}$. Let $(u_n)_{n \geq 0}$ be the sequence of types defined by $u_n(w) = b$ if $|w| = \frac{n(n+1)}{2}$, and $u_n(w) = a$ otherwise. One can check that $(u_n)_{n \geq 0}$ has an upper bound $b(b(b(\ldots)))$, but no lowest upper bound in $\mathcal{R}$ (although it has one in $\mathcal{T}$: the type t defined by $t(w) = b$ if $\exists n, |w| = \frac{n(n+1)}{2}$ and $t(w) = a$ otherwise).

3 Testing the satisfiability of subtyping constraints

Let $\mathcal{V}$ be a countable set of variables, noted $\alpha, \beta, \ldots$. Types with variables are defined as the set, noted $\mathcal{T}_{\mathcal{V}}$, of (possibly infinite) types built upon the signature $(\mathcal{K} \cup \mathcal{V}, \leq_{\mathcal{K}}, \mathcal{L}^+, \mathcal{L}^-, a)$. A subtyping constraint is a pair of *finite* types t_1 and t_2 in $\mathcal{T}_{\mathcal{V}}$ and is noted $t_1 \leq t_2$. For a system C of subtyping constraints, we note $V(C)$ the set of variables occuring in C.

Definition 10 *A substitution $\rho : \mathcal{V} \rightarrow \mathcal{T}$* satisfies *the constraint $t_1 \leq t_2$, noted $\rho \models t_1 \leq t_2$, if $\rho(t_1) \leq \rho(t_2)$. The subtyping constraint $t_1 \leq t_2$ is* satisfiable *if there exist a substitution ρ such that $\rho \models t_1 \leq t_2$.*

For the sake of simplicity, we will suppose, without loss of generality, that the constraint systems considered contain only flat terms. A *flat term* is either a variable, a constant, or a term of depth 1 where all leaves are variables. For example, int, $list(\alpha)$ and α are flat terms while $list(int)$ is not. Clearly, given a constraint system, one can find an equivalent constraint system where all terms are flat terms, by introducing variables for arguments of terms that are not flat terms, and by introducing equality (double inequality) constraints between these variables and the corresponding arguments.

3.1 Closed systems

We first define pre-closed systems as constraint systems where variables are bounded. We recall in table 1 the partial function dec used for breaking constraints in Trifonov and Smith's algorithm [12].

Definition 11 (Pre-closed systems) *A constraint system c is said to be* upper pre-closed *if for all variable $\alpha \in V(C)$, there exists $t \notin \mathcal{V}$ such that $t \leq \alpha \in C$. C is said to be* lower pre-closed *if for all $\alpha \in V(C)$, there exists $t \notin \mathcal{V}$ such that $\alpha \leq t \in C$. A constraint system is said to be* pre-closed *if it is upper and lower pre-closed.*

$$\begin{array}{rcl} dec(\alpha \leq \beta) & = & \{\alpha \leq \beta\} \\ dec(\alpha \leq t) & = & \{\alpha \leq t\} \\ dec(t \leq \alpha) & = & \{t \leq \alpha\} \\ \multicolumn{3}{l}{\text{if } t_1(\epsilon) \leq_{\mathcal{K}} t_2(\epsilon) \text{then :}} \\ dec(t_1 \leq t_2) & = & \displaystyle\bigcup_{l \in a^+(t_1(\epsilon)) \cap a^+(t_2(\epsilon))} \{t_1/l \leq t_2/l\} \cup \bigcup_{l \in a^-(t_1(\epsilon)) \cap a^-(t_2(\epsilon))} \{t_2/l \leq t_1/l\} \end{array}$$

Table 1: Trifonov and Smith's partial decomposition function dec [9, 12]. α, β denote type variables and t, t_1, t_2 denote non variable types.

Definition 12 (Closed system) *A constraint system C is* closed *if it is pre-closed and if for all constraints $c \in C$, $dec(c)$ is defined and included in C and for all $\{t_1 \leq \alpha, \alpha \leq t_2\} \subseteq C$, $dec(t_1 \leq t_2)$ is defined and included in C.*

The application of the decomposition function on non variable types is used to enforce the presence of the corresponding inequalities between their arguments. For example, in a closed constraint system C, if $list(\alpha) \leq list(\beta) \in C$ then $\alpha \leq \beta \in C$. The last condition is used to enforce the transitivity of the constraint system.

In the case of lattices, the substitution ρ such that for all variables $\alpha \in V(C)$, $\rho(\alpha) = \sqcup\rho(\downarrow\alpha)$ is defined and is a solution to C. However, in the case of quasi-lattices, the choice of the head constructor of a lowest upper bound depends on the existence of a lowest upper bound for each argument[2], and such a solution can not be easily defined. For example, let us consider the type constructors $list$, $nhlist$ and int with $a(list) = \{l\}$ and $list \leq_{\mathcal{K}} nhlist$, and the following pre-closed constraint system $C = \{list(\beta) \leq \alpha, list(\delta) \leq \alpha, int \leq \beta \leq int, nhlist \leq \delta \leq nhlist\}$. The only solution of this system is $\rho : \alpha \mapsto nhlist, \beta \mapsto int, \delta \mapsto nhlist$, in particular we have $\rho(\alpha)(\epsilon) = nhlist \neq \sqcap\{list\} = list$. Such solutions are constructed in the proof of theorem 3 below.

Some technical notions are necessary. For a variable α, let $\Uparrow_C \alpha = \{t \mid t \notin \mathcal{V}, \alpha \leq t \in C\}$ be the set of upper bounds of α in C, and let $\Downarrow_C \alpha = \{t \mid t \notin \mathcal{V}, t \leq \alpha \in C\}$ be the set lower bounds α in C. For a set of variables A, we note $\Uparrow_C A = \bigcup_{\alpha \in A} \Uparrow_C \alpha$ the set of types that are upper bound of an element of A in C, and $\Downarrow_C A = \bigcup_{\alpha \in A} \Downarrow_C \alpha$ the set of types that are a lower bound of an element of A in C. By abuse of notation, when C is clear from the context, we will omit C in the notations.

Definition 13 *Given a constraint system C, let* $sol : \wp(V(C)) \times \wp(V(C)) \to \mathcal{K}$ *be the partial function defined by* $sol(A, B) = \sqcup(\downarrow_{a(\sqcup D)}(\sqcap U))$ *when it exists, where* $U = \{t(\epsilon) \mid t \in \Uparrow B\}$ *and* $D = \{t(\epsilon) \mid t \in \Downarrow A\}$.

Lemma 1 *In a closed system C,* $sol(A, B)$ *is defined for all non empty sets* A, B *such that* $\forall \alpha \in A$, $\forall \beta \neq \alpha \in B$, $\alpha \leq \beta \in C$.

Proof. Let $t \in \Downarrow A$ and $t' \in \Uparrow B$. Since C is closed, $dec(t \leq t')$ is defined, hence $t(\epsilon) \leq_{\mathcal{K}} t'(\epsilon)$. So $\forall k \in D, \forall k' \in U, k \leq_{\mathcal{K}} k'$. Since C is closed, $\Downarrow A \neq \emptyset$ and $\Uparrow B \neq \emptyset$. So $\sqcup D$ and $\sqcap U$ are defined and $\sqcup D \leq_{\mathcal{K}} \sqcap U$. Therefore $\downarrow_{a(\sqcup D)}(\sqcap U) \neq \emptyset$ and $sol(A, B)$ is well defined.

Lemma 2 *Let C be a closed constraint system and* $A, B \subseteq V(C)$ *verifying the conditions of lemma 1. For all labels* $l \in a(sol(A, B))$*, if* $l \in \mathcal{L}^+$ *then* $(\Downarrow A/l, \Uparrow B/l)$ *satisfies the condition of lemma 1, and if* $l \in \mathcal{L}^-$ *then* $(\Uparrow B/l, \Downarrow A/l)$ *satisfies the condition of lemma 1.*

Lemma 3 *Let C be a closed constraint system. Let* $A, B, E, F \subseteq V(C)$*. If* $\Downarrow A \subseteq \Downarrow E$ *and* $\Uparrow F \subseteq \Uparrow B$ *and* A, B *and* E, F *satisfy the conditions of lemma 1, then* $sol(A, B) \leq_{\mathcal{K}} sol(E, F)$.

