HAL
open science

Non-Structural Subtype Entailment in Automata Theory

Joachim Niehren, Tim Priesnitz

To cite this version:

Joachim Niehren, Tim Priesnitz. Non-Structural Subtype Entailment in Automata Theory. 4th International Symposium on Theoretical Aspects of Computer Software, 2001, Sendai, Japan. pp.360–384. inria-00536514

HAL Id: inria-00536514

https://inria.hal.science/inria-00536514

Submitted on 16 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Non-Structural Subtype Entailment in Automata Theory

Joachim Niehren Tim Priesnitz

Programming Systems Lab, Universität des Saarlandes, Saarbrücken, Germany
www.ps.uni-sb.de/~{niehren,tim}

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.

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 [6, 12, 1, 4, 18, 20]. *Subtype satisfiability* can be checked in cubic time for many type languages [9, 16]. A quadratic time algorithm for the variable free case is presented in [10].

Interest in *subtype entailment* was first raised by practical questions on type inference engines with subtyping [5, 21, 19]. 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 [22] 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 [7] for finite trees (simple types) and PSPACE-complete [8] 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 [8]. 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 [15] (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 satisfiablity but not for entailment [9, 16]. Our extension to entailment yields a new characterization of NSSE that uses regular expressions and word equations [11, 17]. 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 [13, 14].

2 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.

A *word* over an alphabet $\mathbf{A}$ is a finite sequence of letters in $\mathbf{A}$. We denote words by π, μ, or ν and the set of words over $\mathbf{A}$ with $\mathbf{A}^*$. 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 define *regular expressions* R over alphabet $\mathbf{A}$ as usual:

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

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

$$E ::= R_1 R_2^\circ \mid \ E_1 \cup E_2$$

Cap set expressions E denote sets of words $\mathcal{L}(E) \subseteq \mathbf{A}^*$ 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 $S^\circ \subseteq \mathbf{A}^*$ for all sets $S \subseteq \mathbf{A}^*$. Let *pr* be the prefix operator lifted to sets of words. We set:

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

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. 27) such that 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 and for possibly infinite trees.*

The theorem allows to derive the following robustness result of NSSE against variations from automata transformations (Sec. 12).

Corollary 2. *All variants of NSSE with signature $\{\bot, f, \top\}$ where the arity of f at least $n \geq 2$ are equivalent. It does not even matter whether finite 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 [23]. The reduction can easily be based on the following lemma that is well known in the field of string unification.

Lemma 3. *For all words $\pi \in \mathbf{A}^*$ and nonempty words $\mu \in \mathbf{A}^+$ it holds that $\pi \in pr(\mu^*)$ if and only if $\pi \in pr(\mu\pi)$. Thus, all sets $S \subseteq \mathbf{A}^+$ of nonempty words satisfy:*

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

We can thus 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 4. *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 $\{1, \dots, n\}$.*

Unfortunately, even the positive $\forall\exists^3$ fragment of a single word equation is undecidable [3] except if the alphabet is infinite [2] or a singleton [24]. 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 3.

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 4.

Corollary 5. *NSSE is decidable for the signature $\{\bot, f, \top\}$ if f is unary.*

Plan of the Paper. We first recall the precise definition of NSSE (Sec. 3) and then prove Theorem 1 in 5 subsequent steps. This covers most of the paper (Sec. 4 – 11).

First, we express NSSE by a so called *safety* property for sets of words (Sec. 4). Second, we introduce *cap-automata* – a restricted version of P-automata as introduced [15] – which can recognize exactly the same languages as cap set expressions (Sec. 5). Third, we show how to construct cap automata corresponding to entailment judgments. This construction encodes NSSE into universality of cap automata (Sec. 6). We prove the soundness (Sec. 7) and completeness (Sec. 8) of our construction. Fourth, we infer restrictions that are satisfied by all constructed cap automata and define corresponding restrictions for cap set expressions (Sec. 9 and 10). Fifth, we give a back translation (Sec. 11) that reduces universality of restricted cap automata into NSSE.

Finally, we present transformations on restricted cap automata that allow us to derive Corollary 2 from Theorem 1 (Sec. 12), and conclude.

3 Non-Structural Subtype Constraints

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 $\mathsf{ar}_\bot = \mathsf{ar}_\top = 0$ and $\mathsf{ar}_f \geq 1$.

