Non-Structural Subtype Entailment in Automata Theory

Joachim Niehren, Tim Priesnitz

Non-Structural Subtype Entailment in Automata Theory

Joachim Niehren and Tim Priesnitz

Programming Systems Lab, Universität des Saarlandes, Saarbrücken, Germany
E-mail: niehren@ps.uni-sb.de, tim@ps.uni-sb.de

Abstract

Decidability of non-structural subtype entailment is a long-standing open problem in programming language theory. In this paper, we apply automata theoretic methods to characterize the problem equivalently by using regular expressions and word equations. This characterization induces new results on non-structural subtype entailment, constitutes a promising starting point for further investigations on decidability, and explains for the first time why the problem is so difficult. The difficulty is caused by implicit word equations that we make explicit.

Key words: programming languages, subtyping, finite automata, word equations.

1 Introduction

Subtyping is a common concept of many programming languages (including C++ and Java). A subtype relation $\tau' \leq \tau$ means that all functions in a program that expect an argument of type τ are sufficiently polymorphic so that they can also be applied to values of the subtype τ'. Thus, one can safely replace values of a type τ by values of subtype τ'.

Subtype constraints are systems of inequations $t \leq t'$ that talk about the subtype relation. Terms t and t' in subtype constraints are built from type variables and type constructors. Two logical operations on subtype constraints were investigated: satisfiability and entailment [1–6]. *Subtype satisfiability* can be checked in cubic time for many type languages [7,8]. A quadratic time algorithm for the variable free case is presented in [9].

Interest in *subtype entailment* was first raised by practical questions on type inference engines with subtyping [10–12]. The efficiency of such systems relies on the existence of powerful simplification algorithms for typings. Such operations can be formulated on the basis of algorithms for subtype entailment.

It then turned out that subtype entailment is a quite complex problem, even for unexpressive type languages where types are ordinary trees. Rehof and Henglein [13] clarified the situation for structural subtyping. This is a tree ordering that relates trees of the same shape only. It is induced by lifting an ordering on constants. If trees are built over the signature $\{\mathit{int}, \mathit{real}, \times, \rightarrow\}$, for instance, then structural subtyping is induced by the usual axiom $\mathit{int} \leq \mathit{real}$ which says that every integer is a real number. Rehof and Henglein showed that structural subtype entailment is coNP-complete [14] for finite trees (simple types) and PSPACE-complete [15] for possibly infinite trees (recursive types).

Subtyping becomes *non-structural* if the constants $\bot$ and $\top$ are admitted that stand for the least and greatest type. Now, trees of different shapes can be related since all trees τ satisfy $\bot \leq \tau \leq \top$. Several cases are to be distinguished: one can consider only *finite* trees or admit *infinite* trees, or one may assume that all function symbols are *co-variant* (such as $\times$) or that some are *contra-variant* (as the function type constructor $\rightarrow$ in its first argument). One can also vary the number of type constructors of each *arity*.

Decidability of non-structural subtype entailment (NSSE) is a prominent open problem in programming language theory. Only a PSPACE lower bound is known which holds in both cases, for finite trees and for infinite trees [15]. The signature $\{\bot, f, \top\}$ is enough to prove PSPACE hardness if f is a type constructor of arity at least 2. But this result does not explain why finding a decision procedure for NSSE is so difficult. On the other hand, only a fragment of NSSE could be proved decidable [16] (and PSPACE-complete).

The idea behind the approach of this paper is to first reformulate Rehof and Henglein's approach for structural subtype entailment in automata theory and second to lift it to the non-structural case. We have carried out both steps successfully but report only on the second step.

A similar automata theoretic approach is already known for satisfiability but not for entailment [7,8]. Our extension to entailment yields a new characterization of NSSE that uses regular expressions and word equations [17,18]. Word equations raise the real difficulty behind NSSE since they spoil the usual pumping arguments from automata theory. They also clarify why NSSE differs so significantly from seemingly similar entailment problems [19,20].

A tree automata based approach to non-structural subtype entailment was proposed recently [21]. It is completely unrelated to the approach of the present paper, where we only deal with word automata. Tree automata are used in the

alternative proposal to recognize the set of all solutions of a subtype constraint. Every solution is seen as tuples of trees [22] that is recognized by tuple tree automata with equality constraints. But unfortunately, the emptiness problem of tuple tree automata with equality constraints is undecidable [23].

The present paper is an extension on a paper presented at TACS'01 [24]. All proofs omitted in this earlier version are given. In particular this subsumes the quite involved completeness proof for our automata construction (Sec. 9).

Plan of the Article.

We recall the definition of NSSE in Section 2, state our characterization of NSSE in Theorem 1 of Section 3 and discuss its consequences.

The remainder of the paper is devoted to the proof of Theorem 1. First, we express NSSE by a so called *safety* property for sets of words (Section 4). Second, we introduce *cap-automata* – a restricted version of P-automata as introduced [16] – which can recognize exactly the same languages as cap set expressions (Section 5). Third we show how to construct cap automata corresponding to entailment judgments. This construction encodes NSSE into universality of cap automata, under the assumption that a satisfiability test for subtype constraints exists (Section 6). Fourth, we present an algorithm that decides satisfiability of non-structural subtype constraints (Section 7). Fifth, we prove the soundness (Section 8) and completeness (Section 9) of our construction. Sixth, we infer restrictions that are satisfied by all constructed cap automata and define corresponding restrictions for cap set expressions (Section 10 and 11). Seventh, we give a back translation (Section 12) that reduces universality of restricted cap automata into NSSE. Finally, we present transformations on restricted cap automata that prove Corollary 1 of Theorem 1 (Section 13) and conclude.

2 Non-Structural Subtype Entailment

In this paper we investigate non-structural subtype constraints over signatures of function symbols $\Sigma = \{\bot, f, \top\}$ with a single non-constant function symbol f that is co-variant. We write ar_g for the *arity* of a function symbol $g \in \Sigma$, i.e $ar_\bot = ar_\top = 0$ and $ar_f \geq 1$.

The choice of such signatures imposes two restrictions: first, we do not allow for contravariant type constructors. These could be covered in our framework even though this is not fully obvious. Second, we do not treat larger signatures

with more than one non-constant function symbol. This is a true restriction that cannot be circumvented easily.

2.1 Words, Trees, and Types

We start with finite, regular, and infinite trees over Σ. Finite trees model simple types while regular or infinite trees model recursive types. We use a standard definition of trees, whose idea is to identify every node of a tree with the word that addresses it relative to the root.

A *word* over an alphabet A is a finite sequence of letters in A. We denote words by π, μ, or ν and the set of words over A with A^*. The *empty word* is written as ε and the free-monoid *concatenation* of words π and μ by juxtaposition $\pi\mu$, with the property that $\varepsilon\pi = \pi\varepsilon = \pi$. A *prefix* of a word π is a word μ for which there exists a word ν such that $\pi = \mu\nu$. If μ is a prefix of π then we write $\mu \leq \pi$ and if μ is a proper prefix of π then we write $\mu < \pi$.

We consider *trees* over Σ as partial functions $\tau : \mathbb{N}^* \rightsquigarrow \Sigma$ which map words over natural numbers to function symbols. The words in $D_\tau \subseteq \mathbb{N}^*$ are called the *nodes* or *paths* of the tree. We require that every tree has a root $\varepsilon \in D_\tau$ and that tree domains D_τ are always prefix closed and arity-consistent. The latter means for all trees τ, nodes $\pi \in D_\tau$, and naturals $i \in \mathbb{N}$ that $\pi i \in D_\tau$ if and only if $1 \leq i \leq ar_{\tau(\pi)}$. A tree τ is *finite* if its domain D_τ is finite and otherwise *infinite*.

We call τ' the subtree of τ at path π if $\tau(\pi\pi') = \tau'(\pi')$ holds for all $\pi \in D_{\tau'}$. We write $\tau.\pi$ for the subtree of τ at node π under the presupposition $\pi \in D_\tau$. A tree is *regular* if it has at most finitely many distinct subtrees.

We will freely interpret function symbols in Σ as tree constructors. To make clear distinctions, we will write $=_\Sigma$ for equality of symbols in Σ and $=$ for equality of trees over Σ. Given $g \in \Sigma$ and trees $\tau_1, \ldots, \tau_{ar_g}$ we define $\tau = g(\tau_1, \ldots, \tau_{ar_g})$ by $\tau(\varepsilon) =_\Sigma g$ and $\tau(i\pi) =_\Sigma \tau_i(\pi)$ for all $\pi \in D_{\tau_i}$ and $1 \leq i \leq ar_g$. We thus consider ground terms over Σ as (finite) trees, for instance $f(\bot, \top)$ or $\bot$. Thereby, we have overloaded our notation since a constant $a \in \Sigma$ can also be seen as the tree τ with $\tau(\varepsilon) = a$. But this should never lead to confusion.

2.2 Non-Structural Subtyping

Let $<_\Sigma$ be the irreflexive partial order on Σ that satisfies $\bot <_\Sigma f <_\Sigma \top$ and $\leq_\Sigma$ its reflexive counterpart. We define *non-structural subtyping* to be the

unique relation $\leq$ on trees which satisfies for all trees τ_1, τ_2 over Σ:

$$\tau_1 \leq \tau_2 \quad \text{iff} \quad \tau_1(\pi) \leq_\Sigma \tau_2(\pi) \text{ for all } \pi \in D_{\tau_1} \cap D_{\tau_2}$$

Again $\leq$ is a reflexive, transitive, and anti-symmetric relation and thus, a partial order.

2.3 Constraint Language

We assume an infinite set of tree valued variables that we denote by x, y, z, u, v, w. A *subtype constraint* φ is a conjunction of literals with the following abstract syntax:

$$\varphi ::= x \leq f(y_1, \ldots, y_n) \mid f(y_1, \ldots, y_n) \leq x \mid x = \bot \mid x = \top \mid \varphi \wedge \varphi'$$

where $n = ar_f$. We interpret constraints φ in the structure of trees over Σ with non-structural subtyping. We distinguish two cases, the structure of finite trees or else of possibly infinite trees. We interpret function symbols in both cases as tree constructors and the predicate symbol $\leq$ by the non-structural subtype relation. Again, this overloads notation: we use the same symbol $\leq$ for the subtype relation on trees and the predicate symbol denoting the subtype relation in constraints. Again, this should not raise confusion.

Note that we do not allow formulas $x \leq y$ in our constraint language. This choice will help us to simplify our presentation essentially. It is, however, irrelevant from the point of view of expressiveness. We can still express $x \leq y$ by using existential quantifiers:

$$x \leq y \leftrightarrow \exists z \exists u \; (f(x, u, \ldots, u) \leq z \wedge z \leq f(y, u, \ldots, u))$$

As in this equivalence, we will sometimes use first-order formulas Φ built from constraints and the usual first-order connectives. We will write V_Φ for the set of free variables occurring in Φ. A solution of Φ is a variable assignment α into the set of finite (resp. possibly infinite) trees which satisfies the required subtype relations; we write $\alpha \models \Phi$ if α solves Φ and say that Φ is *satisfiable*.

Example 1 *The constraint $x \leq f(x)$ is satisfiable, even when interpreted over finite trees. We can solve it by mapping x to $\bot$. In contrast, the equality constraint $x \leq f(x) \wedge f(x) \leq x$ is unsatisfiable over finite trees. It can however be solved by mapping x to the infinite tree $f(f(f(\ldots)))$.*

A formula Φ_1 *entails* Φ_2 (we write $\Phi_1 \models \Phi_2$) if all solutions $\alpha \models \Phi_1$ satisfy $\alpha \models \Phi_2$. We will consider entailment judgments that are triples of the form (φ, x, y) that we write as $\varphi \models^? x{\leq}y$. *Non-structural subtype entailment* (NSSE) for Σ is the problem to check whether entailment $\varphi \models x{\leq}y$ holds for a given entailment judgment $\varphi \models^? x{\leq}y$.

Note that entailment judgments of the simple form $\varphi \models^? x{\leq}y$ can express general entailment judgments, where both sides are conjunctions of inequations $t_1{\leq}t_2$ between nested terms or variables (i.e. $t ::= x \mid f(t_1, \ldots, t_n) \mid \bot \mid \top$). The main trick is to replace a judgment $\varphi \models^? t_1{\leq}t_2$ with terms t_1 and t_2 by $\varphi \wedge x{=}t_1 \wedge y{=}t_2 \models^? x{\leq}y$ where x and y are fresh variables. Note also that the omission of formulas $u{\leq}v$ on the left hand side does not restrict the problem. (Existential quantifier on the left hand side of an entailment judgment can be removed.)

Example 2 *The prototypical example where NSSE holds somehow surprisingly is:*

$$x{\leq}f(y) \wedge f(x){\leq}y \models^? x{\leq}y \qquad \textit{(yes)}$$

To see this, note that all finite trees in the unary case are of the form $f \ldots f(\bot)$ or $f \ldots f(\top)$. Thus, $x{\leq}y \vee y{<}x$ is valid in this case. Next let us contradict the assumption that there is a solution $\alpha \models y{<}x \wedge x{\leq}f(y) \wedge f(x){\leq}y$. Transitivity yields $\alpha(y){\leq}f(\alpha(y))$ and then also $f(\alpha(x)){\leq}f(\alpha(y))$. Hence $\alpha(x){\leq}\alpha(y)$ which contradicts $\alpha(y){<}\alpha(x)$.

3 Characterization

We now formulate the main result of this paper and discuss its relevance (Theorem 1). This is a new characterization of NSSE which is based on a new class of extended regular expressions: *cap set expressions* that we introduce first.

We start with *regular expressions* R over some alphabet A that are defined as usual:

$$R := a \mid \varepsilon \mid R_1R_2 \mid R^* \mid R_1{\cup}R_2 \mid \emptyset \qquad \text{where } a \in A$$

Every regular expression R describes a regular language of words $\mathcal{L}(R) \subseteq A^*$. We next introduce *cap set expressions* E over A. (Their name will be explained in Section 5.)

$$E ::= R_1R_2^\circ \mid \; E_1 \cup E_2$$

Cap set expressions E denote sets of words $\mathcal{L}(E) \subseteq A^*$ that we call *cap sets*. We have to define the *cap set operator* $^\circ$ on sets of words, i.e., we must define the set $S^\circ \subseteq A^*$ for all sets $S \subseteq A^*$. Let pr be the prefix operator lifted to sets of words. We set:

$$S^\circ \;=\; \{\pi \mid \pi \in pr(\mu^*),\ \mu \in S\}$$

A word π belongs to S° if π is a prefix of a power $\mu \dots \mu$ of some word $\mu \in S$. Note that cap set expressions subsume regular expressions: indeed, $\mathcal{L}(R) = \mathcal{L}(R\,\varepsilon^\circ)$ for all R. But the cap operator adds new expressiveness when applied to an infinite set: there exist regular expression R such that the language of the cap set expression R° is neither regular nor context free. Consider for instance $(21^*)^\circ$ which denotes the set of all prefixes of words $21^n\,21^n \dots 21^n$ where $n \geq 0$. Clearly this set is not context-free.

We will derive appropriate *restrictions* on cap set expressions (Def. 4) such that the following theorem becomes true.

Theorem 1 (Characterization) *The decidability of NSSE for a signature $\{\bot, f, \top\}$ with a single function symbol of arity $n \geq 1$ is equivalent to the decidability of the universality problem for the class of restricted cap set expressions over the alphabet $\{1, \dots, n\}$. This result holds equally for finite, regular, and possibly infinite trees.*

The theorem allows us to derive the following robustness result of NSSE against variations from automata transformations (Section 13).

Corollary 1 *All variants of NSSE with signature $\{\bot, f, \top\}$ where the arity of f is at least $n \geq 2$ are equivalent. It does not even matter whether finite, regular, or infinite trees are considered.*

Theorem 1 can also be used to relate NSSE to word equations. The idea is to express membership in cap sets in the positive existential fragment of word equations with regular constraints [25]. The reduction can easily be based on the following lemma that is well known in the field of string unification.

Lemma 1 *For all words $\pi \in A^*$ and nonempty words $\mu \in A^+$ it holds that $\pi \in pr(\mu^*)$ if and only if $\pi \in pr(\mu\pi)$.*

PROOF. If $\pi \in pr(\mu^*)$ then there is a natural number $n \geq 1$ such that $\mu^{n-1} \leq \pi \leq \mu^n$. Hence, $\pi \leq \mu\mu^{n-1} \leq \mu\pi$ as required. For the converse, let $\pi \leq \mu\pi$ where $\mu \neq \varepsilon$. We prove $\pi \in pr(\mu^*)$ by induction on the length of π. If $|\pi| \leq |\mu|$ then $\pi \leq \mu$ and thus $\pi \in pr(\mu^*)$ as required. Otherwise, there exists a path π' with $\pi = \mu\pi'$ and thus $\pi' \leq \mu\pi'$ by our assumption that $\pi \leq \mu\pi$. Note that π' is a proper prefix of π since $\mu \neq \varepsilon$. The induction hypothesis applied to π' yields $\pi' \in pr(\mu^*)$ such that $\pi \in pr(\mu^*)$.

Lemma 1 states that all sets $S \subseteq A^+$ of nonempty words satisfy:

$$\pi \in S^\circ \leftrightarrow \exists\mu\exists\nu\ (\mu \in S \wedge \pi\nu{=}\mu\pi)$$

Theorem 1 thus implies that we can express the universality problem of cap set expressions E in the positive $\forall\exists^*$ fragment of the first-order theory of word equations with regular constraints.