Theorem 3 *In a quasi-lattice, any closed constraint system is satisfiable.*

Proof. Let C be a closed constraint system. Let us consider the partial mapping $\Gamma : \wp(V(C)) \times \wp(V(C)) \times \mathcal{L}^* \to \wp(V(C)) \times \wp(V(C))$ defined for couples (A, B) satisfying the conditions of lemma 1 as follows: $\Gamma(A, B, \epsilon) = (A, B)$, if $\Gamma(A, B, w) = (A', B')$ then, for all labels $l \in a(sol(A', B')), l \in \mathcal{L}^+, \Gamma(A, B, w.l) = ((\Downarrow A')/l, (\Uparrow B')/l)$ and if $l \in \mathcal{L}^-, \Gamma(A, B, w.l) = ((\Uparrow B')/l, (\Downarrow A')/l)$. Let us consider $\gamma : \wp(V(C)) \times \wp(V(C)) \to \mathcal{T}$ defined by $\gamma(A, B)(w) = sol(\Gamma(A, B, w))$. By induction, using lemma 2, one can check that $\gamma(A, B)$ is a type. Now, let us consider the substitution $\rho(\alpha) = \gamma(\{\alpha\}, \{\alpha\})$.

We show that $\rho \models C$. By induction, we show that $\forall n \in \mathbb{N}, \forall t_1 \leq t_2 \in C, \rho(t_1) \leq_n \rho(t_2)$ and for all $A, B, E, F \in V(C)$ satisfying the conditions of lemma 3, $\gamma(A, B) \leq_n \gamma(E, F)$.

The case $n = 0$ is trivially true. Now, we show the case $n + 1$.

[2]or a greatest lower bound in the case of contravariant arguments

Let us consider the case $t_1 = \gamma(A, B) \leq_{n+1} \gamma(E, F) = t_2$. By lemma 3, $k_1 = \gamma(A, B)(\epsilon) \leq_{\mathcal{K}} \gamma(E, F)(\epsilon) = k_2$. Let $l \in a(k_1) \cap a(k_2)$, $l \in \mathcal{L}^+$. Since $\Downarrow A \subseteq \Downarrow E$, $(\Downarrow A)/l \subseteq (\Downarrow E)/l$ and thus $\Downarrow((\Downarrow A)/l) \subseteq \Downarrow((\Downarrow E)/l)$. Similarly $\Uparrow((\Uparrow F)/l) \subseteq \Uparrow((\Uparrow B)/l)$. So, using the induction hypothesis, $t_1/l = \gamma((\Downarrow A)/l, (\Uparrow B)/l) \leq_n \gamma((\Downarrow E)/l, (\Uparrow F)/l) = t_2/l$. Similarly, if $l \in \mathcal{L}^-$, $t_2/l = \gamma((\Uparrow F)/l, (\Downarrow E)/l) \leq_n \gamma((\Uparrow B)/l, (\Downarrow A)/l) = t_1/l$. Thus we deduce $t_1 \leq_{n+1} t_2$.

Let us consider $\rho(\alpha) \leq \rho(\beta)$. Since $\alpha \leq \beta \in C$ and since C is closed, $\Downarrow\alpha \subseteq \Downarrow\beta$ and $\Uparrow\beta \subseteq \Uparrow\alpha$. So we can apply the preceding result obtaining $\rho(\alpha) = \gamma(\{\alpha\}, \{\alpha\}) \leq_{n+1} \gamma(\{\beta\}, \{\beta\}) = \rho(\beta)$.

Let us consider $\alpha \leq t$, with $t \notin \mathcal{V}$. Since $t \in \Uparrow\alpha$, $\rho(\alpha)(\epsilon) = sol(\{\alpha\}, \{\alpha\}) \leq_{\mathcal{K}} \sqcap\{t'(\epsilon) \mid t' \in \Uparrow\alpha\} \leq_{\mathcal{K}} t(\epsilon)$. Let $l \in a(\rho(\alpha)(\epsilon)) \cap a(t(\epsilon))$, $l \in \mathcal{L}^+$. Since we use flat terms, t/l is a variable β, thus $\rho(t)/l = \gamma(\{\beta\}, \{\beta\})$ and $\rho(\alpha)/l = \gamma((\Downarrow\alpha)/l, (\Uparrow\alpha)/l)$. Since $\beta \in (\Uparrow\alpha)/l$, $\Uparrow\beta \subseteq \Uparrow((\Uparrow\alpha)/l)$. Since C is closed, for all $\alpha' \in (\Downarrow\alpha)/l, \alpha' \leq \beta \in C$. Since C is closed, we have $\Downarrow((\Downarrow\alpha)/l) \subseteq \Downarrow\beta$. Thus we obtain $\rho(\alpha)/l = \gamma((\Downarrow\alpha)/l, (\Uparrow\alpha)/l) \leq_n \gamma(\{\beta\}, \{\beta\}) = \rho(t)/l$. Similarly, if $l \in \mathcal{L}^-$, $\rho(t)/l = \gamma(\{\beta\}, \{\beta\}) \leq_n \gamma((\Uparrow\alpha)/l, (\Downarrow\alpha)/l) = \rho(\alpha)/l$. So $\rho(\alpha) \leq_{n+1} \rho(t)$

Similarly, we show the case $t \leq \alpha$.

The last case is $t_1 \leq t_2$ with $t_1, t_2 \notin V(C)$. Since C is closed $t_1(\epsilon) \leq_{\mathcal{K}} t_2(\epsilon)$. Let $l \in a(t_1(\epsilon)) \cap a(t_2(\epsilon))$, $l \in \mathcal{L}^+$. Since C is closed, posing $\alpha = t_1/l$ and $\beta = t_2/l$, we have $\alpha \leq \beta \in C$ and, using the induction hypothesis, $\rho(t_1)/l = \rho(\alpha) \leq_n \rho(\beta) = \rho(t_2)/l$. Similarly, if $l \in \mathcal{L}^-$, we obtain $\rho(t_2)/l \leq_n \rho(t_1)/l$. Thus $\rho(t_1) \leq_{n+1} \rho(t_2)$.

So for all $n \in \mathbb{N}$, for all $t_1 \leq t_2 \in C$, $\rho(t_1) \leq_n \rho(t_2)$, *i.e.* $\rho(t_1) \leq \rho(t_2)$. Thus $\rho \models C$. $\square$

Given a pre-closed system C, one can compute its closure, as in Trifonov and Smith's algorithm [12] in $O(n^3)$. The algorithm proceeds by computing the sequence $C, C1, C2, \ldots$ defined by:

$$C^{n+1} = C^n \cup \bigcup_{c \in C^n} dec(c) \cup \bigcup_{\{t \leq \alpha, \alpha \leq t'\} \subseteq C^n} dec(t \leq t')$$

If dec is not defined for some constraint in C^n then the constraint is not satisfiable, that is C^n has no solution. Otherwise, for any C^n, C^{n+1} is defined and it is clearly equivalent to C^n. In this case, the sequence reaches a fix point which is closed, hence satisfiable by theorem 3, and equivalent to C, so C is satisfiable.

Corollary 1 *A pre-closed constraint system C is satisfiable in infinite types in and only if it is satisfiable in regular types.*

Proof. If C is satisfiable, then there exist a closed system C' equivalent to C. The substitution used in the proof theorem 6 above is a regular type. So C' (and thus C) has a solution in the set of regular types.

3.2 Pre-closure algorithm

The algorithm above requires a pre-closed system as an entry. This condition is automatically filled in lattices since there exists a maximal type $\top$ and a minimal type $\bot$. In this case, it is sufficient to add constraints $\bot \leq \alpha$ and $\alpha \leq \top$ to obtain a pre-closed system with the same solutions [9, 12]. In quasi-lattices, the theorem 4 below provides sufficient conditions over $\mathcal{K}$ for deciding the satisfiability of a non pre-closed constraint system. Let $\overline{\mathcal{K}}$ be the set of maximal elements of $\mathcal{K}$ and $\underline{\mathcal{K}}$ the set of its minimal elements.

Theorem 4 *If $\mathcal{K}$ verifies the following conditions:*

1. $\forall k \in \overline{\mathcal{K}} \cup \underline{\mathcal{K}}, a(k) = \emptyset$
2. *For all $k \in \mathcal{K}$, there exists $k_1 \in \underline{\mathcal{K}}$ and $k_2 \in \overline{\mathcal{K}}$ such that $k_1 \leq_{\mathcal{K}} k \leq_{\mathcal{K}} k_2$.*

For any constraint system C let the set of pre-closures $pc(C)$ be:

$$pc(C) = \left\{ C \cup \bigcup_{\alpha \in V(C)} \{t_\alpha \leq \alpha, \alpha \leq t'_\alpha\} \mid t_\alpha(\epsilon) \in \underline{\mathcal{K}}, t'_\alpha(\epsilon) \in \overline{\mathcal{K}} \right\}.$$

All elements in $pc(C)$ are closed and the union of their sets of solutions is equal to the set of solutions of C.