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

3.1 Non-Structural Subtyping

We next define finite and infinite trees over Σ. We consider trees as partial functions $\tau : \mathbb{N}^* \rightsquigarrow \Sigma$ which map words over natural numbers to function symbols. A tree τ is *finite* if its domain D_τ is finite and otherwise *infinite*. The words in $D_\tau \subseteq \mathbb{N}^*$ are called the *nodes* or *paths* of the tree. The idea is to identify a node with the path that addresses it relative to the root. 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 $1 \leq i \leq \mathsf{ar}_{\tau(\pi)}$.

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 $\tau_1, \ldots, \tau_{\mathsf{ar}_g}$ we define $\tau = g(\tau_1, \ldots, \tau_{\mathsf{ar}_g})$ by $\tau(\varepsilon) =_\Sigma g$ and $\tau(i\pi) =_\Sigma \tau_i(\pi)$ for all $\pi \in D_{\tau_i}$ and $1 \leq i \leq \mathsf{ar}_g$. 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 tree $\varepsilon \mapsto a$. But this should never lead to confusion.

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 partial order

on trees which satisfies for all trees τ_1, τ_2 over Σ:

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

3.2 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:

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

where $n = \mathsf{ar}_f$. 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 for 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 6. 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 7. The prototypical example where NSSE holds somehow surprisingly is:

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

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 $\alpha(y){<}\alpha(x)$.

4 Entailment via Safety

We now 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 [15]. Let $\varphi \models^{?} x{\leq}y$ be an entailment judgment and π a word in $\{1, \ldots, \mathsf{ar}_f\}^*$. 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 [15] 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 refine the notion of safety into two dual notions: *left (l) safety* and *right (r) safety*.

$$\pi \text{ is } \textit{l-safe} \text{ for } \varphi \models^{?} x{\leq}y \;\text{ iff }\; \varphi \models \begin{cases} & x(\pi) =_{\Sigma} f \\ \vee \bigvee_{\pi' \leq \pi} & x(\pi') =_{\Sigma} \bot \\ \vee \bigvee_{\pi' \leq \pi} & y(\pi') =_{\Sigma} \top \end{cases}$$

This means that $\varphi \models^{?} x{\leq}y$ cannot be contradicted by a solution α of φ that maps the left hand side x at node π to $\top$, i.e. where $\alpha(x)(\pi) = \top$. The notion of r-safety is analogous, expect that one tries to contradict at the right hand side with $\bot$.

$$\pi \text{ is } \textit{r-safe} \text{ for } \varphi \models^{?} x{\leq}y \;\text{ iff }\; \varphi \models \begin{cases} & y(\pi) =_{\Sigma} f \\ \vee \bigvee_{\pi' \leq \pi} & x(\pi') =_{\Sigma} \bot \\ \vee \bigvee_{\pi' \leq \pi} & y(\pi') =_{\Sigma} \top \end{cases}$$

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.

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 8. *Entailment $\varphi \models x{\leq}y$ holds if and only if all words $\pi \in \{1, \ldots, \mathsf{ar}_f\}^*$ 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 $\alpha(x)(\pi) \in \{f, \top\}$ and π fails to be r-safe.
2. Otherwise $\alpha(y)(\pi) =_\Sigma f$. Thus $\alpha(x)(\pi) =_\Sigma \top$ which contradicts that π is 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 π satisfies the second condition of both l-safety and r-safety 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 $\alpha(y)(\pi') =_\Sigma \top$. Thus, π satisfies the third condition of l-safety and r-safety 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 $\mathsf{ar}_f \geq 1$ yields $\pi = \pi'$. Now, π satisfies the first conditions of l-safety and r-safety for $\alpha \models^? x{\leq}y$.

Example 9. The surprising effect of Example 7 seems to go away if one replaces the unary function symbol there by a binary function symbol:

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

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 10. 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 7. How to understand this effect in general will be explained in Section 6.

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

5 Cap Automata and Cap Sets

We now restrict the class of P-automata introduced in [15] to the class of so called *cap automata*[1]. We then show that the class of languages recognized by cap automata is precisely the class of cap sets, i.e. those sets of words described by cap set expressions.

[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 the corresponding language definitions only. 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, since it is satisfied by all those constructed in the proof of Proposition 12. Thus, cap automata are indeed properly subsumed by the P-automata.