Corollary 2 *NSSE with a single function symbol of arity $n \geq 1$ can be expressed in the positive $\forall\exists^*$ fragment of the first-order theory of word equations with regular constraints over the alphabet $\{1, \ldots, n\}$.*

Unfortunately, even the positive $\forall\exists^3$ fragment of a single word equation is undecidable [26] except if the alphabet is infinite [27] or a singleton [28]. Therefore, it remains open whether NSSE is decidable or not. But it becomes clear that the difficulty is raised by word equations hidden behind cap set expressions R°, i.e. the equation $\pi\nu{=}\mu\pi$ in Lemma 1.

Theorem 1 constitutes a promising starting point to further investigate decidability of NSSE. For instance, we can infer a new decidability result for the monadic case directly from Corollary 2.

Corollary 3 *NSSE is decidable for the signature $\{\bot, f, \top\}$ if f is unary.*

4 Safety

The goal of this section is to characterize NSSE by properties of sets of words, that we call safety properties. Appropriate safety properties can be verified by P-automata as we will show in Section 6.

We use terms $x(\pi)$ to denote the node label of the value of x at path π. Whenever we use this term, we presuppose the existence of π in the tree domain of the value of x. For instance, the formula $x(12) \leq_\Sigma \top$ is satisfied by a variable assignment if and only if the tree assigned to x contains the node 12.

We next recall the notion of safety from [16]. Let $\varphi \models^? x{\leq}y$ be an entailment judgment and π a word in $\{1, \ldots, ar_f\}^*$. We call π *safe* for $\varphi \models^? x{\leq}y$ if entailment cannot be contradicted at π, i.e. if $\varphi \wedge y(\pi) <_\Sigma x(\pi)$ is unsatisfiable. Clearly entailment $\varphi \models x{\leq}y$ is equivalent to that all paths are safe for $\varphi \models^? x{\leq}y$.

For a restricted class of entailment judgments it is shown in [16] that the above notion of safety can be checked by testing universality of P-automata.

Unfortunately, it is unclear how to lift this result to the general case. To work around, we will refine the notion of safety into two dual notions: *left (l) safety* and *right (r) safety*. These notions will be defined on formulas:

$$\mathrm{pref}^g_\pi(x) \quad =_{\mathrm{def}} \quad \bigvee_{\pi' \leq \pi} x(\pi') =_\Sigma g$$

for some function symbol $g \in \Sigma$, word π, and variable x. It requires x to denote a tree that is labeled by g at some prefix π' of π. We now define l-safety for words $\pi \in \{1, \ldots, ar_f\}^*$ with respect to judgments $\varphi \models^? x{\leq}y$:

$$\pi \text{ is } \textit{l-safe} \text{ for } \varphi \models^? x{\leq}y \;\text{ iff }\; \varphi \models \mathrm{pref}^\top_\pi(x) \to \mathrm{pref}^\top_\pi(y)$$

If π is l-safe for judgement $\varphi \models^? x{\leq}y$ then entailment contradicted by a solution α of φ that maps the left hand side x to some tree where $\alpha(x)(\pi) =_\Sigma \top$, except if $\alpha(x)(\pi') =_\Sigma \top$ and $\alpha(y)(\pi'') =_\Sigma \top$ for some prefixes π' and π'' of π. The notion of r-safety is analogous; here one tries to contradict with $\bot$ at the right hand side y:

$$\pi \text{ is } \textit{r-safe} \text{ for } \varphi \models^? x{\leq}y \;\text{ iff }\; \varphi \models \mathrm{pref}^\bot_\pi(y) \to \mathrm{pref}^\bot_\pi(x)$$

We define a variable assignment α to be l-safe or r-safe for $\alpha \models^? x{\leq}y$ by replacing φ literally with α in the above definitions. Note that our safety notions depend on the chosen structure of trees over which we interpret our formulas.

We first illustrate these concepts by a judgment with a unary function symbol:

$$z{=}\top \wedge f(z){\leq}y \models^? x{\leq}y \qquad \text{(no)}$$

Here, ε is r-safe but not l-safe. All other paths $\pi \in 1^+$ are both l-safe and r-safe. There is a variable assignment α which contradicts entailment: $\alpha(x) = \top$, $\alpha(z) = \top, \alpha(y) = f(\top)$. This shows that ε is indeed not l-safe for $\alpha \models^? x{\leq}y$.

Proposition 1 *Entailment $\varphi \models x{\leq}y$ holds if and only if all words $\pi \in \{1, \ldots, ar_f\}^*$ are l-safe and r-safe for $\varphi \models^? x{\leq}y$.*

PROOF. We first assume that entailment does not hold and show that either l-safety or r-safety can be contradicted for some path. As argued above, there exists an unsafe path π such that $\varphi \wedge y(\pi) <_\Sigma x(\pi)$ is satisfiable. Let α be a solution of this formula.

(1) If $\alpha(y)(\pi) =_\Sigma \bot$ then $\alpha \models \mathrm{pref}^\bot_\pi(y)$. Since $\alpha \models y(\pi) <_\Sigma x(\pi)$ it holds $\alpha(x)(\pi) \in \{f, \top\}$ which implies $\alpha \models \neg\mathrm{pref}^\bot_\pi(x)$. Thus, π is not r-safe.
(2) Otherwise $\alpha(y)(\pi) =_\Sigma f$ which implies $\alpha \models \neg\mathrm{pref}^\top_\pi(y)$. Further, it holds $\alpha(x)(\pi) =_\Sigma \top$ which implies $\alpha \models \mathrm{pref}^\top_\pi(x)$. Thus, π is not l-safe.

For the converse, we assume entailment $\varphi \models x{\leq}y$ and show that all paths are l-safe and r-safe for $\varphi \models^? x{\leq}y$. We fix a path π and solution α of φ, and show that π is l-safe and r-safe for $\alpha \models^? x{\leq}y$. Let π' be the longest prefix of π which belongs to $D_{\alpha(x)} \cap D_{\alpha(y)}$.

(1) If $\alpha(x)(\pi') =_\Sigma \bot$ then $\alpha \models \neg\mathrm{pref}^\top_\pi(x)$ so that π is l-safe for $\alpha \models^? x{\leq}y$, and also $\alpha \models \mathrm{pref}^\bot_\pi(x)$ such that π is r-safe for $\alpha \models^? x{\leq}y$.
(2) Suppose $\alpha(x)(\pi') =_\Sigma \top$. Since $\alpha \models \varphi$ and $\varphi \models x{\leq}y$, we know that $\alpha \models x{\leq}y$. Since π' is a node of both trees it follows that $\alpha(x)(\pi') \leq_\Sigma \alpha(y)(\pi')$ and thus $\alpha(y)(\pi') =_\Sigma \top$. Since $\pi' \leq \pi$, $\alpha \models \mathrm{pref}^\top_\pi(y) \wedge \neg\mathrm{pref}^\bot_\pi(y)$. Thus, π is l-safe and r-safe for $\alpha \models^? x{\leq}y$.
(3) The last possibility is $\alpha(x)(\pi') =_\Sigma f$. We can infer from entailment that $\alpha(y)(\pi') \in \{f, \top\}$. If $\alpha(y)(\pi') =_\Sigma \top$ we are done as before. Otherwise, $\alpha(y)(\pi') =_\Sigma \alpha(x)(\pi') =_\Sigma f$ such that the maximality of π' and $ar_f \geq 1$ yields $\pi = \pi'$. Now, $\alpha \models \neg\mathrm{pref}^\bot_\pi(y)$ so that π is r-safe, and also $\alpha \models \neg\mathrm{pref}^\top_\pi(x)$ such that π is l-safe for $\alpha \models^? x{\leq}y$.

Example 3 *The surprising effect of Example 2 seems to go away if one replaces the unary function symbol there by a binary function symbol:*

$$x{\leq}f(y,y) \wedge f(x,x){\leq}y \models^? x{\leq}y \qquad \textit{(no)}$$

Now, all words in $1^* \cup 2^*$ *are l-safe and r-safe, but* 12 *is neither. Entailment can be contradicted by variable assignments mapping* x *to* $f(f(\bot,\top),\bot)$ *and* y *to* $f(f(\top,\bot),\top)$.

Example 4 *This example is a little more complicated. Its purpose is to show that entailment in the binary case can also be raised by a similar effect as in Example 2. How to understand this effect in general will be explained in Section 6.*

$$x{\leq}f(y,y) \wedge f(z,z){\leq}y \wedge f(u,u){\leq}z \wedge u{=}\top \models^? x{\leq}y \qquad \textit{(yes)}$$

5 Cap Automata and Cap Sets

We need a notion of automata that can recognize cap sets. Therefore, we restrict the class of P-automata introduced in [16] to the class of so called *cap automata* [1]. We then show that the class of languages recognized by cap

[1] Cap automata are the same objects as P-automata, i.e. finite automata with a set of P-edges. The difference between both concepts concerns only the corresponding language definitions. Both definitions coincide for those automata $\mathcal{P}$ that satisfy the following condition (the proof is straightforward): if $\mathcal{P} \vdash q_1 \xrightarrow{\pi} q_2 \xrightarrow{\mu} q_3 \dashrightarrow q_1$ then q_2 is a final state in $\mathcal{P}$. This condition can be assumed w.l.o.g for all cap

automata is precisely the class of cap sets, i.e. those sets of words described by cap set expressions.

A *finite automaton* $\mathcal{A}$ over alphabet A consists of a set Q of *states*, a set $I \subseteq Q$ of *initial* states, a set $F \subseteq Q$ of *final* states, and a set $\Delta \subseteq Q \times (A \cup \{\varepsilon\}) \times Q$ of *transitions*. Note that Δ permits ε transitions and single letter transitions. We will write $\mathcal{A} \vdash q$ if $q \in Q$ is a state of $\mathcal{A}$, $\mathcal{A} \vdash \underline{\underline{q}}$ if $q \in F$ is a final state of $\mathcal{A}$, and $\mathcal{A} \vdash \gtrdot q$ if $q \in I$ is an initial state of $\mathcal{A}$. The statement $\mathcal{A} \vdash q \xrightarrow{\pi} q'$ says that $\mathcal{A}$ started at q permits a sequence of transitions consuming π and ending in q'. Note that $\mathcal{A} \vdash q \xrightarrow{\varepsilon} q$ holds for all states $q \in Q$. We call $\mathcal{A}$ *c*omplete if for every word $\pi \in A^*$ there exists states q_0 and q_1 such that $\mathcal{A} \vdash \gtrdot q_0 \xrightarrow{\pi} q_1$.

Definition 1 *A* cap automaton $\mathcal{P}$ *over alphabet* A *consists of a finite automaton* $\mathcal{A}$ *over* A *and a set of* P-edges $P \subseteq Q \times Q$. *We write* $\mathcal{P} \vdash q \dashrightarrow q'$ *if* $\mathcal{P}$ *has a P-edge* $(q, q') \in P$. *A cap automaton* $\mathcal{P}$ *over* A *recognizes the following language* $\mathcal{L}(\mathcal{P}) \subseteq A^*$*:*

$$\mathcal{L}(\mathcal{P}) = \{\pi \mid \mathcal{P} \vdash \gtrdot q_0 \xrightarrow{\pi} \underline{\underline{q_1}}\} \cup \{\pi\mu' \mid \mu' \in pr(\mu^*), \mathcal{P} \vdash \gtrdot q_0 \xrightarrow{\pi} \underline{\underline{q_1}} \xrightarrow{\mu} q_2 \dashrightarrow \underline{\underline{q_1}}\}$$

The first set is the language of the finite automaton underlying $\mathcal{P}$. The second set adds the contribution of P-edges: if a cap automaton traverses a P-edge $\mathcal{P} \vdash q_2 \dashrightarrow q_1$ then it must have reached q_2 from q_1 of some word μ, i.e. $\mathcal{P} \vdash q_1 \xrightarrow{\mu} q_2 \dashrightarrow q_1$; in the sequel the automaton can loop through μ^* and quit the loop at any time.

Fig. 1 contains a cap automaton over the alphabet $\{1, 2\}$ that recognizes the non-context free cap set from the introduction, i.e. described by the cap set expression $(21^*)^\circ$. We generally draw cap automata as one draws finite automata but with additional dashed arrows to indicate P-edges.

The tree on the right in Fig. 1 represents the language recognized by this cap automaton. The language of a cap automaton $\mathcal{P}$ with alphabet $\{1, \ldots, n\}$ is drawn as a n-ary *c*lass tree. This is a complete infinite n-ary tree whose nodes are labeled by classes A, P, and C. Each node of the class tree is a word in $\{1, \ldots, n\}^*$ that is labeled by the class that $\mathcal{P}$ adjoins to it. We assign the class C to all words in the complement of $\mathcal{L}(\mathcal{P})$ of a cap automaton $\mathcal{P}$. The words with class A are recognized by the finite automaton underlying $\mathcal{P}$. All remaining words belong to class P. These are accepted by $\mathcal{P}$ but not by the underlying finite automaton.

We now explain the name *cap*: it is an abbreviation for the regular expression $(\,\mathrm{C} \cup \mathrm{A}^+\mathrm{P}^*\,)^*$. Branches in class trees of cap automata always satisfy that

automata, since it is satisfied by all those constructed in the proof of Proposition 2. Thus, cap automata are properly subsumed by the P-automata.

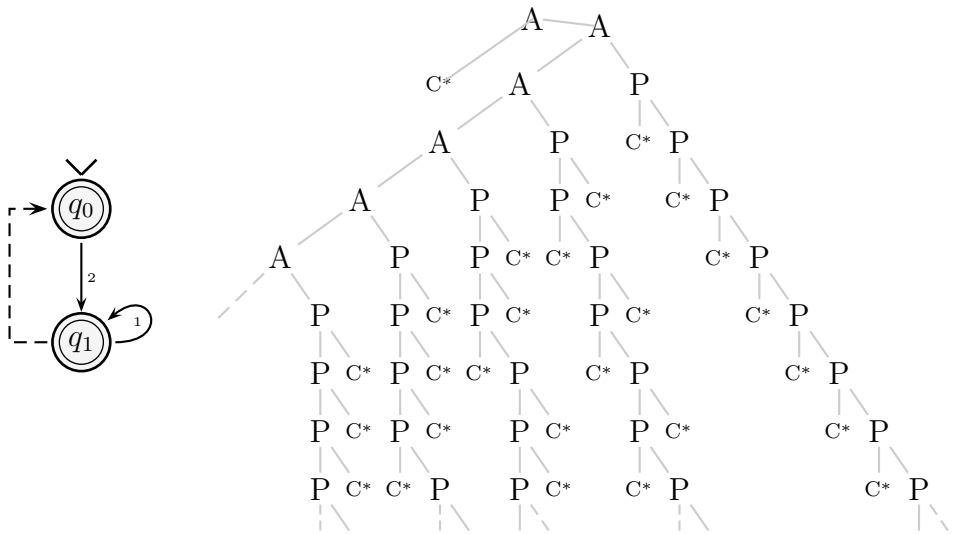


Fig. 1. A cap automaton with a non context-free language $(21^*)^\circ$.

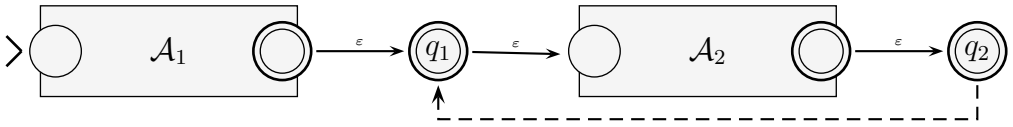


Fig. 2. Construction of a cap automaton for the language $\mathcal{L}(\mathcal{A}_1)\mathcal{L}(A_2)^\circ$.

expression. This means that all nodes of class P in a class tree have a mother node in either of the classes A or P. To see this, note first that root nodes of class trees can never belong to class P. Thus, all P nodes must have a mother. Furthermore, the mother of a P node cannot belong to the C class due to the cap property.

Proposition 2 *Cap set expressions and cap automata recognize precisely the same class of languages. Universality of cap set expressions and cap automata are equivalent modulo deterministic polynomial time transformations.*

PROOF. For the one direction, let R_{q_1,q_2} be a regular expression for the set $\{\pi \mid \mathcal{P} \vdash q_1 \xrightarrow{\pi} q_2\}$ then the language of a cap automaton is equal to the union of $\cup_{\mathcal{P}\vdash} {}_{\succ q_0} \cup_{\mathcal{P}\vdash \underline{\underline{q_1}}} R_{q_0,q_1}$ and $\cup_{\mathcal{P}\vdash} {}_{\succ q_0} \cup_{\mathcal{P}\vdash \underline{\underline{q_1}}} \cup_{\mathcal{P}\vdash q_2} {}_{\dashrightarrow q_1} R_{q_0,q_1}(R_{q_1,q_2})^\circ$. The needed regular expressions can be computed in polynomial time

For the converse, we first note that the class of languages recognized by cap automata is closed under union since cap automata may have several initial states. There thus only remains to build cap automata for expressions $R_1R_2^\circ$. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be finite automata that recognize R_1 respectively R_2. W.l.o.g.

reflexive	$\varphi \vdash x \leq x$	for variables $x \in V_\varphi$
trans.	$\varphi \vdash x \leq z$	if $\varphi \vdash x \leq y$ and $\varphi \vdash y \leq z$
decomp.	$\varphi \vdash x_i \leq y_i$	if $1 \leq i \leq n$, $\varphi \vdash x \leq y$, $f(x_1, \ldots, x_n) \leq x \wedge y \leq f(y_1, \ldots, y_n)$ in φ

Table 1
Syntactic support of inequalities.

we can assume that both automata have a unique initial and a unique final state. Multiple initial or final states of finite automata (but not of cap automata) can be eliminated by introducing new ε-transitions. We now compose $\mathcal{A}_1$ and $\mathcal{A}_2$ into a new cap automaton that recognizes the language of $R_1 R_2^\circ$ as illustrated in Fig. 2: we add two fresh final states q_1 and q_2 and link $\mathcal{A}_1$ and $\mathcal{A}_2$ over these states. This requires 3 new ε-edges and a new P-edge from q_2 to q_1. To account for the prefix closure within the $^\circ$ operator, we finally turn all states of $\mathcal{A}_2$ into additional final states.