Proof. Since $\forall k \in \overline{\mathcal{K}} \cup \underline{\mathcal{K}}, a(k) = \emptyset$, for all $C' \in pc(C)$, $V(C') = V(C)$. So, by construction, the elements of $pc(C)$ are pre-closed. For all $C' \in pc(C)$, we have $C \subseteq C'$, thus $\rho \models C' \Rightarrow \rho \models C$. Now we show that if $\rho \models C$ then there exists $C' \in pc(C)$ such that $\rho \models C'$. By condition 2) for all $\alpha \in V(C)$, one can find $k_\alpha \in \underline{\mathcal{K}}$ and $k'_\alpha \in \overline{\mathcal{K}}$ such that $\rho \models t \leq \alpha, \alpha \leq t'$ with $t(\epsilon) = k_\alpha$ and $t'(\epsilon) = k'_\alpha$. Thus there exists $C' \in pc(C)$ such that $\rho \models C'$. Thus the union of the sets of solutions of the elements of $pc(C)$ is equal to the set of solutions of C. □

If $\overline{\mathcal{K}}$ and $\underline{\mathcal{K}}$ are finite sets, it is possible to enumerate the elements of $pc(C)$. Since these elements are pre-closed, one can test their satisfiability using the closure algorithm of the previous section. This gives an algorithm for testing the satisfiability of non-closed constraint systems in quasi-lattices with a finite number of extrema each with an empty arity. The time complexity of the satisfiability test is in $O(n^3 m^v M^v)$ where n is the size of the constraint system, m is the size of $\underline{\mathcal{K}}$ and M the size of $\overline{\mathcal{K}}$, and v is the number of unbounded variables.

Theorem 5 *The satisfiability problem for subtyping constraints in quasi-lattices with a finite number of extrema each with an empty arity is NP-complete.*

Proof. The satisfiability of pre-closed system is polynomial. By theorem 4 the set of pre-closures of a system can be guessed by enumerating the possible bounds for unbounded variable among a finite set, hence the satisfiability problem is in NP. To prove the NP-completeness, we use the result of Pratt and Tiuryn [10] that the satisfiability of subtyping constraints in n-crowns is NP-complete for $n \geq 2$. An n-crown is a poset with $2n$ elements $k_0, \ldots, k_{2n-1}$, all with an empty arity and partially ordered such that the only comparisons are $k_{2i} \leq_{\mathcal{K}} k_{2i\pm1}$ and $k_0 \leq_{\mathcal{K}} k_{2n-1}$. Clearly, n-crowns with $n \geq 3$ are quasi-lattices with a finite number of extrema each with an empty arity. The satisfiability problem in quasi-lattices is thus NP-complete. □

The first condition imposed on K in theorem 4 expresses that the extrema in the quasi-lattice of constructors have an empty arity. Without this condition, it is worth noting that the introduction of a new constraint $t_\alpha \leq \alpha$ (or $\alpha \leq t'_\alpha$) may also introduce some new unbounded variables appearing in t, that must be bounded by introducing new constraints, which leads to introduce an infinity of variables. Thus, the above algorithm cannot be used in that case. Our result thus lefts open the decidability of the satisfiability of non-structural subtyping constraints in quasi-lattices where some extrema have a non-empty arity.

4 Computing explicit solutions

Although testing the satisfiability of subtyping constraints is sufficient for type checking, type inference requires to exhibit a solution of a constraint system, not just check the existence of a solution.

In lattices, Pottier [8] describes an algorithm for simplifying subtyping constraint systems and computing explicit solutions, by identifying type variables to their bounds. This algorithm transforms a constraint system C into a canonical constraint system noted $Can(C)$. We extend here this algorithm to the case of quasi-lattices by adding some specific simplification.

Let us assume, as in section 3, that constraints are formed upon flat terms and that the constraint system to be simplified is pre-closed. In order to solve a constraint system C in a quasi-lattice of types $\mathcal{T}(\mathcal{K})$, the set of constructors $\mathcal{K}$ is completed in a lattice $\mathcal{K}^{\perp,\top}$ by adding $\perp$ and $\top$ elements with an empty arity and $\perp \leq_{\mathcal{K}^{\perp,\top}} k \leq_{\mathcal{K}^{\perp,\top}} \top$ for all $k \in \mathcal{K}$. The constraint system C is first solved in the lattice of types $\mathcal{T}(\mathcal{K}^{\perp,\top})$ by computing $Can(C)$. Then, another set of rules (given in Table 3 is applied over $Can(C)$, in order to compute a solution in the original quasi-lattice.

Pottier's algorithm may introduce variables for representing the greatest lower or least upper bounds of a set of original variables in C. For a set A of variables in C, let us note γ_A the variable representing the greatest lower bound of A and λ_A the variable representing its least upper bound. We recall in table 2 some useful properties satisfied by $Can(C)$ [9] which show that $Can(C)$ is closed and thus satisfiable in $\mathcal{T}(\mathcal{K}^{\perp,\top})$.

Let us consider the type constructors int, $string$, $list$ and $nhlist$, with $a(list) = \{l\}$ and $list \leq_{\mathcal{K}} nhlist$, and the constraint system $C = \{list(int) \leq \alpha, list(string) \leq \alpha\}$. A naive algorithm would try to find a lower bound to int and $string$ and thus fail, whilst $\alpha = nhlist$ is a solution to C. Some specific rules must thus be defined to cover such cases. They are given in Table 3. Here, we show how these rules can be applied to the previous example Let us consider the following pre-closed constraint system $C = \{list(\beta) \leq \alpha, list(\delta) \leq \alpha, \alpha \leq nhlist, int \leq \beta \leq int, string \leq \delta \leq string\}$. We have $Can(C) = \{list(\lambda_{\{\beta,\delta\}}) \leq \alpha \leq nhlist, int \leq \beta \leq int, string \leq \delta \leq string, \top \leq \lambda_{\{\beta,\delta\}} \leq \top\}$. By applying the rule (Up $\top$), we obtain $D = \{nhlist \leq \alpha \leq nhlist, int \leq \beta \leq int, string \leq \delta \leq string, \top \leq \lambda_{\{\beta,\delta\}} \leq \top\}$, which has a solution $\alpha = nhlist, \beta = int, \gamma = string$ and $\lambda_{\{\beta,\delta\}} = \top$.

Proposition 8 *The application of the rules of table 3 preserve the solutions which co-domain is included in* $\mathcal{T}(\mathcal{K}) \cup \{\perp, \top\}$.

Proof . Let us consider the case (Down $\perp$), the other cases being similar. Let us assume that $\rho(\alpha) \models D, \alpha \leq t, \gamma_A \leq \perp$ with $dom(\rho) \subseteq \mathcal{T}(\mathcal{K}) \cup \{\perp, \top\}$ and let us show that $\rho \models D, \gamma_A \leq$