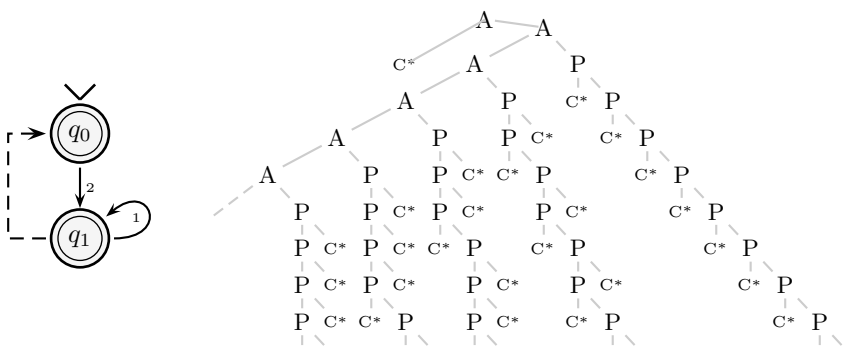


Fig. 1. A cap automaton with a non context-free language $(21^*)^\circ$.

A *finite automaton* $\mathcal{A}$ over alphabet $\mathbf{A}$ 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 $\Delta \subseteq Q \times (\mathbf{A} \cup \{\varepsilon\}) \times Q$ 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 $\mathcal{A} \vdash {>}q$ 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 $\mathcal{A}$ complete if for every word $\pi \in \mathbf{A}^*$ there exists states q_0 and q_1 such that $\mathcal{A} \vdash {>}q_0 \xrightarrow{\pi} q_1$.

Definition 11. A *cap automaton* $\mathcal{P}$ over alphabet $\mathbf{A}$ consists of a finite automaton $\mathcal{A}$ over $\mathbf{A}$ 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 $\mathbf{A}$ recognizes the following language $\mathcal{L}(\mathcal{P}) \subseteq \mathbf{A}^*$:

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

The first set is the language of the finite automaton underlying $\mathcal{P}$. The second set add 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_2 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 class 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.

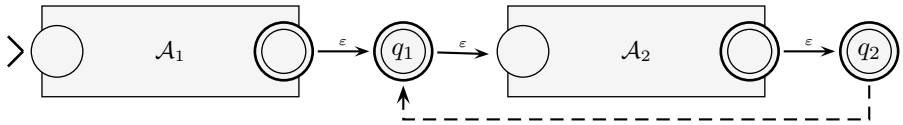


Fig. 2. Construction of a cap automaton for the language $\mathcal{L}(\mathcal{A}_1)\mathcal{L}(A_2)^\circ$.

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 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 12. *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 $\cup_{\mathcal{P}\vdash >q_0} \cup_{\mathcal{P}\vdash \underline{\underline{q_1}}} R_{q_0,q_1}$ and $\cup_{\mathcal{P}\vdash >q_0} \cup_{\mathcal{P}\vdash \underline{\underline{q_1}}} \cup_{\mathcal{P}\vdash q_2 \dashrightarrow q_1} R_{q_0,q_1}(R_{q_1,q_2})^\circ$. 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 built 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. 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 $R_1R_2^\circ$ 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

We now construct cap automata that test l-safety and r-safety of entailment judgments. The same construction applies for finite trees and possibly infinite trees. The only difference between both cases is hidden in different application conditions required for completeness. The appropriate conditions for both cases can be ensured by different preprocessing steps and satisfiability tests (as Prop. 21 will explain).

The automata construction is given in Table 1. 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 $\mathcal{P}_r(\varphi \models^?$