6 Automata Construction

This section presents a construction for cap automata that can test l-safety and r-safety for entailment judgments. The same construction applies for the three structures of finite, regular, and possibly infinite trees. The only difference is hidden in a subroutine for testing satisfiability (see Section 7).

6.1 Closure Algorithm

As a prerequisite for our automata construction, we use a closure algorithm. This algorithm computes a set of inequalities of the form $x \leq y$ that are syntactically supported by a constraint φ.

In contrast to other closure algorithms, we cannot simply add supported inequalities to the initial constraint φ given our syntactic restriction. Instead, Table 1 defines judgments $\varphi \vdash x \leq y$ which state that φ supports $x \leq y$ syntactically. The definition consists of three standard rules. The first two rules express the reflexivity and transitivity of the subtype ordering. Finally and most importantly, the definition of syntactic support accounts for decomposition which can be applied recursively.

Lemma 2 *For all φ, x, and y: if $\varphi \vdash x \leq y$ then $\varphi \models x \leq y$.*

PROOF. By induction on the definition of syntactic support of inequalities.

$\varphi \vdash x \leq i(z)$ if $\varphi \vdash x \leq x'$, $x' \leq f(y_1, \ldots, y_n)$ in φ, $\varphi \vdash y_i \leq z$, and $1 \leq i \leq ar_f$
$\varphi \vdash i(z) \leq x$ if $\varphi \vdash z \leq y_i$, $f(y_1, \ldots, y_n) \leq x'$ in φ, $\varphi \vdash x' \leq x$, and $1 \leq i \leq ar_f$

Table 2
Syntactic support of lower and upper bounds for i-th children.

To keep the automata construction simple, Table 2 define two further forms of syntactic support: $\varphi \vdash x \leq i(y)$ means that y is an upper bound of the i-th child of x and $\varphi \vdash i(y) \leq x$ states the symmetric lower bound for x at i.

Lemma 3 *For all φ, x, y, and $1 \leq i \leq n = ar_f$ if $\varphi \vdash x \leq i(z)$ then:*

$$\varphi \models \exists y_1 \ldots \exists y_{i-1} \exists y_{i+1} \ldots \exists y_n.\ x \leq f(y_1, \ldots, y_{i-1}, z, y_{i+1}, \ldots, y_n).$$

The symmetric property for lower bound $\varphi \vdash i(z) \leq x$ is valid too.

PROOF. Obvious from Lemma 2 and Table 2.

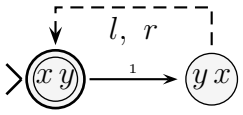
6.2 Left and Right Automata

The automata construction is given in Table 3. For each entailment judgment $\varphi \models^? x \leq y$ we construct a left automaton $\mathcal{P}_l(\varphi \models^? x \leq y)$ and a right automaton $\mathcal{P}_r(\varphi \models^? x \leq y)$. The left automaton is supposed to accept all l-safe paths for $\varphi \models^? x \leq y$, and the right automaton all r-safe paths (up to appropriate assumptions). Entailment then holds if and only if the languages of both cap automata are universal. Note that it remains open whether the set of simultaneously l-safe and r-safe paths can be recognized by a single cap automaton. The problem is that cap automata are not closed under intersection (proof omitted).

The left and right automata always have the same states, transitions, and initial states. When testing for $\varphi \models^? x \leq y$ the only **initial state** is (x, y). A state (u, s) of the left automaton is made **final** if there is an upper bound $u \leq f(u_1, \ldots, u_n)$ in φ, which proves that the actual path is l-safe. The **descend** rule can also be applied in that case. The safety check then continues in some state (u_i, s') and extends the actual path by i. It can chose $s' = \_$ while ignoring the right hand side, or if s is also a variable descend simultaneously on the right hand side. There are three rules that prove that the actual path and **all** its extensions are l-safe: **bot**, **top**, and **reflexivity**. Finally there is a single rule that adds **P-edges** to the left automaton. The rules for the right automaton are symmetric.

alphabet	$A_\Sigma = \{1, \ldots, ar_f\}$	
states	$\mathcal{P}_\theta \vdash (s, s')$	if $s, s' \in V_\varphi \cup \{x, y, \_\}$
	$\mathcal{P}_\theta \vdash \underline{\underline{all}}$	
initial state	$\mathcal{P}_\theta \vdash \ \gtrdot(x, y)$	
final states	$\mathcal{P}_l \vdash \underline{\underline{(u, s)}}$	if $\varphi \vdash u \leq i(u'),\ i \in A_\Sigma$
	$\mathcal{P}_r \vdash \underline{\underline{(s, v)}}$	if $\varphi \vdash i(v') \leq v,\ i \in A_\Sigma$
descend	$\mathcal{P}_\theta \vdash (u, s) \xrightarrow{i} (u', \_)$	if $\varphi \vdash u \leq i(u'),\ i \in A_\Sigma$
	$\mathcal{P}_\theta \vdash (s, v) \xrightarrow{i} (\_, v')$	if $\varphi \vdash i(v') \leq v,\ i \in A_\Sigma$
	$\mathcal{P}_\theta \vdash (u, v) \xrightarrow{i} (u', v')$	if $\varphi \vdash u \leq i(u'),\ \varphi \vdash i(v') \leq v,\ i \in A_\Sigma$
bot	$\mathcal{P}_\theta \vdash \underline{\underline{(u, s)}} \xrightarrow{i} \underline{\underline{all}}$	if $\varphi \vdash u \leq u',\ u' = \bot$ in $\varphi,\ i \in A_\Sigma$
top	$\mathcal{P}_\theta \vdash \underline{\underline{(s, v)}} \xrightarrow{i} \underline{\underline{all}}$	if $\varphi \vdash v' \leq v,\ v' = \top$ in $\varphi,\ i \in A_\Sigma$
reflexivity	$\mathcal{P}_\theta \vdash \underline{\underline{(u, v)}} \xrightarrow{i} \underline{\underline{all}}$	if $\varphi \vdash u \leq v,\ i \in A_\Sigma$
all	$\mathcal{P}_\theta \vdash \underline{\underline{all}} \xrightarrow{i} \underline{\underline{all}}$	if $i \in A_\Sigma$
P−edges	$\mathcal{P}_l \vdash (u, s) \dashrightarrow (v, u)$	
	$\mathcal{P}_r \vdash (s, v) \dashrightarrow (v, u)$	

Table 3
Construction of the cap automata $\mathcal{P}_\theta = \mathcal{P}_\theta(\varphi \models^? x \leq y)$ for both sides $\theta \in \{l, r\}$.

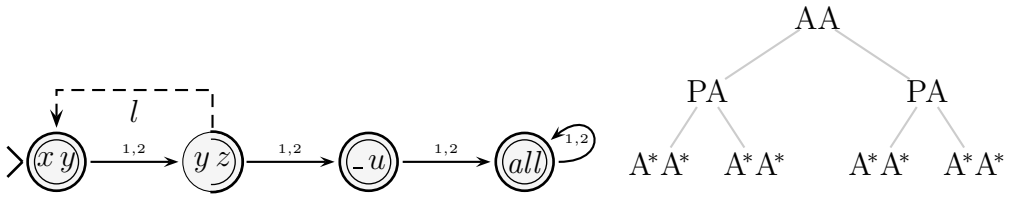


$x \leq f(y) \wedge f(x) \leq y \models^? x \leq y$

AA
PP
PP
P^*P^*

Fig. 3. Automata construction for Example 2. Entailment holds.

When drawing the constructed left and right automata (Fig. 3 and 4), we always share the states and transitions for reasons of economy. Different elements of the two automata carry extra annotations. Final states of the left (right) automaton are put into a left (right) double circle. If a state is final for both automata then it is drawn within a complete double circle. We annotate P-edges of the left automaton by l and of the right automaton with r.



$x{\leq}f(y,y) \wedge f(z,z){\leq}y \wedge f(u,u){\leq}z \wedge u{=}\top \models^{?} x{\leq}y$

Fig. 4. Automata construction for Example 4. Entailment holds.

6.3 Examples

We first illustrate the automata construction for the unary Example 2, recalled in Fig. 3. The alphabet of both automata is the singleton $\{1\}$. The relevant states are $\{(x,y), (y,x)\}$; all others are either unreachable or do not lead to a final state. The constraints $x{\leq}f(y)$ and $f(x){\leq}y$ let both cap automata **descend** simultaneously by the transition $(x,y) \xrightarrow{1} (y,x)$ and turn (x,y) into a **final state** of both automata. There are **P-edges** $(y,x) \dashrightarrow (x,y)$ for both cap automata. Note that we ignore the symmetric P-edges $(x,y) \dashrightarrow (y,x)$ in the picture since they don't contribute to the respective languages.

Fig. 3 also contains the class trees for both cap automata but in an overlaid fashion. The languages of both cap automata are universal due to their P-edges. Given that our construction is sound (see Sec. 8) this proves entailment.

We now consider the more complex binary Example 4 in Fig. 4 where the alphabet is $\{1,2\}$. The constraint $f(u,u){\leq}z$ permits to **descend** from (y,z) while ignoring the variable y on the left hand side; this justifies the transition $(y,z) \xrightarrow{1,2} (\_,u)$. Since the left hand side is ignored, the state (y,z) is only put to the **final states** of the right automaton. The **top** rule can be applied to $u{=}\top$; hence there are transitions $(\_,u) \xrightarrow{1,2} \mathit{all}$ where $(\_,u)$ and all are universal states according to the **all** rule. Finally, there is a **P-edge** $(y,z) \dashrightarrow (x,y)$ for the left cap automaton. We again ignore the symmetric **P-edge** $(x,y) \dashrightarrow (y,z)$ since it does not contribute to the language. The languages of both automata are again universal, in case of the left automaton because of a P-edge.

The next example shows that decomposition closure as provided by the notion of syntactic support is needed for completeness. We consider

$$f(x){\leq}z \wedge z{\leq}f(z) \wedge f(z){\leq}y \models^{?} x{\leq}y \qquad \text{(yes)}$$

Let φ be the left hand side. Since φ contains $f(x){\leq}z \wedge z{\leq}f(z)$, it supports $\varphi \vdash x{\leq}z$ syntactically. Thus, $\varphi \models x{\leq}f(z){\leq}y$ so that entailment holds. This

can also be proved through our automaton construction. First note that $\varphi \vdash x{\leq}1(z)$ holds (since $\varphi \vdash x{\leq}z$ and $z{\leq}f(z)$ in φ). Furthermore, $f(z){\leq}y$ in φ so that $\varphi \vdash 1(z){\leq}y$. Hence, we can **descend** to the first child simultaneously for x and y with the transition:

$$\mathcal{P}_\theta(\varphi \models^? x{\leq}y) \vdash \underline{\underline{(x,y)}} \xrightarrow{1} (z,z)$$

This applies for both sides $\theta \in \{l,r\}$ and proves that ε is l-safe and r-safe for $\varphi \models^? x{\leq}y$. **Reflexivity** shows that all words in 1^+ are l-safe and r-safe too. Thus, our automata construction proves entailment to hold as well.

6.4 Result

The automata construction is sound and complete for both cases, the structures of finite resp. possibly infinite trees.

Proposition 3 (Soundness and Completeness) *Let $\theta \in \{l,r\}$ a side, φ be a constraint and $x,y \in V_\varphi$ variables. If φ is satisfiable then $\mathcal{P}_\theta(\varphi \models^? x{\leq}y)$ accepts the set of all those paths that are θ-safe for $\varphi \models^? x{\leq}y$.*

Soundness will be proved in Section 8 (Proposition 8) and completeness in Section 9 (Proposition 11).

Lemma 4 *The automata construction for judgments $\varphi \models^? x{\leq}y$ can be performed in deterministic polynomial time in the size of φ.*

PROOF. The closure algorithm can compute all valid judgments of the form $\varphi \vdash u{\leq}v$ where u,v are variables in V_φ in time $O(m^3)$ and store its result in a $O(m^2)$ table where m is the size of φ. The left and right automata have $O(m^2)$ many states. We have to show that we can apply all construction rules in polynomial time. This is non-obvious for the three **descend** rules, at least not at first sight. But note that the set

$$\{z \mid \varphi \vdash x{\leq}i(z)\}$$

can be computed in polynomial time for a given x and $1 \leq i \leq n = ar_f$: First, one computes all variables x' such that $\varphi \vdash x{\leq}x'$ in time $O(m)$. Second, one loops over all such x' while computing all y_i for which there exists some literal $x'{\leq}f(y_1,\ldots,y_n)$ in φ. This requires time $O(m)$ for each individual x'. Finally one loops for all such y_i while computing all z such that $\varphi \vdash y_i{\leq}z$. This can be done in time $O(m)$ again. Thus, we obtain an $O(m^3)$ algorithm to compute the above set for a given x and i.

This set $\{z \mid \varphi \vdash x{\leq}i(z)\}$ has to be computed for all $x \in V_\varphi$ and $1 \leq i \leq n$, i.e., for $O(m^2)$ many pairs. Thus, the overall automata construction requires time at most $O(m^6)$ in the size of φ. (We did not try to improve on this upper bound.)

Theorem 2 (Reduction) *NSSE for a signature $\{\bot, f, \top\}$ can be reduced in deterministic polynomial time to the universality problem of cap automata over the alphabet $\{1, \ldots, ar_f\}$. This holds equally for finite, regular, or possibly infinite trees.*

PROOF. The reduction works as follows. Given an entailment judgment $\varphi \models^? x{\leq}y$ we first test whether φ is satisfiable. This tests can be done in polynomial time as we will prove in Theorem 3 of Section 7. Note that this satisfiability test is the single step of the reduction that differs for finite, resp. regular, or possibly infinite trees.

If φ is unsatisfiable, then entailment holds. Otherwise we construct the left and right automata for $\varphi \models^? x{\leq}y$. This requires at most polynomial time according to Lemma 4. Entailment now holds if and only if the languages of both constructed automata are universal as stated by Propositions 3 and 1.

7 Satisfiability

We now presents satisfiability tests for non-structural subtype constraints. It will be needed to reduce NSSE to universality of cap automata (see Theorem 2). Furthermore, we will rely on the notions introduced throughput this section later on.

We consider the two cases of possibly infinite and resp. regular trees first and then turn to the third and more difficult case of finite trees. Satisfiability of infinite trees was studied by Palsberg and O'Keefe [29], and additionally by Pottier [5]. The case of finite trees was solved by Palsberg, Wand, and O'Keefe [8] [2]. As they do, we will construct solutions with smallest shape for satisfiable constraints. In the infinite and regular cases, we will construct least and greatest solutions. These will prove extremely useful for proving the completeness of our entailment test.

[2] An earlier approach [7] treats the simpler problem with greatest but without least type $\bot$.

C1.	$\varphi \vdash x{\leq}y$ and $x{=}\top \wedge y{=}\bot$ in φ
C2.	$\varphi \vdash x{\leq}y$ and $x{=}\top \wedge y{\leq}f(y_1,\ldots,y_n)$ in φ
C3.	$\varphi \vdash x{\leq}y$ and $f(x_1,\ldots,x_n){\leq}x \wedge y{=}\bot$ in φ

Table 4
Label clashes (the definition of judgments $\varphi \vdash x{\leq}y$ is in Table 1).

7.1 Infinite Trees

A label clash in a constraint φ imposes an unsatisfiable condition on the root label of some tree described by φ. Table 4 collects three kinds of label clashes. Clash *C1* requires $\top{\leq_\Sigma}\bot$, clash *C2* needs $\top{\leq_\Sigma}f$, and *C3* imposes $f{\leq_\Sigma}\bot$.

Lemma 5 *If φ contains a label clash then it is unsatisfiable over possibly infinite trees (and thus also over regular or finite trees).*

PROOF. We only consider the case where φ contains a label clash of form *C1*. The two remaining cases *C2* and *C3* are analogous. If φ contains a label clash of form *C1* then:

$$x{=}\top \text{ in } \varphi,\ y{=}\bot \in \varphi,\ \text{and } \varphi \vdash x{\leq}y$$

for some variables x, y in V_φ. Lemma 2 yields $\varphi \models x{\leq}y$. Thus, $\varphi \models \top{\leq}\bot$ which requires that φ is unsatisfiable.

Proposition 4 *A constraint is satisfiable over the structure of possibly infinite trees if and only if it does not contain any label clash (see Table 4).*

One direction of Proposition 4 coincides with Lemma 5. The converse is proved below.