1. For all $\alpha \in V(D)$, there exists exactly one type $t \notin \mathcal{V}$ (called the lower bound of α in D, and noted $\Downarrow_D \alpha$) such that $t \leq \alpha \in D$ and exactly one type $t' \notin \mathcal{V}$ (called the upper bound of α in D and noted $\Uparrow_D \alpha$) such that $\alpha \leq t' \in D$.
2. For all $\{\alpha \leq \beta, \beta \leq \delta\} \subseteq D$, $\alpha \leq \delta \in D$.
3. For all $\alpha \in V(D)$, for all labels $l \in a((\Uparrow_D \alpha)(\epsilon))$, either $l \in \mathcal{L}^+$ and $\exists A, t/l = \gamma_A$, or $l \in \mathcal{L}^-$ and $\exists A, t/l = \lambda_A$. For all labels $l \in a((\Downarrow_D \alpha)(\epsilon))$, either $l \in \mathcal{L}^+$ and $\exists A, t/l = \lambda_A$, or $l \in \mathcal{L}^-$ and $\exists A, t/l = \gamma_A$.
4. If $\alpha \leq \beta \in D$, or if $\alpha \equiv \gamma_A$ and $\beta \equiv \gamma_B$ with $B \subseteq A$ then $k_\alpha = (\Uparrow_D \alpha)(\epsilon) \leq_{\mathcal{K}^{\perp,\top}} k_\beta = (\Uparrow_D \beta)(\epsilon)$ and for all labels $l \in a(k_\alpha) \cap a(k_\beta)$, if $l \in \mathcal{L}^+$, $(\Uparrow_D \alpha)/l = \gamma_E$, $(\Uparrow_D \beta)/l = \gamma_F$, then $F \subseteq E$. If $l \in \mathcal{L}^-$, $(\Uparrow_D \alpha)/l = \lambda_E$, $(\Uparrow_D \beta)/l = \lambda_F$, then $F \subseteq E$.
5. If $\alpha \leq \beta \in D$, or if $\alpha \equiv \lambda_A$ and $\beta \equiv \lambda_B$ with $A \subseteq B$ then $k_\alpha = (\Downarrow_D \alpha)(\epsilon) \leq_{\mathcal{K}^{\perp,\top}} k_\beta = (\Downarrow_D \beta)(\epsilon)$ and for all labels $l \in a(k_\alpha) \cap a(k_\beta)$, if $l \in \mathcal{L}^+$, $(\Downarrow_D \alpha)/l = \lambda_E$, $(\Downarrow_D \beta)/l = \lambda_F$, then $E \subseteq F$. If $l \in \mathcal{L}^-$, $(\Downarrow_D \alpha)/l = \gamma_E$, $(\Downarrow_D \beta)/l = \gamma_F$, then $E \subseteq F$.
6. For all variables $\gamma_A, \Uparrow_D \gamma_A \neq \top$ and for all variables $\lambda_A, \Downarrow_D \lambda_A \neq \perp$.
7. For all $\alpha \in V(C)$, $dec(\Downarrow_D \alpha \leq \Uparrow_D \alpha)$ is defined and included in D.
8. $\forall t, t', t \leq t' \notin D$

Table 2: Properties verified by the canonical form $D = Can(C)$ of a constraint system C [9].

(Down $\perp$)	$D, \gamma_A \leq \perp, \alpha \leq t \rightarrow D, \gamma_A \leq \perp, \alpha \leq t'$	if $t/l = \gamma_A$,
	where $t'(\epsilon) = \sqcup(\downarrow_{a(t(\epsilon))\setminus\{l\}} t(\epsilon))$ if it is defined, $t'(\epsilon) = \perp$ otherwise	
	and for all labels $l' \in t'(\epsilon)$, $t'/l' = t/l'$.	
(Down $\top$)	$D, \top \leq \lambda_A, \alpha \leq t \rightarrow D, \top \leq \lambda_A, \alpha \leq t'$	if $t/l = \lambda_A$,
	where $t'(\epsilon) = \sqcup(\downarrow_{a(t(\epsilon))\setminus\{l\}} t(\epsilon))$ if it is defined, $t'(\epsilon) = \perp$ otherwise	
	and for all labels $l' \in t'(\epsilon)$, $t'/l' = t/l'$.	
(Up $\top$)	$D, \top \leq \lambda_A, t \leq \alpha \rightarrow D, \top \leq \lambda_A, t' \leq \alpha$	if $t/l = \lambda_A$,
	where $t'(\epsilon) = \sqcap(\uparrow_{a(t(\epsilon))\setminus\{l\}} t(\epsilon))$ if it is defined, $t'(\epsilon) = \perp$ otherwise.	
	and for all labels $l' \in t'(\epsilon)$, $t'/l' = t/l'$.	
(Up $\perp$)	$D, \gamma_A \leq \perp, t \leq \alpha \rightarrow D, \gamma_A \leq \perp, t' \leq \alpha$	if $t/l = \gamma_A$,
	where $t'(\epsilon) = \sqcap(\uparrow_{a(t(\epsilon))\setminus\{l\}} t(\epsilon))$ if it is defined, $t'(\epsilon) = \perp$ otherwise.	
	and for all labels $l' \in t'(\epsilon)$, $t'/l' = t/l'$.	

Table 3: Additional rules for computing bounds in a quasi-lattice.

$\perp, \alpha \leq t'$. Clearly, $\rho(\gamma_A) = \perp$. Since for all labels $l' \in t'(\epsilon), t/l' = t'/l'$, it is sufficient to show that $\rho(\alpha)(\epsilon) \leq_{\mathcal{K}} t'(\epsilon)$. $t/l = \gamma_A$, so $\rho(t)/l = \perp$. Since there is no type in $\mathcal{T}(\mathcal{K})$ smaller than $\perp$, and by proposition 3, $l \notin a(\rho(\alpha)(\epsilon))$. Thus $\rho(\alpha)(\epsilon) \leq_{\mathcal{K}^{\perp,\top}} \sqcup(\downarrow_{a(t(\epsilon))\setminus\{l\}} t(\epsilon))$ if it exists and

$\rho(\alpha)(\epsilon) \leq_{\mathcal{K}^{\perp,\top}} \perp$ otherwise. Thus $\rho(\alpha)(\epsilon) \leq_{\mathcal{K}^{\perp,\top}} t'(\epsilon)$. $\square$

Proposition 9 *The rules of table 3 terminate and are confluent.*

Proof. By proposition 1, if $D, \alpha \leq t \rightarrow D, \alpha \leq t'$ then $a(t'(\epsilon)) \subset a(t(\epsilon))$, thus one can apply rules (Down $\perp$) and (Down $\top$) only a finite number of times for each variable. As well, one can apply rules (Up $\perp$) and (Up $\top$) only a finite number of times for each variable. Since no rules increases the arity of the variables bounds, the rewriting system terminates in $O(n)$ steps where n is the size of the constraint system. The confluence is obtained by remarking that $\sqcup(\downarrow_{L'}(\sqcup(\downarrow_L k))) = \sqcup(\downarrow_{L\cap L'} k) = \sqcup(\downarrow_L(\sqcup(\downarrow_{L'} k)))$. $\square$

We note $\mathrm{CanQ}(D)$ the quasi-lattice canonical form of a constraint system D by the rules $\rightarrow$ of Table 3.

Proposition 10 *Let C be a constraint system, $C' = Can(C)$ and $C'' = \mathrm{CanQ}(C')$. Then C'' verifies the properties given in table 2.*

Proof. [3] One can check that properties 1), 2), 3), 6), 7) and 8) are conserved during the application of the rules of table 3. If the application of a rule breaks the property 4) or 5), it is possible to reestablish it by applying this rule on some other variables. Since $\rightarrow^*$ is convergent, we obtain that C'' verifies properties 4) and 5). $\square$

Theorem 6 *Let C be a pre-closed constraint system and $C' = \mathrm{CanQ}(Can(C))$. If $\mathcal{L}^- = \emptyset$ then the upper bounds $\Uparrow_{C'}\alpha$ (resp. lower bounds $\Downarrow_{C'}\alpha$) in C' define a maximal (resp. minimal) solution of C in $\mathcal{T}(\mathcal{K})$.*

Proof. Let be the following equation system: $\{\alpha = t \mid \alpha \leq t \in C'\}$. This system admit a unique solution ρ because each variable appears only once on the left hand side of $=$. Let us show that ρ is a solution of C'. In order to do this, we first show by induction that for all $n \in \mathbb{N}$, for all constraints $t \leq t' \in C'$, $\rho(t) \leq_n \rho(t')$, and if $A \subseteq B$, $\rho(\gamma_B) \leq_n \rho(\gamma_A)$. The case $n = 0$ is trivially verified. Let $\alpha \leq \beta \in C'$. By proposition 10, C' verifies the properties of table 2. Thus $\rho(\alpha)(\epsilon) = (\Uparrow_{C'}\alpha)(\epsilon) \leq_{\mathcal{K}} (\Uparrow_{C'}\beta)(\epsilon) = \rho(\alpha)(\epsilon)$. Let us consider $l \in a(\rho(\alpha)(\epsilon)) \cap a(\rho(\beta)(\epsilon))$. By properties 3) and 4) of table 2, $\rho(\alpha)/l = \gamma_A$ for some A and $\rho(\beta)/l = \gamma_B$ for some B with $B \subseteq A$. By induction $\rho(\gamma_A) \leq_n \rho(\gamma_B)$. Thus $\rho(\alpha) \leq_{n+1} \rho(\beta)$. Similarly, if $A \subseteq B$, $\rho(\gamma_B) \leq_{n+1} \rho(\gamma_A)$. Let $\alpha \leq t \in C'$. We have $\rho(\alpha) = \rho(t)$ so $\rho(\alpha) \leq_{n+1} \rho(t)$. Let $t \leq \alpha \in C'$. Using property 7) of table 2, we have $\rho(t)(\epsilon) \leq_{\mathcal{K}} \rho(\alpha)(\epsilon)$. Moreover, for all labels $l \in a(\rho(t)(\epsilon)) \cap a(\rho(\alpha)(\epsilon))$, $t/l \leq \Uparrow_{C'}\alpha \in C$, thus $\rho(t)/l \leq_n \rho(\alpha)/l$. So $\rho(t) \leq_{n+1} \rho(\alpha)$. So for all constraints $t \leq t' \in C'$, $\rho(t) \leq \rho(t')$, thus $\rho \models C'$. Moreover, since $\mathcal{L}^- = \emptyset$, there are only variables of the form γ_A occuring on the right hand side of the system defining ρ, thus, by using property 6) of table 2, the fact that C is pre-closed with bounds in $\mathcal{K}$ and that rule (Down $\perp$) can not be applied to C', we obtain that for all $\alpha \in V(C)$, $\rho(\alpha) \in \mathcal{T}(\mathcal{K})$. Then we check by induction that for all substitutions $\rho' \models C'$, for all variables $\alpha \in V(C')$, for all $n \in \mathbb{N}$, $\rho'(\alpha) \leq_n \rho(\alpha)$, *i.e.* $\rho'(\alpha) \leq \rho(\alpha)$. ρ is thus a maximal solution