alphabet	$A_\Sigma = \{1, \ldots, \mathsf{ar}_f\}$	
states	$\mathcal{P}_\theta \vdash (s, s')$	if $s, s' \in V_\varphi \cup \{x, y, _\}$
	$\mathcal{P}_\theta \vdash \underline{\underline{\mathit{all}}}$	
initial state	$\mathcal{P}_\theta \vdash >(x, y)$	
final states	$\mathcal{P}_l \vdash \underline{\underline{(u, s)}}$	if $u{\leq}f(u_1, \ldots, u_n)$ *in* φ
	$\mathcal{P}_r \vdash \underline{\underline{(s, v)}}$	if $f(v_1, \ldots, v_n){\leq}v$ *in* φ
descend	$\mathcal{P}_\theta \vdash (u, s) \xrightarrow{i} (u_i, _)$	if $u{\leq}f(u_1, \ldots, u_n)$ *in* $\varphi,\ i \in A_\Sigma$
	$\mathcal{P}_\theta \vdash (s, v) \xrightarrow{i} (_, v_i)$	if $f(v_1, \ldots, v_n){\leq}v$ *in* $\varphi,\ i \in A_\Sigma$
	$\mathcal{P}_\theta \vdash (u, v) \xrightarrow{i} (u_i, v_i)$	if $\begin{cases} u{\leq}f(u_1, \ldots, u_n) \text{ in } \varphi, \\ f(v_1, \ldots, v_n){\leq}v \text{ in } \varphi,\ i \in A_\Sigma \end{cases}$
bot	$\mathcal{P}_\theta \vdash \underline{\underline{(u, s)}} \xrightarrow{i} \underline{\underline{\mathit{all}}}$	if $u{=}\bot$ *in* $\varphi,\ i \in A_\Sigma$
top	$\mathcal{P}_\theta \vdash \underline{\underline{(s, v)}} \xrightarrow{i} \underline{\underline{\mathit{all}}}$	if $v{=}\top$ *in* $\varphi,\ i \in A_\Sigma$
reflexivity	$\mathcal{P}_\theta \vdash \underline{\underline{(u, u)}} \xrightarrow{i} \underline{\underline{\mathit{all}}}$	if $u \in V_\varphi \cup \{x, y\}, i \in A_\Sigma$
all	$\mathcal{P}_\theta \vdash \underline{\underline{\mathit{all}}} \xrightarrow{i} \underline{\underline{\mathit{all}}}$	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 1. Construction of the cap automata $\mathcal{P}_\theta = \mathcal{P}_\theta(\varphi \models^? x{\leq}y)$ for both sides $\theta \in \{l, r\}$.

$x{\leq}y$). 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 automaton 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.

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.

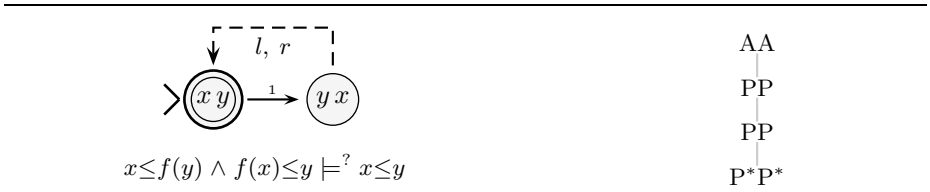


Fig. 3. Automata construction for Example 7. Entailment holds.

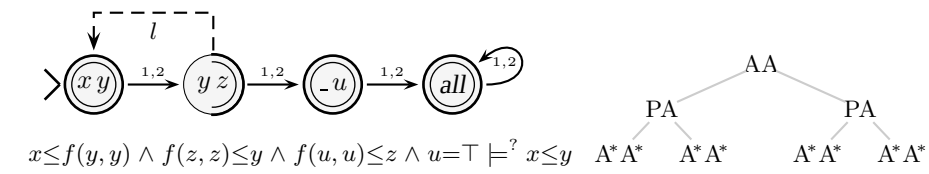


Fig. 4. Automata construction for Example 10. Entailment holds.

We first illustrate the automata construction for the unary Example 7, 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. 7) this proves entailment.

We now consider the more complex binary Example 10 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 $(_, u) \xrightarrow{1,2}$ *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.

Theorem 13 (Reduction). *NSSE for a signature $\{\bot, f, \top\}$ can be reduced in polynomial time to the universality problem of cap automata over the alphabet $\{1, \ldots, \mathsf{ar}_f\}$. This holds equally for finite or possibly infinite trees.*

The automata construction is proven sound in Sec. 7 and complete in Sec. 8.

7 Soundness

In this section, we prove the soundness of the automata construction. The proof is non-trivial and requires a new argument compared to [15]. This argument (see the proof of Proposition 20) is based on Lemma 3 from the introductory Sec. 2.