In order to construct least or greatest solutions, we need the notions of lower and upper bounds in subtree positions. Let u, v be variables and $\pi \in \{1,\ldots,ar_f\}^*$ paths. Let terms $u.\pi$ to denote the subtree of the value of u at node π under the presupposition of existence. We define two first-order formulas by:

$$u{\leq}\pi(v) =_{def} \exists w :\ u{\leq}w \wedge w.\pi = v$$
$$\pi(u){\leq}v =_{def} \exists w :\ w{\leq}v \wedge w.\pi = u$$

The formula $u{\leq}\pi(v)$ means that v is an *upper bound* of u at node π. It is satisfied by variables assignments α where either $\alpha(u)(\pi') = \bot$ for some prefix π' of π or $\alpha(u).\pi \leq \alpha(v)$. Symmetrically, $\pi(u){\leq}v$ states that u is a *lower bound*

$\varphi \vdash x \leq \varepsilon(y)$ if $\varphi \vdash x \leq y$

$\varphi \vdash \varepsilon(x) \leq y$ if $\varphi \vdash x \leq y$

$\varphi \vdash x \leq \pi i(y)$ if $\varphi \vdash x \leq \pi(z),\ z \leq f(z_1, \ldots z_i \ldots, z_n)$ in $\varphi,\ \varphi \vdash z_i \leq y$

$\varphi \vdash \pi i(x) \leq y$ if $\varphi \vdash \pi(z) \leq y,\ f(z_1, \ldots z_i \ldots, z_n) \leq z$ in $\varphi,\ \varphi \vdash x \leq z_i$

Table 5
Syntactic support of lower and upper bounds.

of v at π. It is satisfied by variables assignments α where either $\alpha(v)(\pi') = \top$ for some prefix π' of π or $\alpha(u) \leq \alpha(v).\pi$.

Table 5 defines syntactic support of lower and upper bounds at subtree positions. This generalizes both notions of syntactic support for inequations and children positions in Tables 1 and 2. Note in particular that both judgments $\varphi \vdash x \leq \varepsilon(y)$ and $\varphi \vdash \varepsilon(x) \leq y$ are equivalent to $\varphi \vdash x \leq y$.

Lemma 6 *If $\varphi \vdash \pi(x) \leq y$ then $\varphi \models \pi(x) \leq y$ and if $\varphi \vdash y \leq \pi(x)$ then $\varphi \models y \leq \pi(x)$.*

PROOF. By induction on the derivation of syntactic support. The base case relies on Lemma 3.

Lemma 7 (Decomposition)

(1) If $\varphi \vdash \pi(w) \leq u$ and $\varphi \vdash u \leq \pi\pi'(v)$ then $\varphi \vdash w \leq \pi'(v)$.
(2) If $\varphi \vdash \pi\pi'(v) \leq u$ and $\varphi \vdash u \leq \pi(w)$ then $\varphi \vdash \pi'(v) \leq w$.

PROOF. By induction on the length of the path π.

Proof of Proposition 4 We have to construct solutions for constraints φ that do not have label clashes. The basic idea is to construct the least solution by mapping variables x of V_φ to the least upper bound of all lower bounds of x in φ. Symmetrically, we could also choose the greatest solution.

The construction will use additional judgments $\varphi \vdash \pi(g) \leq x$ where $g \in \Sigma$. Such judgments say that φ syntactically supports g to be a lower bound for the label of x at path π. It is defined as follows:

$\varphi \vdash \pi(\top) \leq x$ if $\varphi \vdash y = \top$ in φ and $\varphi \vdash \pi(y) \leq x$

$\varphi \vdash \pi(f) \leq x$ if $\varphi \vdash f(y_1, \ldots, y_n) \leq y$ in φ and $\varphi \vdash \pi(y) \leq x$

Symmetric judgments of the forms $\varphi \vdash \pi(\top) \leq x$ and $\varphi \vdash \pi(f) \leq x$ are defined in analogy.

We now specify variable assignments least_φ which map variables $z \in V_\varphi$ to the least upper bound of all lower bounds of z in φ. We define $\text{least}_\varphi(z)(\pi)$ by induction on the length of π. Given a word π for which $\text{least}_\varphi(z)(\pi') \notin \{\bot, \top\}$ for all proper prefixes $\pi' < \pi$, we set:

$$\text{least}_\varphi(z)(\pi) = \sup\{\, g \mid \varphi \vdash \pi(g) \leq z \,\}$$

Otherwise, we leave $\text{least}_\varphi(z)(\pi)$ undefined. The definition is sound since all subsets of Σ have a least upper bound. Note also that $\bot$ is the least upper bound of the empty subset of Σ.

Clearly, $\text{least}_\varphi(z)$ is a tree over Σ: its domain is prefix closed and arity consistent by definition. Note that $\text{least}_\varphi(z)$ may be infinite, for instance, if φ is the constraint $f(z) \leq z$.

It remains to show that least_φ is indeed a solution of φ, i.e., that it satisfies all literals of φ. To prove this we distinguish all possible kinds of literals.

(1) Case $z = \bot$ in φ. In this case: $\text{least}_\varphi(z)(\varepsilon) = \bot$. Otherwise, $\varphi \vdash \varepsilon(\top) \leq z$ or $\varphi \vdash \varepsilon(f) \leq z$. But φ then contains a label clash by *C1* or *C3* which contradicts our assumption.

(2) Case $z = \top$ in φ. Obviously, $\text{least}_\varphi(z)(\varepsilon) = \top$.

(3) Case $f(z_1, \ldots, z_n) \leq z$ in φ. Let π be in the common domain of $\text{least}_\varphi(z)$ and $\text{least}_\varphi(f(z_1, \ldots, z_n))$.
   (a) Case $\pi = \varepsilon$. Thus, $\varphi \vdash \varepsilon(f) \leq z$ so that the least solution satisfies $\text{least}_\varphi(z)(\varepsilon) \geq_\Sigma f = \text{least}_\varphi(f(z_1, \ldots, z_n))(\varepsilon)$.
   (b) Case $\pi = i\pi'$ for some $1 \leq i \leq n$. Note that $\text{least}_\varphi(f(z_1, \ldots, z_n))$ is equal to $f(\text{least}_\varphi(z_1), \ldots, \text{least}_\varphi(z_n))$ so that π' must belong to the domain of $\text{least}_\varphi(z_i)$ since π belongs to the domain of $\text{least}_\varphi(f(z_1,\ldots,z_n))$. This implies that $\text{least}_\varphi(z)(\pi')$ is defined and equal to $\sup\{g \mid \varphi \vdash \pi'(g) \leq z_i\}$. But if $\varphi \vdash \pi'(g) \leq z_i$ then $\varphi \vdash i\pi'(g) \leq z$, and hence:

$$\begin{aligned} \text{least}_\varphi(z)(i\pi') &= \sup\{\, g \mid \varphi \vdash \pi(g) \leq z \,\} \\ &\geq_\Sigma \sup\{\, g \mid \varphi \vdash \pi'(g) \leq z_i \,\} \\ &= \text{least}_\varphi(z_i)(\pi') = \text{least}_\varphi(f(z_1, \ldots, z_n))(i\pi') \end{aligned}$$

(4) Case $z \leq f(z_1, \ldots, z_n)$ in φ. Let π be in the common domain of $\text{least}_\varphi(z)$ and $\text{least}_\varphi(f(z_1, \ldots, z_n))$.
   (a) Case $\pi = \varepsilon$. Since φ does not contain any label clash of kind *C2*, it cannot hold that $\varphi \vdash \varepsilon(\top) \leq z$ and hence $\text{least}_\varphi(z)(\varepsilon) \leq_\Sigma f$.
   (b) Case $\pi = i\pi'$ for some $1 \leq i \leq n$. Whenever $\varphi \vdash i\pi'(g) \leq z$ then

$\varphi \vdash \pi'(g) \leq z_i$ by **decomposition** with Lemma 7. Hence:

$$\begin{aligned} \text{least}_\varphi(z)(\pi) &= \sup\{\, g \mid \varphi \vdash \pi'(g) \leq z \,\} \\ &\leq_\Sigma \sup\{\, g \mid \varphi \vdash \pi'(g) \leq z_i \,\} \\ &= \text{least}_\varphi(z_i)(\pi') = \text{least}_\varphi(f(z_1, \ldots, z_n))(i\pi') \end{aligned}$$

By inspection of the proof of Proposition 4 we obtain the following additional result on the existence and form of least and greatest solutions. This result is of its own relevance but also important with respect to entailment.

Proposition 5 *Every constraints that is satisfiable over possibly infinite trees permits a least solution* least_φ *and a greatest solution* great_φ *(over possibly infinite trees). These solutions satisfy for all variables* $z \in V_\varphi$ *and nodes* $\pi \in D_{\text{least}_\varphi(z)}$ *resp.* $\pi \in D_{\text{great}_\varphi(z)}$*:*

$$\begin{aligned} \text{least}_\varphi(z)(\pi) &= \sup\{\, g \mid \varphi \vdash \pi(g) \leq z \,\} \\ \text{great}_\varphi(z)(\pi) &= \inf\{\, g \mid \varphi \vdash z \leq \pi(g) \,\} \end{aligned}$$

PROOF. The proof of Proposition 4 shows that least_φ is indeed solutions of φ; and this solution is clearly smaller than all other solutions of φ. By symmetry, the result for great_φ follows.

7.2 Regular Trees

Least solutions always map variables to regular trees. This has the following consequence:

Proposition 6 *A subtype constraint is satisfiable over finite trees if and only if it is satisfiable over regular trees.*

PROOF. Let φ be a constraint that is satisfiable over finite trees, i.e. which does not have any label clash. We show that the least solution least_φ of φ maps to regular trees only. Proposition 5 shows that the least solution least_φ satisfies:

$$\text{least}_\varphi(z)(\pi) = \sup\{\, g \mid \varphi \vdash \pi(g) \leq z \,\}$$

for all $z \in V_\varphi$ where $\text{least}_\varphi(z)(\pi') \notin \{\bot, \top\}$ for all proper prefixes $\pi' < \pi$, and that $\text{least}_\varphi(z)(\pi)$ is undefined otherwise. We next fix a variable $z \in V_\varphi$ and

C4.	$\exists \pi \neq \varepsilon : \ \varphi \vdash \pi(x) \leq x$ and $\varphi \vdash x \leq y$ and $\varphi \vdash y \leq \pi(y)$

Table 6
Cycle clashes.

show that $\text{least}_\varphi(z)$ has only finitely many distinct subtrees. We define for all paths $\pi \in A_\Sigma$:

$$V_\varphi(z, \pi) = \{\, y \mid \varphi \vdash \pi(y) \leq z \,\}$$

The following claim follows straightforwardly from the definition of least solutions. For all $\pi, \pi' \in D_{\text{least}_\varphi(z)}$:

$$\text{least}_\varphi(z).\pi = \text{least}_\varphi(z).\pi' \text{ if and only if } V_\varphi(z, \pi) = V_\varphi(z, \pi')$$

This claim implies that the number of distinct subtrees of $\text{least}_\varphi(z)$ is uniformly bounded for all $z \in V_\varphi$ by the number of subsets of V_φ, and this number is finite.

7.3 Finite Trees

Satisfiability becomes more tedious in the case of finite trees where we have to care about unsatisfiable cycles. Most typically, $x \leq f(x) \wedge f(x) \leq x$ is unsatisfiable while $f(x) \leq x$ is satisfiable over finite trees. These two examples illustrate that only two-sided cycles in upper and lower bounds are unsatisfiable of finite trees, while one-side cycles can be satisfied. The general form of cycle clashes is given by rule *C4* in Table 6.

Lemma 8 *A constraint with cycle clash is unsatisfiable over finite trees.*

The most problematic fact about satisfiability over finite trees is that satisfiable constraints need not have least or greatest solutions, in contrast to possibly infinite or regular trees (Proposition 5). But fortunately, there always exist solutions with least shape (i.e. least tree domain) for all constraints without label clashes [8]. And least-shape solutions are always finite for constraints without cycle clashes.

The constraint $f(x) \leq x$, for instance, does not have a least solution when interpreted over finite trees. Its finite solutions map x to some tree of the form $f(f(f(\ldots(\top)\ldots)))$, none of which is smaller than all others. The solution mapping of x to $\top$ has the least shape but is greater than all others. Over possible infinite trees, there exists a least solution which maps x to the infinite tree $f(f(f(\ldots)))$.

Proposition 7 *A constraint φ is satisfiable over the structure of finite trees if and only if it does not contain a label clash nor a cycle clash.*

We have already shown the correctness of our clash rules (Lemmas 5 and 8); it remains to prove that constraints without label and cycle clash are satisfiable. This will be the content of Lemmas 9 and 10.

Given a satisfiable constraint φ we now define an assignment s_φ which maps variables to trees with the least possible shape for satisfying φ, i.e., with the least possible tree domain. Let $z \in V_\varphi$ and $\pi \in \{1, \ldots, ar_f\}^*$. We define $\mathsf{s}_\varphi(z)(\pi)$ by induction on the length of π so that we can assume that $\mathsf{s}_\varphi(z)(\pi') = f$ for all proper prefixes of π' of π:

$$\mathsf{s}_\varphi(z)(\pi) = \begin{cases} f & \text{if } \varphi \vdash \pi(f) \leq z \text{ and } \varphi \vdash z \leq \pi(f) \\ \bot & \text{if } \varphi \not\vdash \pi(f) \leq z \text{ and } \varphi \vdash z \leq \pi(g) \text{ where } g \in \{\bot, f\} \\ \top & \text{else} \end{cases}$$

Lemma 9 *If φ is free of label clashes then $\mathsf{s}_\varphi \models \varphi$ over possibly infinite trees.*

PROOF. We show that s_φ satisfies all literals of φ.

(1) Case $z{=}\bot$ in φ. The second clause of the definition of $\mathsf{s}_\varphi(z)(\varepsilon)$ applies since $\varphi \vdash \varepsilon(f){\leq}z$ would prove a label clash of kind *C3* otherwise. Hence $\mathsf{s}_\varphi(z) = \bot$.

(2) Case $z{=}\top$ in φ. The first two clauses of the definition of $\mathsf{s}_\varphi(z)(\varepsilon)$ cannot apply in the absence of label clashes of forms *C1* and *C2*. Hence, $\mathsf{s}_\varphi(z) = \top$.

(3) Case $f(z_1, \ldots, z_n){\leq}z$ in φ. Let π be in the common domain of $\mathsf{s}_\varphi(z)$ and $\mathsf{s}_\varphi(f(z_1, \ldots, z_n))$. Note that $\mathsf{s}_\varphi(f(z_1, \ldots, z_n)) = f(\mathsf{s}_\varphi(z_1), \ldots, \mathsf{s}_\varphi(z_n))$. It remains to prove $\mathsf{s}_\varphi(f(z_1, \ldots, z_n))(\pi) \leq_\Sigma \mathsf{s}_\varphi(z)(\pi)$.
   (a) Subcase $\pi = \varepsilon$. Since $f(z_1, \ldots, z_n){\leq}z$ in φ, $\mathsf{s}_\varphi(z)(\varepsilon)$ cannot be defined by the second clause, and thus $\mathsf{s}_\varphi(z)(\varepsilon) \geq_\Sigma f$.
   (b) Subcase $\pi = i\pi'$ for some $1 \leq i \leq n$. We must prove $\mathsf{s}_\varphi(z_i)(\pi') \leq_\Sigma \mathsf{s}_\varphi(z)(\pi)$ under the assumption that both values are defined. Clearly, this holds in the case $\mathsf{s}_\varphi(z)(\pi) = \top$. If $\mathsf{s}_\varphi(z)(\pi) =_\Sigma f$ then $\varphi \vdash z{\leq}\pi(f)$ so that the decomposition Lemma 7 yields $\varphi \vdash z_i{\leq}\pi'(f)$. Hence, $\mathsf{s}_\varphi(z_i)(\pi') \leq_\Sigma f = \mathsf{s}_\varphi(z)(\pi)$. Otherwise, $\mathsf{s}_\varphi(z)(\pi) = \bot$ so that $\varphi \not\vdash \pi(f){\leq}z$ and $\varphi \vdash z{\leq}\pi(g)$ for some $g \in \{f, \bot\}$. Hence, $\varphi \not\vdash \pi'(f){\leq}z_i$ and $\varphi \vdash z_i{\leq}\pi'(g)$. This implies $\mathsf{s}_\varphi(z_i)(\pi') = \bot$.

(4) Case $z{\leq}f(z_1, \ldots, z_n)$ in φ. For given a word π in the domains of $\mathsf{s}_\varphi(z)$ and $\mathsf{s}_\varphi(f(z_1, \ldots, z_n))$ we prove $\mathsf{s}_\varphi(z)(\pi) \leq_\Sigma \mathsf{s}_\varphi(f(z_1, \ldots, z_n))(\pi)$.
   (a) Subcase $\pi = \varepsilon$. Since $z{\leq}f(z_1, \ldots, z_n)$ in φ one of the first two clauses defines $\mathsf{s}_\varphi(z)(\varepsilon)$ so that $\mathsf{s}_\varphi(z)(\varepsilon) = f$ as required.
   (b) Subcase $\pi = i\pi'$ for some $1 \leq i \leq n$. We show $\mathsf{s}_\varphi(z)(\pi) \leq_\Sigma \mathsf{s}_\varphi(z_i)(\pi')$. This clearly holds if $\mathsf{s}_\varphi(z_i)(\pi') = \top$. If $\mathsf{s}_\varphi(z_i)(\pi') = \bot$ then $\varphi \not\vdash \pi'(f){\leq}z_i$ and $\varphi \vdash z_i{\leq}\pi'(g)$ for some $g \in \{\bot, f\}$. Hence, $\varphi \not\vdash \pi(f){\leq}z$

according to the decomposition Lemma 7. Furthermore, $\varphi \vdash z \leq \pi(g)$ so that $\mathsf{s}_\varphi(z)(\pi) = \bot$. The final case is $\mathsf{s}_\varphi(z_i)(\pi') = f$. Now, $\varphi \vdash z_i \leq \pi'(f)$ and also $\varphi \vdash \pi'(f) \leq z_i$. Thus, $\varphi \vdash z \leq \pi(f)$ so that $\mathsf{s}_\varphi(z)(\pi) \leq_\Sigma f$.