[3] a more detailed version of this proof is available in the appendix

of C in $\mathcal{T}(\mathcal{K})$. $\square$

It is worth noting that in the case where some type constructors have contravariant labels (in $\mathcal{L}^-$), there may be no maximal solution. For example, let us take $\mathcal{K} = \{int, float, \rightarrow\}$ with $int \leq float$, $\mathcal{L}^+ = \{r\}$, $\mathcal{L}^- = \{a\}$, $a(int) = a(float) = \emptyset$, $a(\rightarrow) = \{a, r\}$. Let $C = \{\alpha \rightarrow \alpha \leq \beta, \beta \leq \alpha \rightarrow \alpha, int \leq \alpha, \alpha \leq float\}$. C is pre-closed and has two *incomparable* solutions, namely $\rho(\beta) = int \rightarrow int, \rho(\alpha) = int$ and $\rho'(\beta) = float \rightarrow float, \rho'(\alpha) = float$.

In the case where all type constructors are covariant ($\mathcal{L}^- = \emptyset$), our simplification rules thus give maximal and minimal solutions to pre-closed systems. The combination of the above algorithm with the pre-closure algorithm of section 3.2, gives a set of maximal and minimal solutions for non-pre-closed systems in quasi-lattices with a finite number of extrema with empty arities.

5 Implementation and applications

5.1 Performance evaluation

The subtyping constraint algorithms described in this paper have been implemented [3] using the Constraint Handling Rules (CHR) language [6]. The table 4 shows type checking (with type inference for variables) time for 17 SICStus Prolog libraries. The second column corresponds to type checking using the algorithms presented in this paper for solving subtype inequalities in quasi-lattices. The third column corresponds to type checking using Pottier's algorithms for solving subtype inequalities in lattices, which where also implemented in CHR. The last column is the ratio between the first and the second time.

The algorithm for quasi-lattices performs very well in practice since the ratio is comprised between 1.15 and 3.76 and an average ratio of 2.04. Since the algorithms for quasi-lattices need to compute pre-closure of the constraint system, one could expect a bigger difference due to some combinatory explosion. However, the majority of type variables appearing in the constraint system obtained during type checking already have an upper bound. For these variables, it is sufficient to find one lower bound compatible with the existing upper bound and then propagate the information in the constraint system. For the other variables which are completely unbounded, the strategy consisting in delaying bound enumeration enumeration for these variables suffices to avoid the combinatory explosion in practice.

5.2 Applications

The first application of solving non-structural subtyping constraints in quasi-lattices is in our type system TCLP [4, 2] for constraint logic programs (CLP). In this covariant type system, the type of CLP variables is only constrained by upper bounds. Thus, in the case where the type structure forms a lattice, the type inference algorithm can always assign the empty type $\perp$ to variables, which means that no type error can be found on variables. On the other hand, in the case of a quasi-lattice of type constructors, the type inference algorithm detects incompatible types for variables, for example if a variable is constrained to have a type smaller than both int and $list(\alpha)$. Moreover, the structure of

File	Type checking time for quasi-lattice	for lattice	Ratio
arrays	0.78 s	1.73 s	2.21
assoc	2.18 s	5.02 s	2.30
atts	1.9 s	3.21 s	1.68
bdb	3.17 s	5.89 s	1.85
charsio	0.41 s	0.96 s	2.34
clpr	46.85 s	69.63 s	1.48
fastrw	0.21 s	0.36 s	1.71
heaps	1.87 s	4.44 s	2.37
jasper	0.98 s	1.6 s	1.63
lists	1.87 s	3.5 s	1.87
ordsets	2.38 s	5.89 s	2.47
queues	0.43 s	1.03 s	2.39
random	0.8 s	0.92 s	1.15
sockets	1.83 s	4.33 s	2.36
terms	1.35 s	3.25 s	2.40
trees	0.81 s	2.39 s	2.95
ugraphs	14.14 s	53.19 s	3.76

Table 4: Comparison between lattice and quasi-lattice implementations in CHR

quasi-lattice makes it possible to avoid the use of the metaprogramming type $term = \top$ in modules where this type is not supposed to be used.

Another application can be found in the framework of type inference with subtyping for languages *à la* ML. In [8], Pottier uses subtyping constraints for type inference in a variant of ML with rows. However, in a lattice, the bottom element $\perp$ denotes the empty type, hence a function typed by $\perp \rightarrow \tau$ cannot be applied to any argument. The algorithm for solving subtyping constraints described in this paper allows one to use the quasi-lattice obtained by removing the $\perp$ element from the lattice as a type structure. A type error can then be produced instead of a typing with the empty type.

6 Conclusion

We have studied general forms of non-structural subtyping relations in the quasi-lattice of infinite (regular) types formed over a quasi-lattice of type constructors. We have shown the decidability of the satisfiability problem for subtyping constraints in quasi-lattices , by generalizing Trifonov and Smith's algorithm for testing the satisfiability of subtyping constraints in lattices to the case of quasi-lattices, with a time complexity in $O(m^v M^v n^3)$ where m (resp. M) is the number of minimal (resp. maximal) elements of the quasi-lattice and v the number of unbounded variables. It is worth noting that the complexity of this algorithm is in $O(n^3)$ for constraint systems where all variables are bounded. In the general case we have shown the NP-completeness of this problem.

We have also extended Pottier's algorithm for computing solutions to the case of quasi-lattices, and have shown that the computed solutions are minimal (resp. maximal) solutions when all type constructors are covariant. Finally we have mentioned some applications of these algorithms to type inference problems in constraint logic programming and in functional programming languages.

As for future work, one can mention some problems left open in this paper. We have already mentioned the case where the extrema of the quasi-lattice of constructors have a non empty arity. The decidability of constraint satisfiability in finite types is also an open problem. In the homogeneous case (i.e. when the type constructors in a subtype relation have the same arity), Frey has shown that this problem is Pspace complete in arbitrary posets [5].

References

[1] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.

[2] E. Coquery. `Tclp`: a generic type checker for constraint logic programs. `http://pauillac.inria.fr/~coquery/tclp/`.

[3] E. Coquery and F. Fages. From typing constraints to typed constraint systems in CHR. In *Third workshop on Rule-based Constraint Reasoning and Programming, CP'01*, Cyprus, December 2001.

[4] F. Fages and E. Coquery. Typing constraint logic programs. *Theory and Practice of Logic Programming*, 1, November 2001.

[5] A. Frey. Satisfying subtype inequalities in polynomial space. In *Proceedings of the 1997 International Static Analysis Symposium*, number 1302 in LNCS, 1997.

[6] Th. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, 37(1-3):95–138, October 1998.

[7] J. Palsberg, T. Zhao, and T. Jim. Automatic discovery of covariant read-only fields. In *Ninth International Workshop on Foundations of Object-Oriented Languages*, January 2002.

[8] F. Pottier. A versatile constraint-based type inference system. *Nordic Journal of Computing*, 7(4):312–347, November 2000.

[9] F. Pottier. Simplifying subtyping constraints: a theory. *Information & Computation*, 170(2):153–183, November 2001.