Proposition 14 (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 17) and second in class P (Proposition 20). For both steps, we have to give meaning to the transitions of the constructed finite automata. For variables x, and paths $\pi \in \{1, \ldots, \mathsf{ar}_f\}^*$, we define *path terms* $\pi(x)$ recursively such that we can interpret *path constraints* of the form $x{\leq}\pi(y)$ and $\pi(y){\leq}x$.

$$\varepsilon(x) =_{\mathrm{def}} x \qquad \text{and} \qquad i\pi(x) =_{\mathrm{def}} f(\top, \ldots, \underbrace{\pi(x)}_{i\text{-th position}}, \ldots, \top)$$

Lemma 15 (Semantics of transitions). *For all constraints φ, variables x, y, u, v, tokens $s, s' \in V_\varphi \cup \{x, y, _\}$, and words $\pi \in \{1, \ldots, \mathsf{ar}_f\}^*$:*

$$\textit{if } \mathcal{P}_l(\varphi \models^? x{\leq}y) \vdash (u, s) \xrightarrow{\pi} (v, s') \textit{ then } \varphi \models u{\leq}\pi(v).$$

Proof. This can be shown by induction on paths π. Note that the considered transitions are built by the **descend** rule exclusively.

Lemma 16 (Path constraints and safety). *If $\alpha \models x{\leq}\pi(u)$ then all proper prefixes of π are l-safe for $\alpha \models^? x{\leq}y$.*

Proof. 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 $\alpha \models^? x{\leq}y$.

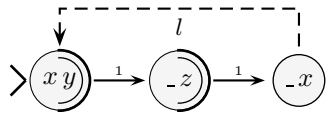
Proposition 17 (Soundness for class A). *For all φ, variables x, y it holds that all paths π with class A accepted by $\mathcal{P}_l(\varphi \models^? x{\leq}y)$ are l-safe for $\varphi \models^? x{\leq}y$.*

Proof. We have to consider all recognizing transitions of the constructed finite automaton. For illustration, we treat the case where π is accepted in a state to which the **final states** rule applies, i.e. π is recognized by a transition of the following form:

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

Lemma 15 yields $\varphi \models x{\leq}\pi i(u')$. Thus π is l-safe for $\varphi \models^? x{\leq}y$ by Lemma 16.

We now approach the soundness of P-edges. It mainly relies on Lemma 18 in combination with Lemma 3 from the introduction.



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

CA
CA
CP
$\mathrm{C}^*\mathrm{P}^*$

Fig. 5. Entailment holds even though the language of the left cap automaton is empty.

Lemma 18 (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 two cases depending on 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 16 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 19 (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 $\alpha \models^{?} x\leq y$.*

Proposition 20 (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 exists 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 15 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 3 on word equations** to our assumption $\mu' \in pr(\mu^*)$ to derive $\mu' \in pr(\mu\mu')$. This verifies the assumptions of Lemma 18 which shows that μ' is l-safe for $\alpha \models^{?} u\leq v$. Finally, the composition Lemma 19 show that $\pi\mu'$ is l-safe for $\alpha \models^{?} x\leq y$ as required.

reflexivity	$\varphi \vdash x{\leq}x$	if $x \in V_\varphi$
transitivity	$\varphi \vdash x{\leq}z$	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 2. Syntactic support.

closure 1a	$f(x_1,\ldots x_n){\leq}y$ *in* φ	if $f(x_1,\ldots x_n){\leq}x$ *in* φ and $\varphi \vdash x{\leq}y$
closure 1b	$y{\leq}f(x_1,\ldots x_n)$ *in* φ	if $x{\leq}f(x_1,\ldots x_n)$ *in* φ and $\varphi \vdash y{\leq}x$
closure 2a	$f(y_1,\ldots y_n){\leq}x$ *in* φ	if $f(x_1,\ldots x_n){\leq}x$ *in* φ and for all $1 \leq i \leq n:\ \varphi \vdash y_i{\leq}x_i$
closure 2b	$x{\leq}f(y_1,\ldots y_n)$ *in* φ	if $x{\leq}f(x_1,\ldots x_n)$ *in* φ and for all $1 \leq i \leq n:\ \varphi \vdash x_i{\leq}y_i$

Table 3. Closure rules.

8 Completeness

In this section, we state the completeness of the automata construction. Its proof is non-trivial but can be obtained as a straightforward extension of the completeness proof in [15].

We first note that completeness depends on additional assumptions. For instance, entailment holds in Fig. 5 even though the corresponding left cap automaton does not accept any word. The problem is that the constraint on the left $f(x){\leq}z \ \wedge\ z{\leq}f(z)$ entails $x{\leq}z$ which has to be derived by decomposition at before hand. Otherwise, it will not be taken into account by the automaton construction. The closure under these rules can also be used for satisfiability checking and can be imposed in polynomial time w.l.o.g.