Lemma 10 *If φ doesn't contain cycle clashes then $\mathsf{s}_\varphi(z)$ is finite for all $z \in V_\varphi$.*

PROOF. Suppose that $\mathsf{s}_\varphi(z)$ is infinite for some $z \in V_\varphi$. Then there exists an infinite word ω over the alphabet $\{1, \ldots, ar_f\}$ such that $\mathsf{s}_\varphi(z)(\pi) = f$ for all finite prefixes π of ω. This yields $\varphi \vdash \pi(f) \leq z$ and $\varphi \vdash z \leq \pi(f)$ for all such prefixes. Hence, there exists variables $x_\pi, y_\pi \in V_\varphi$ for all prefixes π of ω such that $z = x_\varepsilon = y_\varepsilon$ and for all prefixes $\pi\pi'$ of ω:

$$\varphi \vdash \pi'(x_{\pi\pi'}) \leq x_\pi \qquad \text{and} \qquad \varphi \vdash x_\pi \leq y_\pi \qquad \text{and} \qquad \varphi \vdash y_\pi \leq \pi'(y_{\pi\pi'})$$

Since there are only finitely many distinct pairs (x_π, y_π) of variables in V_φ, at least one such pair must be repeated for sufficiently large π and π'. This pumping argument establishes a cycle clash *C4* in φ which contradicts our assumption. Thus, $\mathsf{s}_\varphi(z)$ must be finite for all $z \in V_\varphi$.

7.4 Summary

So far, we have seen that we can decide satisfiability of non-structural subtype constraints by testing for the existence of different kinds of clashes.

Definition 2 *We call a constraint φ* clash free *for the structure of*

(1) possibly infinite trees if it does not contain any label clash,
(2) regular trees if it does not contain any label clash,
(3) finite trees if it does neither contain a label clash nor a cycle clash.

We now consider efficiency issues. The existence of a label clash in a constraint φ can be tested in cubic time in the size of φ. First, one computes a table of quadratic size that stores all valid judgments $\varphi \vdash u \leq v$. Second, one compares the labels required by φ for all u and v with $\varphi \vdash u \leq v$. The existence of a cycle clash can also be tested in cubic time. All together, this yields the following result:

Theorem 3 *A subtype constraint φ is satisfiable (over finite, regular, resp. possibly infinite trees) if and only if it is clash-free (over finite, regular, resp. possibly infinite trees). In all three cases, satisfiability can be tested in cubic time in the size of φ. Least and greatest solutions of satisfiable constraints exist for the infinite and regular cases, but not necessarily over finite trees.*

PROOF. From Propositions 4, 5, 6, 7 and the above discussion on efficiency.

8 Soundness

In this section, we prove the soundness of the automata construction. The proof is non-trivial and requires a new argument compared to [16]. This argument (see the proof of Proposition 10) is based on Lemma 1 from Section 3.

Proposition 8 (Soundness) *For all φ, variables x, y, and sides $\theta \in \{l, r\}$ it holds that all paths accepted by $\mathcal{P}_\theta(\varphi \models^? x{\leq}y)$ are θ-safe for $\varphi \models^? x{\leq}y$.*

We only consider the left side $\theta = l$. We proceed in two steps: we first treat accepted words in class A (Proposition 9) and second in class P (Proposition 10). For both steps, we have to characterize transitions of the constructed automata.

Lemma 11 (Transitions without *all* states) *For all constraints φ, variables x, y, u, v, sides $\theta \in \{l, r\}$, and nonempty words $\pi \in \{1, \ldots, ar_f\}^+$:*

(1) $\mathcal{P}_\theta(\varphi \models^? x{\leq}y) \vdash (u, s) \xrightarrow{\pi} (v, s')$ *for some s, s' if and only if* $\varphi \vdash u{\leq}\pi(v)$.

(2) $\mathcal{P}_\theta(\varphi \models^? x{\leq}y) \vdash (s, u) \xrightarrow{\pi} (s', v)$ *for some s, s' if and only if* $\varphi \vdash \pi(v){\leq}u$.

This lemma would fail for $\pi = \varepsilon$. But this does not matter since if $\varphi \vdash u{\leq}v$ then **reflexivity** yields for all non-empty words π:

$$\mathcal{P}_\theta(\varphi \models^? x{\leq}y) \vdash \underline{\underline{(u, v)}} \xrightarrow{\pi} \underline{\underline{\mathit{all}}}$$

PROOF. We only prove the first statement for the upper bounds for the implication from right to left. (The second property is analogous but for lower bounds.) The proof is by induction on the length of $\pi \neq \varepsilon$.

First, assume $\pi = i$ is a single letter path where $i \in \mathcal{A}_\Sigma$. The supported upper bound $\varphi \vdash u{\leq}i(v)$ permits to apply the **descend** rule. Hence $\mathcal{P}_\theta(\varphi \models^? x{\leq}y) \vdash (u, \_) \xrightarrow{i} (v, \_)$ for arbitrary $\theta \in \{l, r\}$.

Second, consider a path π of length at least two. We can decompose π into $\pi = \pi' i$ for some nonempty path π' and $i \in \mathcal{A}_\Sigma$. Also we can decompose $\varphi \vdash u{\leq}\pi(v)$ into $\varphi \vdash u{\leq}\pi'(v')$ and $\varphi \vdash v'{\leq}i(v)$ for some variable v'. The induction hypothesis applies twice and yields $\mathcal{P}_\theta(\varphi \models^? x{\leq}y) \vdash (u, \_) \xrightarrow{\pi} (v', \_)$ and $\vdash (v', \_) \xrightarrow{i} (v, \_)$.

Lemma 12 (Bounds and safety)

(1) If $\alpha \models x{\leq}\pi(u)$ then all proper prefixes of π are l-safe for $\alpha \models^? x{\leq}y$.
(2) If $\alpha \models x{\leq}\pi(u) \wedge u{=}\bot$ then all paths $\pi\pi'$ with $\pi' \in A^*_\Sigma$ are l-safe for $\alpha \models^? x{\leq}y$.
(3) If $\alpha \models x{\leq}\pi(u) \wedge \pi(u){\leq}y$ then all paths $\pi\pi'$ with $\pi' \in A^*_\Sigma$ are l-safe for $\alpha \models^? x{\leq}y$.

PROOF. We only prove 1 (the other two cases are similar). Let π' be a proper prefix of π. Every solution $\alpha \models x{\leq}\pi(u)$ satisfies either $\alpha(x)(\pi') =_\Sigma f$ or there exists a path $\nu < \pi'$ with $\alpha(x)(\nu) =_\Sigma \bot$; thus all π' are l-safe for $\alpha \models^? x{\leq}y$.

Proposition 9 (Soundness for class A) *For all φ, variables x, y it holds that all paths π with class* A *accepted by $\mathcal{P}_l(\varphi \models^? x{\leq}y)$ are l-safe for $\varphi \models^? x{\leq}y$.*

PROOF. We have to consider all recognizing transitions of the constructed finite automaton. According to the automaton construction there are three possibilities for doing this.

(1) Assume that the path is accepted in a state to which the **final states** rule applies, i.e. π is recognized by a transition of the following form:

$$\mathcal{P}_l(\varphi \models^? x{\leq}y) \vdash \ \gtrdot(x,y) \xrightarrow{\pi} \underline{\underline{(u,s)}} \xrightarrow{i} (u',s').$$

Lemma 11 yields $\varphi \models x{\leq}\pi i(u')$. Thus π is l-safe for $\varphi \models^? x{\leq}y$ by Lemma 12 case 1.

(2) Assume π is accepted by the **reflexivity** rule. Then, π must be of the form $\pi_1\pi_2$, where we can identify transition of the following form where $\varphi \vdash u{\leq}u'$:

$$\mathcal{P}_l(\varphi \models^? x{\leq}y) \vdash \ \gtrdot(x,y) \xrightarrow{\pi_1} \underline{\underline{(u,u')}}.$$

Lemma 11 yields $\varphi \models x{\leq}\pi_1(u)$. By symmetrical reasoning $\varphi \models \pi_1(u'){\leq}y$ and thus $\varphi \models \pi_1(u){\leq}y$. Case 3 of Lemma 12 shows that π is l-safe for $\varphi \models^? x{\leq}y$.

(3) Assume π is accepted by the **bot** rule (the case that the **top** rule fires is analogous). Then, π must be of the form $\pi_1\pi_2$, where we can identify transition of the following form:

$$\mathcal{P}_l(\varphi \models^? x{\leq}y) \vdash \ \gtrdot(x,y) \xrightarrow{\pi_1} \underline{\underline{(u,s)}},\ \varphi \vdash u{\leq}u',\ \text{and } u'{=}\bot \text{ in } \varphi.$$

Again Lemma 11 yields $\varphi \models x{\leq}\pi_1(u')$. Therefore π is l-safe for $\varphi \models^? x{\leq}y$ by Lemma 12 case 2.

We now approach the soundness of P-edges. It mainly relies on Lemma 13 in combination with Lemma 1 on word equations.

Lemma 13 (Safety and word equations) *Let $\pi \neq \varepsilon$ be a path, u, v variables, and $\alpha \models u{\leq}\pi(v)$. All words π' with $\pi' \in pr(\pi\pi')$ are l-safe for $\alpha \models^? u{\leq}v$.*

PROOF. We distinguish whether π' belongs to $D_{\alpha(v)}$ or not.

a. Case $\pi' \in D_{\alpha(v)}$. It follows in this case from $\alpha \models u{\leq}\pi(v)$, that $\alpha \models \exists v'(u{\leq}\pi\pi'(v'))$. By Lemma 12 all proper prefixes of $\pi\pi'$ are l-safe for $\alpha \models^? u{\leq}v$. Thus π' has this property since π' is a prefix of $\pi\pi'$ and $\pi \neq \varepsilon$ by assumption.
b. Case $\pi' \notin D_{\alpha(v)}$. Let π'' be the maximal prefix of π' in $D_{\alpha(v)}$. Hence, $\alpha(v)(\pi'') \in \{\bot, \top\}$. First we assume the case $\alpha(v)(\pi'') =_\Sigma \top$ which implies that all paths σ with $\pi'' \leq \sigma$, in particular π', are l-safe. Second we assume the left case $\alpha(v)(\pi'') =_\Sigma \bot$. Since $\alpha \models u{\leq}\pi(v)$, there exists a path π''' with $\pi''' \leq \pi\pi'$ such that $\alpha(u)(\pi''') =_\Sigma \bot$. Both together, $\pi''' \leq \pi\pi'$ and the assumption $\pi' \leq \pi\pi'$ show that the paths π''' and π' are comparable: if $\pi' \geq \pi'''$ then π' is l-safe according to the definition of l-safe. Otherwise, $\pi' < \pi'''$ holds and $\alpha(u)(\pi') =_\Sigma f$ implies π' to be l-safe.

Lemma 14 (Composing safety) *If $\alpha \models x{\leq}\pi(u)$, $\alpha \models \pi(v){\leq}y$, and π' is l-safe for $\alpha \models^? u{\leq}v$ then $\pi\pi'$ is l-safe for $\alpha \models^? x{\leq}y$.*

PROOF. It follows from the assumption π' is l-safe for $\alpha \models^? u{\leq}v$ that $\alpha(u)(\pi') =_\Sigma f$ or that there exists $\pi''{\leq}\pi'$ with $\alpha(u)(\pi'') =_\Sigma \bot$ or $\alpha(v)(\pi'') =_\Sigma \top$. The assumption $\alpha \models x{\leq}\pi(u)$ and $\alpha \models \pi(v){\leq}y$ imply that $\alpha(x)(\pi\pi') =_\Sigma f$ or there exists $\pi''{\leq}\pi\pi'$ with $\alpha(x)(\pi'') =_\Sigma \bot$ or $\alpha(y)(\pi'') =_\Sigma \top$; thus $\pi\pi'$ is l-safe for $\alpha \models^? x{\leq}y$.

Proposition 10 (Soundness for class P) *For all φ and variables x, y, all paths of class* P *accepted by $\mathcal{P}_l(\varphi \models^? x{\leq}y)$ are l-safe for $\varphi \models^? x{\leq}y$.*

PROOF. A path ν of class P can only be recognized by using a P-edge. Thus, there exist words μ, μ', π such that $\nu = \pi\mu'$, $\mu' \in pr(\mu^*)$ and for some $u, v \in V_\varphi \cup \{x, y\}$ and $s \in V_\varphi \cup \{x, y, \_\}$:

$$\mathcal{P}_l(\varphi \models^? x{\leq}y) \vdash (x, y) \xrightarrow{\pi} \underline{\underline{(u, v)}} \xrightarrow{\mu} (v, s) \dashrightarrow \underline{\underline{(u, v)}}$$

Lemma 11 yields $\varphi \models x{\leq}\pi(u)$, $\varphi \models \pi(v){\leq}y$, and $\varphi \models u{\leq}\mu(v)$. We fix an arbitrary solution $\alpha \models \varphi$ and show that ν is l-safe for $\alpha \models x{\leq}y$. Note that $\mu \neq \varepsilon$ since ν would belong to class A otherwise. We can thus apply **Lemma**

1 on word equations to our assumption $\mu' \in pr(\mu^*)$ to derive $\mu' \in pr(\mu\mu')$. This verifies the assumptions of Lemma 13 which shows that μ' is l-safe for $\alpha \models^? u{\leq}v$. Finally, the composition Lemma 14 shows that $\pi\mu'$ is l-safe for $\alpha \models^? x{\leq}y$ as required.

9 Completeness

We prove the completeness of the automata construction. This proof was not given in the conference version of this article [24] and is simplified in many aspects compared to its relatives [16].

Proposition 11 (Completeness) *Let φ be a constraint with variables $x, y \in V_\varphi$ and $\theta \in \{l, r\}$ a side. If φ is satisfiable (for finite, regular, resp. possibly infinite trees) then $\mathcal{P}_\theta(\varphi \models^? x{\leq}y)$ accepts all paths that are θ-safe for $\varphi \models^? x{\leq}y$ (with respect to the considered structure of finite, regular, resp. possibly infinite trees).*

In a first step, we reduce Proposition 11 to simpler statements in Proposition 12 and 13. By symmetry, we can restrict ourselves to the case of r-safety. This notion can be reformulated on basis of standard logical transformations.

Lemma 15 *Let ν be a word in $\{1, \ldots ar_f\}^*$ and $\varphi \models^? x{\leq}y$ and entailment judgment. Then ν is not r-safe for $\varphi \models^? x{\leq}y$ if and only if $\varphi \wedge \mathrm{pref}^\perp_\nu(y) \wedge \neg\mathrm{pref}^\perp_\nu(x)$ is satisfiable.*

PROOF. The word ν is r-safe for $\varphi \models^? x{\leq}y$ iff $\varphi \models \mathrm{pref}^\perp_\nu(y) \rightarrow \mathrm{pref}^\perp_\nu(x)$ iff $\varphi \wedge \neg(\mathrm{pref}^\perp_\nu(y) \rightarrow \mathrm{pref}^\perp_\nu(x))$ is unsatisfiable, i.e., if $\varphi \wedge \mathrm{pref}^\perp_\nu(y) \wedge \neg\mathrm{pref}^\perp_\nu(x)$ is satisfiable.

In order to prove Proposition 11 we can assume a satisfiable constraint φ with variables x, y, and a path ν in $\{1, \ldots, ar_f\}^*$ that does not belong to the language of $\mathcal{P}_l(\varphi \models^? x{\leq}y)$. We then have to prove that ν is not r-safe for $\varphi \models^? x{\leq}y$. Using the above Lemma, this is equivalent to the satisfiability of the following formula:

$$\varphi \wedge \mathrm{pref}^\perp_\nu(y) \wedge \neg\mathrm{pref}^\perp_\nu(x)$$

We can eliminate the negative subformula $\neg\mathrm{pref}^\perp_\nu(x)$ by a simple trick. Let

$$x_\varepsilon =_{\mathrm{def}} x$$

and fix a set $F_\nu(x)$ of fresh and distinct variables x_μ for finitely many nonempty paths μ that are successors of prefixes of ν:

$$F_\nu(x) =_{\text{def}} \{\ x_{\pi i} \mid \varepsilon \leq \pi \leq \nu,\ \ 1 \leq i \leq ar_f\ \}$$

Note that $F_\nu(x)$ does *not* contain x_ε. Next, we define a constraint $\text{low}_{\nu(f)}(x)$ which imposes the lower bound $\nu(f)$ on x:

$$\text{low}_{\nu(f)}(x)\ =_{\text{def}} \bigwedge_{\varepsilon \leq \pi \leq \nu} f(x_{\pi 1}, \ldots, x_{\pi n}) \leq x_\pi$$

Lemma 16 *The constraint* $\text{low}_{\nu(f)}(x)$ *is satisfaction equivalent to* $\neg\text{pref}^\perp_\nu(x)$.

PROOF. Indeed, the existential formula $\exists F_\nu(x).\ \text{low}_{\nu(f)}(x)$ is equivalent to $\neg\text{pref}^\perp_\nu(x)$. Note first that $\text{low}_{\nu(f)}(x) \vdash \nu(f){\leq}x$ and hence, $\exists F_\nu(x).\ \text{low}_{\nu(f)}(x) \models \nu(f){\leq}x$. Clearly, the converse holds as well, i.e., both formulas are equivalent. Finally, note that $\nu(f){\leq}x$ is also equivalent to $\neg\text{pref}^\perp_\nu(x)$.

According to Lemma 16, the remaining goal is to prove the satisfiability of the formula: $\varphi \wedge \text{pref}^\perp_\nu(y) \wedge \text{low}_{\nu(f)}(x)$. The cases of possibly infinite or regular trees will be proved in Proposition 12. The existence of finite solutions is then derived from the existence of possibly infinite solutions in Proposition 13. Before proving these proposition, we formulate some needed properties of the constraint $\text{low}_{\nu(f)}(x)$ in two technical Lemmas 17 and 18.