[10] Vaughan Pratt and Jerzy Tiuryn. Satisfiability of inequalities in a poset. *Fundamenta Informaticae*, 28(1-2):165–182, 1996.

[11] G. Smolka. *Logic Programming over Polymorphically Order-Sorted Types*. PhD thesis, Universitat Kaiserslautern, 1989.

[12] V. Trifonov and S. Smith. Subtyping constrained types. In *Proc. 3rd Int'l Symposium on Static Analysis*, number 1145 in LNCS, pages 349–365. Springer, 1996.

A Proofs of section 2

Lemma 4 *Let $S \neq \emptyset \subseteq \mathcal{T}$. If $t \in \downarrow S$, then $t(\epsilon) \in \downarrow_{UL\downarrow S} \sqcap \{s(\epsilon) \mid s \in S\}$. Similarly, if $t \in \uparrow(S)$, then $t(\epsilon) \in \uparrow_{UL\uparrow S} \sqcup \{s(\epsilon) \mid s \in S\}$.*

Proof. One can check that $t(\epsilon) \leq_{\mathcal{K}} \sqcap\{s(\epsilon) \mid s \in S\}$ and that $a(t(\epsilon)) \cap a(\sqcap\{s(\epsilon) \mid s \in S\}) \subseteq UL\downarrow S$. □

Corollary 2 *Let $S \neq \emptyset \subseteq \mathcal{T}$. If $\downarrow S \neq \emptyset$ (resp. $\uparrow S \neq \emptyset$) then $\sqcap_\epsilon S$ (resp. $\sqcup_\epsilon S$) is defined.*

Proof. Since $\downarrow S \neq \emptyset$, there exists $t \in \downarrow S$ and, by lemma 4, $t(\epsilon) \in \downarrow_{UL\downarrow S} \sqcap\{s(\epsilon) \mid s \in S\}$. Thus $\downarrow_{UL\downarrow S} \sqcap \{s(\epsilon) \mid s \in S\} \neq \emptyset$ and by definition, it has an upper bound. Thus it has a least upper bound $\sqcap_\epsilon S$. □

Corollary 3 *Let $S \neq \emptyset \subseteq \mathcal{T}$. If $\downarrow S \neq \emptyset$ (resp. $\uparrow S \neq \emptyset$) then for all $n \in \mathbb{N}$, $\sqcap_n S$ (resp. $\sqcup_n S$) is defined.*

Proof. We show it by induction over n: the case $n = 0$ is trivial. For the case $n+1$, let us consider $\sqcap_{n+1} S$ (the case $\sqcup_{n+1} S$ is similar). By corollary 2, $\sqcap_\epsilon S$ is defined. Let $l \in a(\sqcap_\epsilon S)$, by proposition 1 $l \in SL\downarrow_{UL\downarrow S} k \subseteq UL\downarrow S$. Thus S/l has a lower bound (or an upper bound, depending of the sign of l). Thus, by induction, $(\sqcap_{n+1} S)/l$ is defined. □

Lemma 5 *Let $S \neq \emptyset \subseteq \mathcal{T}$. $\forall t \in \mathcal{T}$, if $t \in \downarrow S$ (resp. $t \in \uparrow S$) then $t(\epsilon) \leq_{\mathcal{K}} \sqcap_\epsilon S$ (resp. $\sqcup_\epsilon S \leq_{\mathcal{K}} t(\epsilon)$) .*

Proof. Let $t \in \downarrow S$. By lemma 4, $k \in \downarrow_{UL\downarrow S} \sqcap\{s(\epsilon) \mid s \in S\}$. Thus, by definition of $\sqcap_\epsilon S$, $k \leq_{\mathcal{K}} \sqcap_\epsilon(S)$. The proof is similar for $\sqcup_\epsilon S$. □

Corollary 4 *Let $S \neq \emptyset \subseteq \mathcal{T}$. $\forall t \in \downarrow S$ (resp. $t \in \uparrow S$), $\forall n \in \mathbb{N}$ $t \leq_n \sqcap_n S$ (resp. $\sqcap_n S \leq_n t$).*

Proof. By induction over n. The case $n = 0$ is trivial. Let us consider $t \leq_{n+1} \sqcap_{n+1} S$. Since $t \in \downarrow S$ with $t(\epsilon) = k$ then, by lemma 5, $k \leq_{\mathcal{K}} \sqcap_\epsilon S$. Let $l \in a(t(\epsilon)) \cap a(\sqcap_\epsilon S)$. If $l \in \mathcal{L}^+$, then , since $t \in \downarrow S$, by proposition 3, $t/l \in \downarrow(S/l)$. By induction, we obtain $t/l \leq_n \sqcap_n(S/l) = (\sqcap_{n+1} S)/l$. The case $l \in \mathcal{L}^-$, is similar. Thus $t \leq_{n+1} \sqcap_{n+1} S$. Similarly, if $t \in \uparrow S$, then $t \geq_{n+1} \sqcup_{n+1} S$. □

Lemma 6 *Let $S \neq \emptyset \subseteq \mathcal{T}$. If $\downarrow S \neq \emptyset$ (resp. $\uparrow S \neq \emptyset$) then $\forall s \in S, \sqcap_\epsilon S \leq_{\mathcal{K}} s(\epsilon)$ (resp. $s(\epsilon) \leq_{\mathcal{K}} \sqcup_\epsilon S$).*

Proof. Since $\exists t \in \downarrow S$, $\sqcap_\epsilon S$ is defined. Let $K_S = \{s(\epsilon) \mid s \in S\}$. $\sqcap_\epsilon S = \sqcup(\downarrow_{UL\downarrow S} \sqcap K_S)$. Moreover $\forall s \in S, \sqcap K_S \leq_{\mathcal{K}} s(\epsilon)$. But, by definition, $\sqcap_\epsilon S \leq_{\mathcal{K}} \sqcap K_S$. Thus $\forall s \in S, \sqcap_\epsilon S \leq_{\mathcal{K}} s(\epsilon)$. The proof is similar for $\sqcup_\epsilon S$. □

Corollary 5 *Let $S \neq \emptyset \subseteq \mathcal{T}$. If $\downarrow S \neq \emptyset$ (resp. $\uparrow S \neq \emptyset$) then for all $n \in \mathbb{N}$, $\forall s \in S, \sqcap_n S \leq_n s$ (resp. $\sqcup_n S \geq_n s$).*

Proof. By induction over n. The case $n = 0$ is trivial. Let $s \in S$. Let us consider $\sqcap_{n+1} S \leq_{n+1} s$. We have $(\sqcap_{n+1} S)(\epsilon) = \sqcap_\epsilon S$, and, by lemma 6, $\sqcap_\epsilon S \leq_\mathcal{K} s(\epsilon)$. Let $l \in a(\sqcap_\epsilon S) \cap a(s(\epsilon))$. By proposition 1, $l \in SL\downarrow_{UL\downarrow S} k \subseteq UL\downarrow(S)$. If $l \in \mathcal{L}^+$, $(\sqcap_{n+1} S)/l = \sqcap_n (S/l)$. We have $s/l \in S/l$. Thus, by induction, $(\sqcap_{n+1} S)/l = \sqcap_n (S/l) \leq_n s/l$. The case $l \in \mathcal{L}^-$ is symmetrical. Similarly, $\sqcup_{n+1} S \geq_{n+1} s$. □

The following lemma tells that if $m \leq n$ then $\sqcap_m S$ is an approximation of $\sqcap_n S$ until depth m.