Lemma 17 *The following properties hold for a constraint* φ *with variables* $x, u, v \in V_\varphi$, *words* $\nu, \pi, \mu \in \{1, \ldots, ar_f\}^*$ *such that* $x_\mu \in F_\nu(x)$. *Recall that* $F_\nu(x)$ *is a set of fresh variables disjoint from* V_φ. *Let* $\varphi' = \varphi \wedge \text{low}_{\nu(f)}(x)$.

(1) $\varphi' \vdash u{\leq}\pi(v)$ *if and only if* $\varphi \vdash u{\leq}\pi(v)$.
(2) $\varphi' \vdash x_\mu{\leq}\pi(v)$ *if and only if* $\varphi \vdash x{\leq}\mu(x')$, $\varphi \vdash x'{\leq}\pi(v)$ *for some* x'.
(3) $\varphi' \vdash \pi(u){\leq}v$ *if and only if* $\varphi \vdash \pi(u){\leq}v$.
(4) $\varphi' \vdash \pi(x_\mu){\leq}v$ *if and only if* $\varphi \vdash x{\leq}\mu'(x')$ *and* $\varphi \vdash \pi'(x'){\leq}v$
where $\mu{=}\mu'\nu$, $\pi{=}\pi'\nu$ *for some* μ', π', ν, x'.
(5) $\varphi' \vdash \mu' i(x_{\mu\mu' i}){\leq}x_\mu$ *if* $\mu\mu' \leq \nu$ *and* $i \in \{1, \ldots, ar_f\}$.
(6) $\varphi' \vdash \varepsilon(x_\mu){\leq}x_\mu$ *and* $\varphi' \vdash x_\mu{\leq}\varepsilon(x_\mu)$.

PROOF. The proofs of these properties are tedious but not difficult. Note that (5) and (6) are trivial but they will simplify the proof.

All inverse implications are straightforward so that we only treat the most complicated property (4) explicitly. Let the right hand side of (4) be true. Since $\mu \in V_\nu(X)$ we have $\text{low}_{\nu(f)}(x) \vdash \mu(x_\mu){\leq}x$ and thus $\text{low}_{\nu(f)}(x) \vdash \mu'\nu(x_\mu){\leq}x$. Because of $\varphi \vdash x{\leq}\mu'(x')$ we can apply the decomposition Lemma 11 which

yields $\varphi' \vdash \nu(x_\mu) \leq x'$. Combined with $\varphi \vdash \pi'(x') \leq v$ this implies $\varphi' \vdash \pi'\nu(x_\mu) \leq v$ and hence $\varphi' \vdash \pi(x_\mu) \leq v$ as required.

We prove all remaining implications of (1)–(5) together with (6) simultaneously. For this we define a new set $C_{\varphi'}$ of path constraints: a constraint ψ is in $C_{\varphi'}$ if and only if one of the properties (1)–(6) licenses $\varphi' \vdash \psi$. In the rest of the proof we do not distinguish between $\varepsilon(x) \leq y$ in $C_{\varphi'}$ and $x \leq \varepsilon(y)$ in $C_{\varphi'}$. We also write $x \leq y$ in $C_{\varphi'}$ in each of both cases. It remains to prove that $C_{\varphi'}$ is closed under the conditions given in Table 5 of lower and upper bounds. We restricted ourself to prove that $C_{\varphi'}$ is again closed under reflexivity, transitivity, decomposition. The set $C_{\varphi'}$ is closed under reflexivity for all variables in V_φ by property (1) and for all variables in $F_\nu(x)$ by (6). For transitivity and decomposition we have to consider literals of the restricted form $u \leq v$ where u and v are distinct variables. Such literals are defined in (1)–(4) where $\pi = \varepsilon$. They are not defined in (5) or (6).

We prove that $C_{\varphi'}$ is closed under transitivity. There is only one interesting case left where the transitivity rule can be applied. Let $v \leq u$ in $C_{\varphi'}$ with $v, u \in V_\varphi$ be contributed by $\varphi \vdash v \leq u$ in the case of property (1) or (3). Also let $x_\mu \leq v$ in $C_{\varphi'}$ be contributed by (2) or (4) which require $\varphi \vdash x \leq \mu(x')$, $\varphi \vdash x' \leq v$ for some x, x', v, μ. Following Lemma 7 it holds $\varphi \vdash x \leq \mu(u)$ and thus, $x_\mu \leq u$ in $C_{\varphi'}$ again by (2).

We prove $C_{\varphi'}$ to be closed under decomposition.

(1) Assume $u \leq f(\ldots, u_i, \ldots) \wedge f(\ldots, v_i, \ldots) \leq v$ in φ for variables $u, u_i, v, v_i \in V_\varphi$. Also assume $v \leq u$ in $C_{\varphi'}$ by (1) or (3) contributed by $\varphi \vdash v \leq u$. Then, $\varphi \vdash v_i \leq u_i$ and also $v_i \leq u_i$ in $C_{\varphi'}$ by (2).
(2) Assume $u \leq f(\ldots, u_i, \ldots)$ in φ with $u, u_i \in V_\varphi$ and $f(x_1, \ldots, x_n) \leq x_\varepsilon$ in $\mathrm{low}_{\nu(f)}(x)$. Note that $x_\varepsilon = x$, $x \in V_\varphi$ by assumption. Also assume $x \leq u$ in $C_{\varphi'}$ which is contributed by (1) or (3) with $\varphi \vdash x \leq u$. Then, $x_i \leq u_i$ in $C_{\varphi'}$ by (2).
(3) Assume $u \leq f(\ldots, u_i, \ldots)$ in φ with $u, u_i \in V_\varphi$ and $f(x_{\pi 1}, \ldots, x_{\pi n}) \leq x_\pi$ in $\mathrm{low}_{\nu(f)}(x)$ where $\pi \neq \varepsilon$. Thus, $x_\pi \in F_\nu(x)$. Then, $x_i \leq u_i$ in $C_{\varphi'}$ by (2).

Lemma 18 *Let $\nu \in \{1, \ldots, ar_f\}^*$ and let φ be a constraint with variables x, y. It holds that $\varphi \wedge \mathrm{low}_{\nu(f)}(x) \vdash \nu(f) \leq y$ if and only if*

(1) $\varphi \vdash \nu(f) \leq y$, or
(2) there exist a state z and paths $\pi_1 \leq \nu$, π_2 such that $\varphi \vdash \pi_1\pi_2(z) \leq y$, $\varphi \vdash x \leq \pi_1(z)$, and if $\pi_2 \neq \varepsilon$ then also $\nu \in \pi_1\, pr(\pi_2^*)$.

PROOF. From right to left. Clearly, $\varphi \vdash \nu(f) \leq y$ implies $\varphi \wedge \mathrm{low}_{\nu(f)}(x) \vdash \nu(f) \leq y$. So let $\varphi \vdash \pi_1\pi_2(z) \leq y$ and $\varphi \vdash x \leq \pi_1(z)$. Let ν' be an arbitrary path and $\nu = \pi_1\nu'$. If $\psi \vdash \pi_1\pi_2(z) \leq y$, $\psi \vdash x \leq \pi_1(z)$, and $\psi \vdash \pi_1\nu'(f) \leq x$ then

$\psi \vdash \pi_1\pi_2\nu'(f){\leq}y$ according to Lemma 7. Note that $\mathrm{low}_{\nu(f)}(x) \vdash \nu(f){\leq}x$. Thus, $\varphi \wedge \mathrm{low}_{\nu(f)}(x) \vdash \pi_1\pi_2\nu'(f){\leq}y$ which also implies $\varphi \wedge \mathrm{low}_{\nu(f)}(x) \vdash \pi_1\pi(f){\leq}y$ for every prefix $\pi \leq \pi_2\nu'$. The case $\pi = \nu'$, which was to prove, holds for $\pi_2 = \varepsilon$ or $\nu' \in pr(\pi_2^*)$ according to Lemma 1.

From left to right. Let assume $\varphi \wedge \mathrm{low}_{\nu(f)}(x) \vdash \nu(f){\leq}y$, $y \in V_\varphi$ which is equivalent to $\varphi \wedge \mathrm{low}_{\nu(f)}(x) \vdash \nu(u){\leq}y$ together with either (1) $\varphi \vdash \varepsilon(f){\leq}u$ where $u \in V_\varphi$ or (2) $\mathrm{low}_{\nu(f)}(x) \vdash \varepsilon(f){\leq}u$ where (2a) $u = x$ or (2b) $u \in F_\nu(x)$. Let us assume case (1). By Lemma 17 also $\varphi \vdash \nu(u){\leq}y$ since $y, u \in V_\varphi$. Our case assumption $\varphi \vdash \varepsilon(f){\leq}u$ implies $\varphi \vdash \nu(f){\leq}y$.

Next, assume case (2a). Again by Lemma 17 also $\varphi \vdash \nu(x){\leq}y$ since $y, u \in V_\varphi$. Then, there exist $\pi_1 = \varepsilon, \pi_2 = \nu, z$ with $\varphi \vdash \pi_1\pi_2(z){\leq}y$ and $\varphi \vdash x{\leq}z$. Also $\nu \in pr(\pi_1^*)$.

At last assume the remaining case (2b). Our assumption $\mathrm{low}_{\nu(f)}(x) \vdash \varepsilon(f){\leq}u$, $u \in F_\nu(x)$ holds in the case $u = x_\pi$ where $\pi \leq \nu$. Lemma 17.4 together with our assumption $\varphi \wedge \mathrm{low}_{\nu(f)}(x) \vdash \nu(x_\pi){\leq}y$ implies $\varphi \vdash x{\leq}\pi'(x')$, $\varphi \vdash \nu'(x'){\leq}y$ where $\pi{=}\pi'\nu''$, $\nu{=}\nu'\nu''$ for some ν', ν'', x'. Since $\pi \leq \nu$, we get $\pi'\nu'' \leq \nu = \nu'\nu''$. This implies $\pi' \leq \nu'$. So, let $\nu' = \pi'\pi''$ for some π''. Then we identify $\varphi \vdash \pi'\pi''(x'){\leq}y$ and $\varphi \vdash x{\leq}\pi'(x')$ where $\pi' \leq \nu$. Since $\pi'\nu'' \leq \nu'\nu''$ and $\nu' = \pi'\pi''$ it holds that $\pi'\nu'' \leq \pi'\pi''\nu''$ and thus, $\nu'' \leq \pi''^*$ for $\pi'' \neq \varepsilon$ according to Lemma 1. It holds that $\nu = \pi'\pi''\nu'' \in \pi'\ pr(\pi''^*)$ in the case $\pi'' \neq \varepsilon$.

We now return to the main line of the completeness proof, i.e., we show that $\varphi \wedge \mathrm{low}_{\nu(f)}(x) \wedge \mathrm{pref}_\nu^\perp(y)$ is satisfiable.

Proposition 12 *Consider the structure of possibly infinite or of regular trees respectively. Let ν be a word in $\{1, \ldots, ar_f\}^*$ that does not belong to the language of $\mathcal{P}_r(\varphi \models^? x{\leq}y)$. Let φ be a satisfiable constraint with variables $x, y \in V_\varphi$. Then the least solution of the constraint $\varphi \wedge \mathrm{low}_{\nu(f)}(x)$ exists and satisfies $\mathrm{pref}_\nu^\perp(y)$ simultaneously.*

PROOF. Let $\varphi' =_{\mathrm{def}} \varphi \wedge \mathrm{low}_{\nu(f)}(x)$. In order to show that φ' permits a least solution, we first show that it does not contain any label clash (Theorem 3). Assume the contrary, i.e., that φ' contains a label clash. Then there exists variables $u, v \in V_{\varphi'}$ with $\varphi' \vdash u{\leq}v$ such that φ' requires contradicting label bounds for u and v according to $C1$, $C2$, or $C3$ in Table 4. Since $\mathrm{low}_{\nu(f)}(x)$ does not impose any upper label bounds, it follows that $v \in V_\varphi$. If the lower bound for u belongs already to φ then φ contains a label clash. Hence, $\mathrm{low}_{\nu(f)}(x)$ imposes the lower bound on u. But the only lower bounds that $\mathrm{low}_{\nu(f)}(x)$ imposes are f-bounds for some variables x_π; thus $u = x_\pi$ for some π. Furthermore, lower f-bounds clash only with upper $\perp$-bounds, i.e., we have a label clash of kind

$C3$:

$$f(...)\leq x_\pi \text{ in } \mathrm{low}_{\nu(f)}(x), \quad \varphi' \vdash x_\pi \leq v, \quad \text{and} \quad v{=}\bot \text{ in } \varphi.$$

Because of $f(...)\leq x_\pi$ in $\mathrm{low}_{\nu(f)}(x)$ it follows that $\varepsilon \leq \pi \leq \nu$. We next show that $\varphi \vdash x\leq\pi(v)$. If $\pi = \varepsilon$ then $x_\pi \in V_\varphi$ so that part 1 of Lemma 17 yields $\varphi \vdash x\leq\pi(v)$. Otherwise, $\pi \neq \varepsilon$ so that $x_\pi \in F_\nu(x)$. Part 2 of Lemma 17 yield $\varphi \vdash x\leq\pi(v)$ in this case, so it holds in all cases.

Lemma 11 implies that the automaton $\mathcal{P}_r(\varphi \models^? x\leq y)$ reaches state $(v, \_)$ over word π. Since $v{=}\bot$ in φ, the **bot** rule of the automata construction (Table 3) lets $\mathcal{P}_r(\varphi \models^? x\leq y)$ accepts all words that π is a prefix of. And $\pi \leq \nu$ so that ν is accepted by the right automaton, in contrast to our assumption.

Recall that the least solution $\mathrm{least}_{\varphi'}$ is regular but possibly infinite. In order to prove $\mathrm{least}_{\varphi'} \models \mathrm{pref}^{\bot}_{\nu}(y)$ we assume the contrary. By definition of $\mathrm{least}_{\varphi'}$ (Section 7.3) the contrary holds if and only if φ' one of the following lower bounds for y: either $\pi(\top)$ for some prefix $\pi \leq \nu$ or $\nu(f)$:

$$\varphi' \vdash \pi(\top)\leq y \quad \text{or} \quad \varphi' \vdash \nu(f)\leq y$$

In the first case, there exists some equation $z{=}\top$ in φ' such that $\varphi' \vdash \pi(z)\leq y$. But $z{=}\top$ cannot belong to $\mathrm{low}_{\nu(f)}(x)$. It thus belongs to φ so that $z \in V_\varphi$. Part 3 of Lemma 17 yields $\varphi \vdash \pi(z)\leq y$. Lemma 11 shows that the automaton $\mathcal{P}_r(\varphi \models^? x\leq y)$ reaches state $(\_, z)$ over word π. Because of the **top** rule of the automata construction (Table 3) $\mathcal{P}_r(\varphi \models^? x\leq y)$ accepts all words of which π is a prefix, and thus ν, in contrast to our assumption.

The remaining second case $\varphi' \vdash \nu(f)\leq y$ is the crucial step in this proof. Lemma 18 leaves only two possibilities that we distinguish:

(1) Case $\varphi \vdash \nu(f)\leq y$. Hence, there exists z such that $\varphi \vdash \nu(z)\leq y$ and $f(\ldots) \leq z$ in φ. Lemma 11 proves that $\mathcal{P}_r(\varphi \models^? x\leq y)$ can reach state $(\_, z)$ over word ν. And this state is final since the **final state** rule applies given $f(\ldots) \leq z$ in φ.

(2) In the other case, there exist $\pi_1 \leq \nu$, π_2, and z with $\varphi \vdash \pi_1\pi_2(z)\leq y$, $\varphi \vdash x\leq\pi_1(z)$.
   (a) In the case $\pi_1 = \pi_2 = \varepsilon$ it holds that $\varphi \vdash x\leq y$. The **initial state** and **reflexivity** prove all words including ν to be in in the language of $\mathcal{P}_r(\varphi \models^? x\leq y)$.
   (b) Let $\pi_2 = \varepsilon$ but $\pi_1 \neq \varepsilon$. Lemma 11 and **initial state** imply

$$\mathcal{P}_r(\varphi \models^? x\leq y) \vdash \ \gtrdot(x, y) \xrightarrow{\pi_1} (z, z).$$

   Since $\pi_1 \leq \nu$ **reflexivity** proves ν to be in in the language of $\mathcal{P}_r(\varphi \models^? x\leq y)$.
   (c) Let $\pi_1 \neq \varepsilon$ and also $\pi_1 \neq \varepsilon$. Then, there is also the assumption

$\nu \in \pi_1\, pr(\pi_2^*)$. Lemma 11 shows for some token s:

$$\mathcal{P}_r(\varphi \models^? x{\leq}y) \vdash\ \gtrdot(x,y) \xrightarrow{\pi_1} (z,s) \xrightarrow{\pi_2} (\_,z) \dashrightarrow (z,s)$$

Again, ν is contained in the language of $\mathcal{P}_r(\varphi \models^? x{\leq}y)$ according to the second P-edge rule.

(d) The remaining case is $\pi_1 = \varepsilon$, but $\pi_2 \neq \varepsilon$. Again, $\nu \in \pi_1\, pr(\pi_2^*)$. Since $\varphi \vdash x{\leq}z$ and $\varphi \vdash \pi_2(z){\leq}y$ it holds also $\varphi \vdash \pi_2(x){\leq}y$. Lemma 11 shows:

$$\mathcal{P}_r(\varphi \models^? x{\leq}y) \vdash\ \gtrdot(x,y) \xrightarrow{\pi_2} (\_,x) \dashrightarrow (x,y)$$

Again, ν is contained in the language of $\mathcal{P}_r(\varphi \models^? x{\leq}y)$.