Lemma 7 *Let $S \neq \emptyset \subseteq \mathcal{T}$ such that $\downarrow S \neq \emptyset$ (resp. $\uparrow S \neq \emptyset$). Let $m, n \in \mathbb{N}$, $m \leq n$, $\forall w \in \mathcal{L}^*$, if $|w| < m$ then $w \in dom(\sqcap_m S) \Leftrightarrow w \in dom(\sqcap_n S)$ and $(\sqcap_m S)(w) = (\sqcap_n S)(w)$ (resp. $w \in dom(\sqcup_m S) \Leftrightarrow w \in dom(\sqcup_n S)$ and $(\sqcup_m S)(w) = (\sqcup_n S)(w)$).*

Proof. By induction over m: the case $n = 0$ is trivial because there is no w such that $|w| < 0$. Let us consider the case $m + 1 \leq n + 1$. If $w = \epsilon$, by definition $\epsilon \in dom(\sqcap_{m+1} S)$, $\epsilon \in dom(\sqcap_{n+1} S)$ and $(\sqcap_{m+1} S)(\epsilon) = \sqcap_\epsilon S = (\sqcap_{n+1} S)(\epsilon)$. If $w = l.w'$, since $(\sqcap_{m+1} S)(\epsilon) = \sqcap_\epsilon S = (\sqcap_{n+1} S)(\epsilon)$ with $a(\sqcap_\epsilon S) = L$, we have $l \in L$. If $l \in \mathcal{L}^+$, then $(\sqcap_{m+1} S)/l = \sqcap_m (S/l)$ and $(\sqcap_{n+1} S)/l = \sqcap_n (S/l)$. By induction, $w' \in dom(\sqcap_m (S/l)) \Leftrightarrow w' \in dom(\sqcap_n (S/l))$. Since $l \in L$, we thus have $w = l.w' \in dom(\sqcap_{m+1} S)) \Leftrightarrow w \in dom(\sqcap_{n+1} S))$. By induction, we have $(\sqcap_m (S/l))(w') = (\sqcap_n (S/l))(w')$, thus $(\sqcap_{m+1} S)(l.w') = ((\sqcap_{m+1} S)/l)(w') = (\sqcap_m (S/l))(w') = (\sqcap_n (S/l))(w') = ((\sqcap_{n+1} S)/l)(w') = (\sqcap_{n+1} S)(l.w')$. The case $l \in \mathcal{L}^-$ is symmetrical. The proof is similar for $\sqcup_{m+1}$ □

Proof of proposition 4. Since $\downarrow S \neq \emptyset$, by corollary 3, for all $n \in \mathbb{N}$, $\sqcap_n S$ is defined. Now we show that $\sqcap_\mathcal{T} S$ is a type:

- Let $w.l \in dom(\sqcap_\mathcal{T} S)$. We have $w.l \in dom(\sqcap_{|w.l|+1} S)$. Thus $w \in dom(\sqcap_{|w.l|+1} S)$. By lemma 7 we obtain $w \in dom(\sqcap_{|w|+1} S)$, thus $w \in dom(\sqcap_\mathcal{T} S)$.
- $\epsilon \in dom(\sqcap_1 S)$, thus $\epsilon \in dom(\sqcap_\mathcal{T} S)$
- Let $w \in dom(\sqcap_\mathcal{T} S)$ such that $(\sqcap_\mathcal{T} S)(w) = k = (\sqcap_{|w|+1} S)(w)$. We have $w.l \in dom(\sqcap_\mathcal{T} S) \Leftrightarrow w.l \in dom(\sqcap_{|w.l|+1} S)$. By lemma 7, $(\sqcap_{|w.l|+1} S)(w) = (\sqcap_{|w|+1} S)(w) = k$. Thus $w.l \in dom(\sqcap_{|w.l|+1} S) \Leftrightarrow l \in a(k)$.

□

Proposition 11 *Let $S \subseteq \mathcal{T}$ with $S \neq \emptyset$ such that $\downarrow S \neq \emptyset$. $\forall l \in a((\sqcap_\mathcal{T} S)(\epsilon))$:*

- *Either $l \in \mathcal{L}^+$ and $(\sqcap_\mathcal{T} S)/l = \sqcap_\mathcal{T} (S/l)$*
- *Or $l \in \mathcal{L}^-$ and $(\sqcap_\mathcal{T} S)/l = \sqcup_\mathcal{T} (S/l)$*

Similarly, if $\uparrow S \neq \emptyset$ then $\forall l \in a((\sqcup_\mathcal{T} S)(\epsilon))$:

- *Either* $l \in \mathcal{L}^+$ *and* $(\sqcap_{\mathcal{T}} S)/l = \sqcup_{\mathcal{T}}(S/l)$
- *Or* $l \in \mathcal{L}^-$ *and* $(\sqcap_{\mathcal{T}} S)/l = \sqcap_{\mathcal{T}}(S/l)$

Proof . $((\sqcap_{\mathcal{T}} S)/l)(w) = (\sqcap_{\mathcal{T}} S)(l.w) = (\sqcap_{|l.w|+1} S)(l.w)$. If $l \in \mathcal{L}^+$ then $(\sqcap_{|l.w|+1} S)/l = \sqcap_{|w|+1}(S/l)$.
In this case $(\sqcap_{|l.w|+1} S)(l.w) = ((\sqcap_{|l.w|+1} S)/l)(w) = (\sqcap_{|w|+1}(S/l))(w) = (\sqcap_{\mathcal{T}}(S/l))(w)$. The case $l \in \mathcal{L}^-$ is symmetrical. We show in the same way the proposition for $\sqcup_{\mathcal{T}} S$. □

Lemma 8 *Let* $S \neq \emptyset \subseteq \mathcal{T}$ *such that* $\downarrow S \neq \emptyset$ *(resp.* $\uparrow S \neq \emptyset$*).* $\forall n \in \mathbb{N}$, $\sqcap_n S \leq_n \sqcap_{\mathcal{T}} S$ *and* $\sqcap_n(S) \geq_n \sqcap_{\mathcal{T}} S$ *(resp.* $\sqcup_n S \leq_n \sqcup_{\mathcal{T}} S$ *and* $\sqcup_n S \geq_n \sqcup_{\mathcal{T}} S$*).*

Proof . By induction over n: the case $n = 0$ is trivial. Let us consider the case $\sqcap_{\mathcal{T}} S \leq_{n+1} \sqcap_{n+1} S$. By lemma 7, $(\sqcap_{\mathcal{T}} S)(\epsilon) = (\sqcap_1 S)(\epsilon) = (\sqcap_{n+1} S)(\epsilon)$. We have $(\sqcap_{\mathcal{T}} S)(\epsilon) \leq_{\mathcal{K}} (\sqcap_{n+1} S)(\epsilon)$ and $(\sqcap_{\mathcal{T}} S)(\epsilon) \geq_{\mathcal{K}} (\sqcap_{n+1} S)(\epsilon)$. Let $l \in a((\sqcap_{\mathcal{T}} S)(\epsilon))$. If $l \in \mathcal{L}^+$, then, by proposition 11, $(\sqcap_{\mathcal{T}} S)/l = \sqcap_{\mathcal{T}}(S/l)$. We also have $(\sqcap_{n+1} S)/l = \sqcap_n(S/l)$. By induction $\sqcap_{\mathcal{T}}(S/l) \leq_n \sqcap_n(S/l)$ and $\sqcap_{\mathcal{T}}(S/l) \geq_n \sqcap_n(S/l)$. Thus $(\sqcap_{n+1} S)/l \leq_n (\sqcap_{\mathcal{T}} S)/l$ and $(\sqcap_{n+1} S)/l \geq_n (\sqcap_{\mathcal{T}} S)/l$. The case $l \in \mathcal{L}^-$ is symmetrical. Thus $\sqcap_{\mathcal{T}} S \leq_{n+1} \sqcap_{n+1} S$ et $\sqcap_{\mathcal{T}} S \geq_{n+1} \sqcap_{n+1} S$. Similarly $\sqcup_{\mathcal{T}} S \leq_{n+1} \sqcup_{n+1} S$ and $\sqcup_{\mathcal{T}} S \geq_{n+1} \sqcup_{n+1} S$. □

Proof of proposition 5. Let $s \in S$. Let $n \in \mathbb{N}$. By corollary 5, $\sqcap_n S \leq_n s$. By lemma 8, $\sqcap_{\mathcal{T}} S \leq_n \sqcap_n S$, thus $\sqcap_{\mathcal{T}} S \leq_n s$. Thus $\sqcap_{\mathcal{T}} S \leq s$. Similarly $s \leq \sqcup_{\mathcal{T}} S$. □

Proof of proposition6. Let $t \in \downarrow S$. Let $n \in \mathbb{N}$. By corollary 4, $t \leq_n \sqcap_n S$ and by lemma 8, $\sqcap_n S \leq_n \sqcap_{\mathcal{T}} S$, thus $t \leq_n \sqcap_{\mathcal{T}} S$. Thus $t \leq \sqcap_{\mathcal{T}} S$. Similarly, if $t \in \uparrow S$, then $\sqcup_{\mathcal{T}} S \leq t$. □

Proof of proposition 7. Let S be the set of subterms of t_1 and t_2. Let $S_\sqcap = \{u \sqcap_{\mathcal{T}} v \mid u \in S \wedge v \in S\}$ and $S_\sqcup = \{u \sqcup_{\mathcal{T}} v \mid u \in S \wedge v \in S\}$.