We finally treat the case of finite trees. We reuse satisfiability for the infinite case rather then restarting from scratch. This requires another trick which is hidden in the proof of the next proposition.

Proposition 13 *Let ν be a word in $\{1,\ldots,ar_f\}^*$ that does not belong to the language of $\mathcal{P}_r(\varphi \models^? x{\leq}y)$ and φ be a constraint that is satisfiable over finite trees and contains variables $x, y \in V_\varphi$. Then the constraint $\varphi \wedge \mathrm{low}_{\nu(f)}(x) \wedge \mathrm{pref}^{\perp}_{\nu}(y)$ is also satisfiable in the structure of finite trees.*

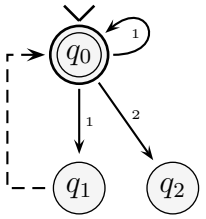
PROOF. In a first step, we express the formula $\mathrm{pref}^{\perp}_{\nu}(y)$ by a satisfaction the equivalent constraint $\mathrm{up}_{\nu(\perp)}(y)$:

$$\mathrm{up}_{\nu(\perp)}(y) \ =_{\mathrm{def}}\ y_\nu{=}\perp \wedge \bigwedge_{\varepsilon\leq\pi<\nu} y_\pi{\leq}f(y_{\pi 1},\ldots,y_{\pi n})$$

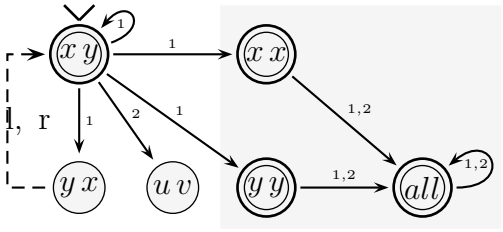
where $y_\varepsilon =_{\mathrm{def}} y$ and $F_\nu(y)$ is a set of fresh and distinct variables as before. Since $\varphi \wedge \mathrm{low}_{\nu(f)}(x) \wedge \mathrm{pref}^{\perp}_{\nu}(y)$ is satisfiable over possibly infinite trees by Proposition 12, we know that

$$\varphi' \quad =_{\mathrm{def}} \quad \varphi \wedge \mathrm{low}_{\nu(f)}(x) \wedge \mathrm{up}_{\nu(\perp)}(y)$$

is also satisfiable over possibly infinite trees. Now comes the trick: The constraint φ' cannot contain a label clash. (Otherwise it were unsatisfiable over possibly infinite trees by Proposition 4.) Furthermore, φ' cannot have a cycle clash, given that φ doesn't (by Proposition 7) and since the addition of $\mathrm{low}_{\nu(f)}(x) \wedge \mathrm{up}_{\nu(\perp)}(y)$ leaves this property invariant. Hence, Proposition 7 shows that φ' has a finite solution; and the satisfaction equivalent formula $\varphi \wedge \mathrm{low}_{\nu(f)}(x) \wedge \mathrm{pref}^{\perp}_{\nu}(y)$ has a finite solution, too.



an unshuffled cap automaton,



a suffered extension, and

corresponding entailment judgment

$$\left.\begin{array}{r} x{\leq}f(y,u) \wedge f(x,v){\leq}y \\ \wedge x{\leq}f(x,u) \wedge f(y,v){\leq}y \end{array}\right\} \models^{?} x{\leq}y$$

Fig. 5. An example for the shuffle property.

10 Restrictions of Constructed Automata

Constructed cap automata satisfy a set of restrictions that we must assume for the back translation in Section 12.

Definition 3 *We call a cap automaton $\mathcal{P}$ over A* restricted *if it is strictly epsilon free, gap universal, strictly cap, and shuffled.*

strictly epsilon free: *$\mathcal{P}$ has a unique initial state and no ε-transition.*
gap universal: *If a final state q_2 can be reached from a non-final state q_1 over some transition $\mathcal{P} \vdash q_1 \xrightarrow{i} \underline{\underline{q_2}}$ with $i \in A$ then q_2 is universal, i.e., for all $\pi \in A^*$ there exists a final state q_3 that can be reach over π from q_2: $\mathcal{P} \vdash \underline{\underline{q_2}} \xrightarrow{\pi} \underline{\underline{q_3}}$.*
strictly cap: *If $\mathcal{P} \vdash q_2 \xrightarrow{\pi} q_3 \dashrightarrow q_1$ with $\pi \neq \varepsilon$ then q_2 is a final state.*
shuffled: *If there are transitions $\mathcal{P} \vdash q \xleftarrow{\pi} q_0 \xrightarrow{\pi} q' \dashrightarrow q$ where $\mathcal{P} \vdash \gtrdot q_0$ is the initial state and $q \neq q'$ then the language $\{\pi' \mid \pi\pi' \in \mathcal{L}(\mathcal{P})\}$ is universal.*

We conjecture that these restrictions don't truly restrict the universality problem of cap automata but cannot prove this so far. Indeed, every cap automaton whose underlying finite automaton is deterministic can be made restricted. Again, this is not obvious. The proof exploits that "deterministic" cap automata are always shuffled. But unfortunately, the usual determination procedure fails for cap automata.

However, the shuffle property might be problematic, as illustrated by the example in Table 5. On the top, it presents a cap automaton over the alphabet $\{1, 2\}$ which violates the shuffle property at word 1. This automaton rejects all words in $1^*2(1{\cup}2)^*$. Below, a shuffled extension of this automaton is given; the additional nodes are marked in grey. The extended automaton recognizes

more words as it only rejects the words in $2(1 \cup 2)^*$ but is still not universal. A corresponding entailment judgment is also given: our automata construction applied to this judgment (for both left and right side) generates the shuffled extension of the original automaton.

The example shows that we cannot simply make an automaton shuffled without extending its language. There are also examples, where the shuffle extension of a non-universal automata becomes universal. So it remains open, whether assuming the shuffle property restricts the universality problem of cap automata or not.

Proposition 14 *Constructed cap automata $\mathcal{P}_\theta(\varphi \models^? x \leq y)$ are restricted.*

PROOF. Let $\mathcal{P}_l = \mathcal{P}_l(\varphi \models^? x \leq y)$ be a constructed cap automaton for the left side. $\mathcal{P}_l$ is clearly strictly epsilon free, as it has a unique initial state (x, y) and no ε-transitions. To see that it is gap universal, suppose that there is a transition from a non-final to a final state in $\mathcal{P}_l$. The second form of the **descend** rule is the only rule which may license such a transition. It thus has the form $\mathcal{P}_l \vdash (s, u) \xrightarrow{i} (\_, v)$ for some $u, v \in V_\varphi$, and $s \in V_\varphi \cup \{\_\}$. The only rule which can turn $(\_, v)$ into a final state is the **top** rule, but this rule turns $(\_, v)$ directly into a universal state. (The **final states** rule does not apply because of the underscore on the left.)

To prove that $\mathcal{P}_l$ is shuffled, we assume a path π and two different states q and q' with $\mathcal{P}_l \vdash q \xleftarrow{\pi} q_0 \xrightarrow{\pi} q' \dashrightarrow q$. We unify the states q_0, q, q' with the rules of Table 3 and get $\mathcal{P}_l \vdash (v, u) \xleftarrow{\pi} (x, y) \xrightarrow{\pi} (u, s) \dashrightarrow (v, u)$ for some $u, v \in V_\varphi \cup \{x, y\}$ and $s \in V_\varphi \cup \{x, y, \_\}$. By construction of the automaton (Table 3), $\mathcal{P}_l \vdash (x, y) \xrightarrow{\pi} (u, u)$ must also hold. By **reflexivity** and **all**, the language $\{\pi' \mid \pi\pi' \in \mathcal{L}(\mathcal{P}_l)\}$ is universal.

We finally prove the strict cap property. All P-edges of $\mathcal{P}_l$ are of the form $\mathcal{P} \vdash (u, s) \dashrightarrow q$ for some state q. The last transition in all transition sequences reaching (u, s) must be licensed by the **descend** rule, and thus is of the form $\mathcal{P} \vdash (v, s_0) \xrightarrow{i} (u, s)$. Now, the **final states** rule applies to (v, s_0). Repeating this argument inductively shows that all states leading to (v, s_0) are final too.

11 Restricted Cap Set Expressions

We now formulate corresponding restrictions for cap set expressions. Thereby, we obtain the restrictions needed for Theorem 1 to hold.

Definition 4 *We call a cap set expression over alphabet A* restricted *if it is*

a shuffled expression with the following abstract syntax where R_1, R_2, R range over regular expressions over A:

$$F ::= pr(R_1 R_2^\circ) \mid \ RA^* \mid \ F_1 \cup F_2$$

A cap set expression of sort F is called shuffled *if all its components of the form $pr(R_1 R_2^\circ)$ with $\mathcal{L}(R_2) \neq \{\varepsilon\}$ satisfy:*

shuffle: *for all words $\pi \in \mathcal{L}(R_1) \cap \mathcal{L}(R_1 R_2)$ it holds that $\pi A^* \subseteq \mathcal{L}(R_1)$.*

Proposition 15 *Universality of restricted cap set expressions and restricted cap automata are equivalent modulo deterministic polynomial time transformations.*

PROOF. It is easy to see that those cap automata are gap-universal, strictly cap, and shuffled that the proof of Proposition 2 constructs for restricted cap expressions. They can be made strictly ε-free in addition, as we will see in Lemma 20.

Conversely, given a restricted cap automaton $\mathcal{P}$, we can express the regular part of $\mathcal{P}$ by a restricted cap set expression $pr(R_1\varepsilon^\circ) \cup R_2 A^*$, because of $\mathcal{P}$ is gap universal. The cap automaton $\mathcal{P}$ is also strictly cap, so we can translate every P-edge of $\mathcal{P}$ in a restricted cap set expression $pr(R_1 R_2^\circ)$. All build restricted cap set expressions are shuffled since $\mathcal{P}$ is shuffled.

In order to complete the preceeding proof, we must show how to make cap automata strictly ε-free. We call a state q of a cap automaton $\mathcal{P}$ *normalized* if q has no in-going transitions and no out-going P-edges.

Lemma 19 *If a cap automaton $\mathcal{P}$ has a unique initial state then this state can be normalized, while preserving the language of the automaton, gap-universality, strict cap, and the shuffle property.*

PROOF. Let $\mathcal{P}$ be a cap automaton with one initial state q_0. We construct a new automaton $\mathcal{P}'$ by adding a state q_0' to $\mathcal{P}$ which inherits all out-going Δ-transitions and in-going P-edges from q_0. We let q_0' be the unique initial state of $\mathcal{P}'$. This state is normalized.

Lemma 20 (Epsilon elimination) *Every cap automaton can be made strictly ε-free in polynomial time, while preserving the language and the properties: gap-universal, strictly cap, and shuffle.*

left	$l(q) \leq f(l(q_1),\ldots,l(q_n))$ in $\varphi_{\mathcal{P}}$	if $\mathcal{P} \vdash \underline{\underline{q}} \xrightarrow{i} q_i$ for all $1 \leq i \leq n$.
right	$f(r(q_1),\ldots,r(q_n)) \leq r(q)$ in $\varphi_{\mathcal{P}}$	if $\mathcal{P} \vdash q \xrightarrow{i} q_i$ for all $1 \leq i \leq n$
top	$r(q') = \top$ in $\varphi_{\mathcal{P}}$	if $\mathcal{P} \vdash q \xrightarrow{i} \underline{\underline{q'}}$, q not final
P-edges	$l(q) \leq i[\, r(q_2)\,]$ in $\varphi_{\mathcal{P}}$	if $\mathcal{P} \vdash \underline{\underline{q}} \xrightarrow{i} q_1 \dashrightarrow q_2,\ q_1 \neq q_2$

Table 7
Back translation: the constraint $\varphi_{\mathcal{P}}$ of a restricted cap automaton $\mathcal{P}$.

PROOF. First, we eliminate ε-edges in the underlying finite automaton. This yields a cap automaton which may have more than one initial state. We assume w.l.o.g that this automaton consists of n independent parts where each part has exactly one initial state. Second, we normalize all n initial states according to Lemma 19.

We prove that the ε-elimination does not affect the language. Let $\mathcal{P}_{\not\varepsilon}$ be the cap automaton that results after ε-elimination in a cap automaton $\mathcal{P}_{\varepsilon}$. By induction it holds that $\mathcal{P}_{\varepsilon} \vdash q_1 \xrightarrow{\pi} q_2$ if and only if $\mathcal{P}_{\not\varepsilon} \vdash q_1 \xrightarrow{\pi} q_2$ where $\pi \neq \varepsilon$. Further it holds $\mathcal{P}_{\varepsilon} \vdash \succ q_0 \xrightarrow{\varepsilon} q_0'$ if and only if $\mathcal{P}_{\not\varepsilon} \vdash \succ q_0'$. This implies

$$\mathcal{P}_{\varepsilon} \vdash \succ q_0 \xrightarrow{\pi} \underline{\underline{q_1}} \xrightarrow{\mu} q_2 \dashrightarrow \underline{\underline{q_1}} \quad \text{if and only if} \quad \mathcal{P}_{\not\varepsilon} \vdash \succ q_0' \xrightarrow{\pi} \underline{\underline{q_1}} \xrightarrow{\mu} q_2 \dashrightarrow \underline{\underline{q_1}}.$$

So $\mathcal{L}(\mathcal{P}_{\varepsilon})$ and $\mathcal{L}(\mathcal{P}_{\not\varepsilon})$ are equal.

Second, we copy the whole cap automaton n-times where n is the number of the initial states. The result is a big cap automaton which consists of n independent parts, each has a single initial state.

Third, we unify all n initial states into a single initial state. The unified initial state inherits all P- and Δ-edges of the unified initial states. It is final if and only if one of the previous initial states was. Since all initial states are normalized this step does neither change the language of $\mathcal{P}$, nor gap-universal, the strict cap nor the shuffle property.

12 Back Translation for Restricted Cap Automata

We now encode universality of restricted cap automata over alphabet $\{1,\ldots,n\}$ back to NSSE over the signature $\{\bot, f, \top\}$ where $ar_f = n$. Again, our construction applies to finite, regular, and possibly infinite trees.

Definition 5 *Given a restricted cap automaton we assume two fresh variables $l(q)$ and $r(q)$ for each state $\mathcal{P} \vdash q$. The judgment $J(\mathcal{P})$ of a restricted cap*

$x{\leq}f(y) \models^? x{\leq}y$ $\quad$ $l(p){\leq}f(l(q)) \wedge f(r(q)){\leq}r(p) \wedge l(p){\leq}f(r(p)) \models^? l(p){\leq}r(p)$

Fig. 6. A judgment, its pair of cap automata, and the back translation of the left cap automaton.

automaton $\mathcal{P}$ with initial state $\mathcal{P} \vdash \rightarrow q_0$ is $\varphi_{\mathcal{P}} \models^? l(q_0){\leq}r(q_0)$ where $\varphi_{\mathcal{P}}$ is the least constraint with the properties in Table 7.

The judgment $J(\mathcal{P})$ is defined such that $\mathcal{P}$ recognizes exactly the set of l-safe words for $J(\mathcal{P})$ whereas the set of r-safe words for $J(\mathcal{P})$ is A^*.

Proposition 16 (Correctness) *Every complete and restricted cap automaton $\mathcal{P}$ with initial state $\mathcal{P} \vdash \rightarrow q_0$ over alphabet A satisfies:*

$$\mathcal{L}(\mathcal{P}) \;=\; \mathcal{L}(\mathcal{P}_l(J(\mathcal{P})) \quad \textit{and} \quad A^* \;=\; \mathcal{L}(\mathcal{P}_r(J(\mathcal{P})).$$

For proof we need an auxiliary lemma.

Lemma 21 *If $J(\mathcal{P}) \vdash u{\leq}v$ then $u = v$.*

PROOF. By structural induction on derivations $J(\mathcal{P}) \vdash u{\leq}v$. The derivation rules are given in Table 1. The consideration for the rules **reflexive** and **trans.** are obvious. So, we can restrict ourself to the **decomp.** rule. So assume that the decomposition rule derives $J(\mathcal{P}) \vdash u{\leq}v$. Hence, $f(\ldots){\leq}u$ and $v{\leq}f(\ldots)$ in $J(\mathcal{P})$ while $J(\mathcal{P}) \vdash u{\leq}v$. The induction hypothesis yields $u = v$. The construction of $J(\mathcal{P})$ in Table 7 implies that $u = r(q)$ and $v = l(q)$ for some states p, q. Hence $r(q) = l(q)$ which is impossible so that the **decomp.** rule cannot be applicable.

Proof of Proposition 16

(1) The language $\mathcal{L}(\mathcal{P}_r(J(\mathcal{P})))$ is universal: Since $\mathcal{P}$ is complete such that the **right** rule implies for all words $\pi \in A^*$ that there exists a state $\mathcal{P} \vdash q$ satisfying $\varphi_{\mathcal{P}} \models \pi(r(q)){\leq}r(q_0)$. Thus, $\mathcal{P}_r(J(\mathcal{P})) \vdash \underline{\underline{(l(q_0), r(q_0))}} \xrightarrow{\pi} \underline{\underline{(\,\_\,, l(q))}}$ by the second case of the **descend** rule, i.e. π is accepted by $\overline{\overline{\mathcal{P}_r(J(\mathcal{P}))}}$.

(2) We omit the proof for $\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\mathcal{P}_l(J(\mathcal{P})))$ which only requires the completeness of $\mathcal{P}$ and the strictly cap property.

(3) The remaining inclusion $\mathcal{L}(\mathcal{P}_l(J(\mathcal{P}))) \subseteq \mathcal{L}(\mathcal{P})$ is most interesting. Note that according to Lemma 21 we have not to consider any support $J(\mathcal{P}) \vdash u{\leq}v$ in the construction of the appendant cap automata $\mathcal{P}_l(J(\mathcal{P}))$. We start with an auxiliary **claim**: If $\mathcal{P}$ provides transitions

$$\mathcal{P}_l(J(\mathcal{P}) \vdash (l(q_0), s_0) \xrightarrow{\pi} (l(q_n), s_n) \xrightarrow{i} (l(q), s)$$

then there exist transitions $\mathcal{P} \vdash \underline{\underline{q_0}} \xrightarrow{\pi} \underline{\underline{q_n}}$. This claim can be proved as follows: All transitions must be licensed by a constraint in $\varphi_{\mathcal{P}}$ which is of the form $l(q_i){\leq}f(\ldots, l(q_{i+1}), \ldots)$ where $1 \leq i \leq n$. Such constraints can only be created by the **left** rule. There thus exist transitions $\mathcal{P} \vdash \underline{\underline{q_0}} \xrightarrow{\pi} q_n$ such $\mathcal{P} \vdash \underline{\underline{q_i}}$ for all $0 \leq i < n$. We can infer $\mathcal{P} \vdash \underline{\underline{q_n}}$ as required.

We now come back to the main proof. Suppose $\pi \in \mathcal{L}(\mathcal{P}_l(J(\mathcal{P})))$. There are three kinds of transitions by which π can be recognized.

(a) We first consider transitions using the **reflexivity** rule to recognize π. These contain a transition sequence of the following form for some prefix $\pi' \leq \pi$:

$$\mathcal{P}_l(J(\mathcal{P})) \vdash (l(q_0), r(q_0)) \xrightarrow{\pi'} \underline{\underline{(r(q_n), r(q_n))}}$$

Either $\pi' = \pi$ or this sequence can be continued to recognize π in the state $\underline{\underline{all}}$. The first continuation step is by the **reflexivity** rule itself and all subsequent steps are due to the **all** rule.

Note that $n \geq 1$. We first consider the descendants on the left hand side, which starts from state $l(r_0)$ and continues over $l(l_{n-1})$ to $r(q_m)$. The last step must be induced by a constraint in $\varphi_{\mathcal{P}}$ that is contributed by the **P-edges** rule. This and the preceding claim yield the existence of the following transitions for some state $q \neq q_n$:

$$\mathcal{P} \vdash q_0 \xrightarrow{\pi'} q \dashrightarrow q_n$$

We next consider the descendants on the right hand side. They must be induced by constraints in $\varphi_{\mathcal{P}}$ that are inherited form the following transition sequence:

$$\mathcal{P} \vdash q_0 \xrightarrow{\pi'} q_n$$

Now we can apply that $\mathcal{P}$ is shuffled which shows that the language $\{\pi'' \mid \pi'\pi'' \in \mathcal{L}(\mathcal{P})\}$ is universal (since $q \neq q_n$)). Thus, $\pi \in \mathcal{L}(\mathcal{P})$ as required.

(b) Second, we consider transitions using the **top** rule. These contain a part of the following form for some prefix $\pi' \leq \pi$ and such that $r(q_n){=}\top$ in $\varphi_{\mathcal{P}}$.

$$\mathcal{P}_l(J(\mathcal{P})) \vdash (l(q_0), r(q_0)) \xrightarrow{\pi'} \underline{\underline{(s_n, r(q_n))}}$$

Again, either $\pi' = \pi$ or this sequence can be continued to recognize π in the state $\underline{\underline{all}}$. The first continuation step is by the **top** rule itself and all subsequent steps are due to the **all** rule.

The above transitions of $\mathcal{P}_l(J(\mathcal{P}))$ are induced by the following transition sequence in $\mathcal{P}$ where q_{n-1} is not final:

$$\mathcal{P} \vdash q_0 \xrightarrow{\pi'} \underline{\underline{q_n}}$$

The gap universal property which holds for $\mathcal{P}$ by assumption yields that $\{\pi'' \mid \mathcal{P} \vdash q_n \xrightarrow{\pi''} \underline{\underline{q_n}}\}$ is universal. Thus, $\pi \in \mathcal{L}(\mathcal{P})$.

(c) Third, we consider the last case where the class of π is A in $\mathcal{P}_l(J(\mathcal{P}))$. The recognizing transition has to apply the rule for **final states**:

$$\mathcal{P}_l(J(\mathcal{P})) \vdash \underline{\underline{(l(q_0), r(q_0))}} \xrightarrow{\pi} \underline{\underline{(l(q_n), s_n))}} \xrightarrow{i} (\theta(q), p(\pi))$$

All transitions except the last one must be contributed by the **left** rule. The **P-edges** can only apply at the end. In this case however, we can freely exchange the last transition by another using the **left** rule as well. Given this, we can apply our initial claim which yields:

$$\mathcal{P} \vdash \underline{\underline{q_0}} \xrightarrow{\pi} \underline{\underline{q_n}}$$

Thus, we have shown that $\pi \in \mathcal{L}(\mathcal{P})$ for this case too.

(d) Finally, we have to consider transitions that recognize π through P-edges of $\mathcal{P}_l(J(\mathcal{P}))$. Here we have transitions where π is a prefix of $\pi_1\pi_2^k$ for some $k \geq 0$: $\mathcal{P}_l(J(\mathcal{P})) \vdash$

$$(l(q_0), r(q_0)) \xrightarrow{\pi_1} (l(q_i), r(q_i)) \xrightarrow{\pi_2} (r(q_n), s_n) \dashrightarrow (l(q_i), r(q_i))$$

The **P-edges** rule in the construction of $\mathcal{P}_l$ requires $q_n = q_i$. The automaton $\mathcal{P}$ thus has the following transitions for some state q:

$$\mathcal{P} \vdash q_0 \xrightarrow{\pi_1} q_i \xrightarrow{\pi_2} q \dashrightarrow q_i$$

This transition and the strictly cap property allows $\mathcal{P}$ to recognize all prefixes of $\pi_1\pi_2^k$ for all $k \geq 0$, i.e. $\pi \in \mathcal{L}(\mathcal{P})$.

For illustration, we reconstruct an entailment judgment for $\mathcal{P}_l(x{\leq}f(y) \models^? x{\leq}y)$ given in Table 6. Before we start we rename the states of $\mathcal{P}_l(x{\leq}f(y) \models^? x{\leq}y)$ to p and q. We translate the edge $\underline{\underline{p}} \xrightarrow{1} q$ to the constraint $l(p){\leq}f(l(q)) \wedge f(r(q)){\leq}r(p)$ (rule **left** and **right** of Table 7). The rule **P-edges** maps the P-edge $q \dashrightarrow p$ to the constraint $l(p){\leq}f(r(p))$. If we now construct the left automaton of the computed constraint, we get the original automaton back.

Lemma 22 *Let $\mathcal{P}$ be a restricted cap automaton with initial state $\mathcal{P} \vdash \ \rightarrow q$. The constructed constraint $\varphi_{\mathcal{P}}$ is satisfiable over finite and infinite trees.*

Theorem 4 (Back translation) *Universality of restricted cap automata over the alphabet $\{1, \ldots, ar_f\}$ can be reduced in polynomial time to NSSE with signature $\{\bot, f, \top\}$ (respectively over finite, regular, or possibly infinite trees).*

PROOF. Let $\mathcal{P}$ be complete and restricted cap automaton. Universality of $\mathcal{L}(\mathcal{P})$ is equivalent to universality of both languages: $\mathcal{L}(\mathcal{P}_l(J(\mathcal{P})))$ and $\mathcal{L}(\mathcal{P}_r(J(\mathcal{P})))$ (Proposition 16). Since $\varphi_{\mathcal{P}}$ is clash-free (Lemma 22), the latter is equivalent to that NSSE holds for the judgment $J(\mathcal{P})$ (Theorem 2).

13 Equivalence of Variants of NSSE

We prove Corollary 1 which states that all variants of NSSE over the signature $\{\bot, f, \top\}$ are equivalent if the arity of f is at least 2. Given the characterization of NSSE in Theorem 1 it remains to prove a corresponding result for restricted cap automata:

Proposition 17 *The universality problems of restricted cap automata over the alphabet $\{1, \ldots, n\}$ are equivalent for all $n \geq 2$ modulo polynomial time transformations.*

PROOF. We first show how to extend to alphabet. Consider a restricted cap automaton $\mathcal{P}$ and an alphabet $A = \{1, \ldots, n{-}1\}$. We construct another restricted cap automaton $\mathcal{P}'$ with an alphabet $A' = \{1, \ldots, n\}$ in linear time. The cap automaton $\mathcal{P}'$ is identical to $\mathcal{P}$ up to the additions

$$\mathcal{P}' \vdash q_0 \xrightarrow{n} \underline{\underline{q_2}} \xrightarrow{1,\ldots,n} \underline{\underline{q_2}} \qquad \mathcal{P}' \vdash q_0 \xrightarrow{1,\ldots,n} q_1 \xrightarrow{1,\ldots,n} q_1 \xrightarrow{n} \underline{\underline{q_2}} \xrightarrow{1,\ldots,n} \underline{\underline{q_2}}$$

where q_0 is the initial state of $\mathcal{P}$ and $\mathcal{P}'$ and q_1, q_2 are two fresh states. This construction composes:

$$\mathcal{L}(\mathcal{P}') = \{\ \pi\sigma \mid \pi \in \mathcal{L}(\mathcal{P}), \ \text{ and } \sigma \in n(1, \ldots, n)^*\ \}.$$

We now consider alphabet restriction. Let $\mathcal{P}$ be a restricted cap automaton with alphabet $A = \{1, \ldots, n^2\}$ where $n^2 \geq 3$. We can assume w.l.o.g that A is of that form. Otherwise we can increase A by the previous construction until this form is reached.

We next construct a restricted cap automaton $\mathcal{P}'$ with alphabet $A' = \{1, \ldots, n\}$ in polynomial time such that $\mathcal{L}(\mathcal{P})$ is universal if and only if $\mathcal{L}(\mathcal{P}')$ is universal. We encode a letter of A in two letters of A' to the base n via the standard encoding $d : A \to A' \times A'$:

$$d(i) = (\, d_1(i)\,,\, d_2(i)\,) = \left(\, \left\lfloor \frac{i}{n} \right\rfloor \,,\, i \bmod n \,\right).$$

The cap automaton $\mathcal{P}'$ has two states q and q' for every state q of $\mathcal{P}$. The states q and q' are final in $\mathcal{P}'$ if q is final in $\mathcal{P}$, i.e.

$$\mathcal{P}' \vdash \underline{\underline{q_1}} \text{ and } \mathcal{P}' \vdash \underline{\underline{q_1'}} \quad \text{if} \quad \mathcal{P} \vdash \underline{\underline{q_1}}.$$

The cap automaton $\mathcal{P}$ and $\mathcal{P}'$ share the same initial state and the same P-edges. We define the transitions of $\mathcal{P}'$ by

$$\mathcal{P}' \vdash q_1 \xrightarrow{d_1(i)} q_1' \xrightarrow{d_2(i)} q_2 \quad \text{if} \quad \mathcal{P} \vdash q_1 \xrightarrow{i} q_2.$$

We can show by induction that the word $i_1 \ldots i_m$ is in $\mathcal{L}(\mathcal{P})$ if and only if the words $d_1(1)\, d_2(1) \ldots d_1(m-1)\, d_2(m-1)\, d_1(m)$ and $d_1(1)\, d_2(1) \ldots d_1(m)\, d_2(m)$ are in $\mathcal{L}(\mathcal{P}')$.

Conclusion and Future Work

We have characterized NSSE equivalently by using regular expressions and word equations. This explains why NSSE is so difficult to solve and links NSSE to the area of string unification where powerful proof methods are available. Given that NSSE is equivalent to universality of restricted cap set expressions, one cannot expect to solve NSSE without treating word equations.

We have also shown that all variants of NSSE with a single function symbol of arity at least two are equivalent modulo polynomial time transformations. One might also want to extend the presented characterization to richer signatures. For instance, it should be possible to treat NSSE with a contra-variant function symbol. But how to deal with more than one non-constant function symbol is much less obvious.

Another open question is whether there exists a direct relation between cap automata and tuple tree automata with equality tests, which are used in the alternative approach to subtype entailment in [21].

Acknowledgements

We would like to thank Zhendong Su, Klaus Schulz, Jean-Marc Talbot, Sophie Tison, and Ralf Treinen for discussions and comments on early versions.

References

[1] Y. Fuh, P. Mishra, Type inference with subtypes, Theoretical Computer Science 73 (2) (1990) 155–175.

[2] J. C. Mitchell, Type inference with simple subtypes, The Journal of Functional Programming 1 (3) (1991) 245–285.

[3] R. M. Amadio, L. Cardelli, Subtyping recursive types, ACM Transactions on Programming Languages and Systems 15 (4) (1993) 575–631.

[4] J. Eifrig, S. Smith, V. Trifonov, Sound polymorphic type inference for objects, in: ACM Conference on Object-Oriented Programming: Systems, Languages, and Applications, ACM Press, 1995, pp. 169–184.

[5] F. Pottier, Simplifying subtyping constraints, in: ACM SIGPLAN International Conference on Functional Programming, ACM Press, 1996, pp. 122–133.

[6] F. Pottier, Type inference in the presence of subtyping: from theory to practice, Ph.D. thesis, Institut de Recherche d'Informatique et d'Automatique (1998).

[7] D. Kozen, J. Palsberg, M. I. Schwartzbach, Efficient inference of partial types, Journal of Computer and System Sciences 49 (2) (1994) 306–324.

[8] J. Palsberg, M. Wand, P. O'Keefe, Type Inference with Non-structural Subtyping, Formal Aspects of Computing 9 (1997) 49–67.

[9] D. Kozen, J. Palsberg, M. I. Schwartzbach, Efficient recursive subtyping, Mathematical Structures in Computer Science 5 (1995) 1–13.

[10] J. Eifrig, S. Smith, V. Trifonov, Type inference for recursively constrained types and its application to object-oriented programming, Electronic Notes in Theoretical Computer Science 1.

[11] J. Rehof, Minimal typings in atomic subtyping, in: ACM Symposium on Principles of Programming Languages, ACM Press, 1997, pp. 278–291.

[12] F. Pottier, A framework for type inference with subtyping, in: ACM SIGPLAN International Conference on Functional Programming, ACM Press, 1998, pp. 228–238.

[13] J. Rehof, The complexity of simple subtyping systems, Ph.D. thesis, DIKU, Copenhagen (1998).

[14] F. Henglein, J. Rehof, The complexity of subtype entailment for simple types, in: IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, 1997, pp. 362–372.

[15] F. Henglein, J. Rehof, Constraint automata and the complexity of recursive subtype entailment, in: International Colloquium on Automata, Languages, & Programming, Lecture Notes in Computer Science 1443, Springer-Verlag, 1998, pp. 616–627.

[16] J. Niehren, T. Priesnitz, Entailment of non-structural subtype constraints, in: Asian Computing Science Conference, Lecture Notes in Computer Science 1742, Springer-Verlag, 1999, pp. 251–265.

[17] G. Makanin, The problem of solvability of equations in a free semigroup, Mathematics of the USSR Sbornik 32, translated from Russian.

[18] W. Plandowski, Satisfiability of word equations with constants is in PSPACE, in: IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1999, pp. 495–500.

[19] M. Müller, J. Niehren, R. Treinen, The first-order theory of ordering constraints over feature trees, in: IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, 1998, pp. 432–443.

[20] J. Niehren, M. Müller, J.-M. Talbot, Entailment of atomic set constraints is PSPACE-complete, in: IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, 1999, pp. 285–294.

[21] Z. Su, A. Aiken, J. Niehren, T. Priesnitz, R. Treinen, First-order theory of subtyping constraints, in: ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press, 2002, pp. 203–216.

[22] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree automata techniques and applications, Available at `http://www.grappa.univ-lille3.fr/tata` (Oct. 1999).

[23] R. Treinen, Predicate logic and tree automata with tests, in: Foundations of Software and Computer Structures, Lecture Notes in Computer Science 1784, Springer-Verlag, 2000, pp. 329–343.

[24] J. Niehren, T. Priesnitz, Non-structural subtype entailment in automata theory, in: Fourth International Symposium on Theoretical Aspects of Computer Software, Lecture Notes in Computer Science 2215, Springer-Verlag, 2001, pp. 360–384.

[25] K. U. Schulz, Makanin's algorithm for word equations – two improvements and a generalization., in: Word Equations and Related Topics, Lecture Notes in Computer Science 572, Springer-Verlag, 1991, pp. 85–150.

[26] V. Durnev, Unsolvability of positive $\forall\exists^3$-theory of free groups, in: Sibirsky mathematichesky jurnal, Vol. 36(5), 1995, pp. 1067–1080, in Russian, also exists in English translation.

[27] F. Baader, K. Schulz, Unification in the union of disjoint equational theories: Combining decision procedures, Journal of Symbolic Computation 21 (1996) 211–243.

[28] Y. Vazhenin, B. Rozenblat, Decidability of the positive theory of a free countably generated semigroup, in: Mathematics of the USSR Sbornik, Vol. 44, 1983, pp. 109–116.

[29] J. Palsberg, P. O'Keefe, A Type System Equivalent to Flow Analysis, ACM Transactions on Programming Languages and Systems 17 (4) (1995) 576–599.