We show that for all $t \in S_\sqcap \cup S_\sqcup$, for all $w \in dom(t)$, $t/w \in S_\sqcap \cup S_\sqcup$, by induction over w: if $w = \epsilon$ then $t/\epsilon = t \in S_\sqcap \cup S_\sqcup$. If $w = l.w'$: Let us assume that $t = u \sqcap_{\mathcal{T}} v$, with $u \in S$ and $v \in S$ (the proof is similar for $t = u \sqcup_{\mathcal{T}} v$). Let us also assume that $l \in \mathcal{L}^+$ (the proof is similar for $\in \mathcal{L}^-$). By proposition 11, $t/l = (\sqcap_{\mathcal{T}}\{u, v\})/l = \sqcap_{\mathcal{T}}(\{u, v\}/l)$.

- If u/l is defined but not v/l then $t/l = u/l \in S \subseteq S_\sqcap \cup S_\sqcup$
- If v/l is defined but not u/l then $t/l = v/l \in S \subseteq S_\sqcap \cup S_\sqcup$
- If u/l and v/l are both defined, then $u/l \in S$ and $v/l \in S$ and $t/l = u/l \sqcap_{\mathcal{T}} v/l \in S_\sqcap \cup S_\sqcup$.

Thus $t/l \in S_\sqcap \cup S_\sqcup$. By induction, $(t/l)/w' \in S_\sqcap \cup S_\sqcup$, thus $t/w \in S_\sqcap \cup S_\sqcup$.

Since t_1 and t_2 are regular types, S is finite and thus $S_\sqcap \cup S_\sqcup$ is finite. However $t_1 \sqcap_{\mathcal{T}} t_2 \in S_\sqcap$, all its subterms are in $S_\sqcap \cup S_\sqcup$, thus there is only a finite number of such terms, thus $t_1 \sqcap_{\mathcal{T}} t_2$ is a regular type. Similarly $t_1 \sqcup_{\mathcal{T}} t_2$ a regular type. □

B Proofs of section 3

Proof of lemma 2. Since all terms occuring in C are flat terms, $(\Downarrow A)/l \subseteq V(C)$ and $(\Uparrow B)/l \subseteq V(C)$. Moreover, since C is closed and $\forall \alpha \in A, \forall \beta \in B, \alpha \leq \beta \in C$, for all $t \in \Downarrow A, t' \in \Uparrow B$, $dec(t \leq t')$ is defined and included in C. Let $l \in a(sol(A,B))$ and $l \in \mathcal{L}^+$. Let $\alpha \in (\Downarrow A)/l$ and $\beta \neq \alpha \in (\Uparrow B)/l$. There exists $t \in \Downarrow A$ such that $t/l = \alpha$ and $t' \in \Uparrow B$ such that $t'/l = \beta$. Since $dec(t \leq t') \subseteq C$ and $l \in a(t(\epsilon)) \cap a(t'(\epsilon))$, $\alpha \leq \beta \in C$. Similarly, if $l \in \mathcal{L}^-$, $\forall \beta \in (\Uparrow B)/l, \forall \alpha \in (\Downarrow A)/l, \beta \leq \alpha \in C$. □

Lemma 9 *Let $k_1, k_2, k_3, k_4 \in \mathcal{K}$ such that $k_1 \leq_{\mathcal{K}} k_2$,, $k_1 \leq_{\mathcal{K}} k_3$, $k_2 \leq_{\mathcal{K}} k_4$ and $k_3 \leq_{\mathcal{K}} k_4$. Then $\sqcup(\downarrow_{a(k_1)} k_2) \leq_{\mathcal{K}} \sqcup(\downarrow_{a(k_3)} k_4)$.*

Proof . Let $k = \sqcup(\downarrow_{a(k_3)} k_4)$ and $k' = \sqcup(\downarrow_{a(k_1)} k_2)$, $L_i = a(k_i)$, $L = a(k)$ and $L' = a(k')$. By proposition 1, we have $L' = SL\downarrow_{L_1} k_2 \subseteq L_2 \cap L_1$. Since $k_1 \leq_{\mathcal{K}} k_3 \leq_{\mathcal{K}} k_4$, $L_1 \cap L_4 \subseteq L_3$. Thus $L' \cap L_4 \subseteq L_2 \cap L_1 \cap L_4 \subseteq L_2 \cap L_3 \cap L_4 \subseteq L_3 \cap L_4$. Moreover $k' \leq_{\mathcal{K}} k_2 \leq_{\mathcal{K}} k_4$, thus $k' \in \downarrow_{L_3} k_4$, thus $k' \leq_{\mathcal{K}} k$. □

Proof of lemma 3. Let $U_B = \{t(\epsilon) \mid t \in \Uparrow B\}$, $D_A = \{t(\epsilon) \mid t \in \Downarrow A\}$, $U_F = \{t(\epsilon) \mid t \in \Uparrow F\}$, $D_E = \{t(\epsilon) \mid t \in \Downarrow E\}$. We have $D_A \subseteq D_E$ and $U_F \subseteq U_B$. Thus $\sqcup D_A \leq_{\mathcal{K}} \sqcup D_E$ and $\sqcap U_B \leq_{\mathcal{K}} \sqcap U_F$. Moreover $\sqcup D_A \leq_{\mathcal{K}} \sqcap U_B$ and $\sqcup D_E \leq_{\mathcal{K}} \sqcap U_F$. Thus, by lemma 9,
$sol(A,B) = \sqcup(\downarrow_{a(\sqcup D_A)} \sqcap U_B) \leq_{\mathcal{K}} \sqcup(\downarrow_{a(\sqcup D_E)} \sqcap U_F) = sol(E,F)$. □

C Proofs of section 4

Proof of proposition 10 (more detailed). Properties 1), 2), 3), 6) and 8) are not modified by the application of the rules of table 3. Let us consider the application of the rule (Down $\perp$): $C' \rightarrow^* D_1 = D, \alpha \leq t \rightarrow D_2 = D, \alpha \leq t'$. By proposition 8, $D, \alpha \leq t'$ is satisfiable. Thus there exists ρ such that $t_i = \rho(\Downarrow_{D_1} \alpha) \leq t_\alpha = \rho(\alpha) \leq \rho(t)$ and $t_i \leq t_\alpha \leq \rho(t')$. Thus we have $t_i(\epsilon) \leq_{\mathcal{K}} t_\alpha(\epsilon) \leq_{\mathcal{K}} t'(\epsilon)$,
thus $(\Downarrow_{D_2} \alpha)(\epsilon) \leq_{\mathcal{K}} (\Uparrow_{D_2} \alpha)(\epsilon)$. Since for all $l' \in a(t'(\epsilon)), t'/l' = t/l$, property 7) is preserved (the proof for the other rules is similar). Let us show that C'' verifies property 4). Let us consider the following application of the rule (Down $\perp$): $C' \rightarrow^* D_1 = D, \beta \leq t \rightarrow D_2 = D, \beta \leq t'$. This transition can break property 4). Let us take $\alpha \leq \beta \in D_1$ (other cases are similar) and let us show that is it possible to make a transition (Down $\perp$) to reestablish property 4) w.r.t. α and β. The only way property 4) can be broken by (Down $\perp$) is that $k_\alpha = (\Uparrow_{D_2} \alpha)(\epsilon) \not\leq_{\mathcal{K}} t'(\epsilon)$. Let l be the label use when applying the rule (Down $\perp$). We pose $t'(\epsilon) = k'_\beta$ and $t(\epsilon) = k_\beta$. If $l \notin a(k_\alpha)$, then $k_\alpha \in \downarrow_{(a(k_\beta) \setminus \{l\})} k_\beta$ and $k_\alpha \leq_{\mathcal{K}} k'_\beta$. Otherwise,we have $t/l \neq \beta$, because $t \neq \perp$. Thus property 4) is verified for $(\Uparrow_{D_2} \alpha)/l$ and $(\Uparrow_{D_1} \beta)/l$, thus $\Uparrow_{D_2}((\Uparrow_{D_2} \alpha)/l) = \perp$ and we can apply rule (Down $\perp$) upon D_2 and α, thus obtaining D_3. Thus we have $l \notin a((\Uparrow_{D_3} \alpha)(\epsilon))$, and thus $(\Uparrow_{D_3} \alpha)(\epsilon) \leq_{\mathcal{K}} k'_\beta$. Thus we can always apply rule (Down $\perp$) to reestablish property 4). Similarly, this can be done for the other rules and for property 5). Since by proposition 9, the rewriting system is convergent et

terminates, C'' verifies the properties 4) et 5). $\square$

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399