We define the notion of syntactic support $\varphi \vdash x{\leq}y$ in Table 2 (rather than admitting inequations $x{\leq}y$ on the left hand side of entailment judgments). The definition is based on three standard rules expressing **reflexivity**, **transitivity**, and the **decomposition** property. We then call a constraint φ *closed* if it satisfies all **closure** rules in Table 3. The constraint in Fig. 5 is not closed. Its closure contains $x{\leq}f(z)$ in addition.

Proposition 21 (Completeness). *Let φ be a closed constraint, x, y variables and $\theta \in \{l, r\}$ a side. If $\varphi \wedge x{\leq}y$ is satisfiable for finite (resp. infinite) trees. Then $\mathcal{P}_\theta(\varphi \models^? x{\leq}y)$ accepts all paths that are θ-safe for $\varphi \models^? x{\leq}y$.*

The proof of Proposition 21 covers the rest of this section. By symmetry, it is sufficient to treat the case $\theta = l$. We prove it by contradiction. Let φ be closed constraint with $x, y \in V_\varphi$ such that $\varphi \wedge x{\leq}y$ is satisfiable, and ν a minimal path in $\mathbf{A}^*_\Sigma \setminus \mathcal{P}_\theta(\varphi \models^? x{\leq}y)$. We prove that ν is not l-safe, i.e. there exists a variable assignment α satisfying:

$$\alpha \quad \models \quad \varphi \wedge x(\nu) \neq_\Sigma f \wedge \bigwedge_{\nu' \leq \nu} x(\nu') \neq_\Sigma \bot \wedge \bigwedge_{\nu' \leq \nu} y(\nu') \neq_\Sigma \top.$$

$\varphi \vdash x{\leq}\varepsilon(y)$	if $\varphi \vdash x{\leq}y$
$\varphi \vdash \varepsilon(y){\leq}y$	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 4. Additional Syntactic Support.

a. φ *in sat*(φ,ν)	
b. for all $q^z_\pi \in W(\varphi,\nu):\ q^z_\pi{\leq}f(q^z_{\pi 1},\ldots,q^z_{\pi n}) \wedge f(q^z_{\pi 1},\ldots,q^z_{\pi n}){\leq}q^z_\pi$ *in sat*(φ,ν)	if $\pi < \nu$
c. $q^y_\nu{\leq}f(q^y_{\nu 1},\ldots,q^y_{\nu n}) \wedge f(q^y_{\nu 1},\ldots,q^y_{\nu n}){\leq}q^y_\nu$ *in sat*(φ,ν)	
d. $q^x_\nu{=}\top$ *in sat*(φ,ν)	
e. for all $q^z_\pi \in W(\varphi,\nu), u \in V_\varphi, i \in \mathbf{A}_\Sigma:\ q^z_\pi{\leq}i[\,u\,]$ *in sat*(φ,ν)	if $\varphi \vdash z{\leq}\pi i(u)$
f. for all $q^z_\pi \in W(\varphi,\nu), u \in V_\varphi, i \in \mathbf{A}_\Sigma:\ i[\,q^z_\pi\,]{\leq}u$ *in sat*(φ,ν)	if $\varphi \vdash z{\leq}i\pi(u)$
g. for all $q^z_{o\pi}, q^{z'}_{o'\pi}{\in}W(\varphi,\nu), i{\in}\mathbf{A}_\Sigma$: $q^z_{o\pi}{\leq}i[\,q^{z'}_{o'\pi}\,]$ *in sat*(φ,ν)	if $\exists v.\varphi \vdash z{\leq}o(v),\ \varphi{\vdash}o'i(v){\leq}z'$

Table 5. Saturation *sat*(φ,ν) of a constraint φ at path ν.

Every solution of $\varphi \wedge x(\nu) =_\Sigma \top \wedge y(\nu) =_\Sigma f$ has this property. It remains to show that such a solution exists. We will define a constraint *sat*(φ,ν) in Definition 23 that is satisfaction equivalent to $\varphi \wedge x(\nu) =_\Sigma \top \wedge y(\nu) =_\Sigma f$ and prove that it is satisfiable.

The definition of *sat*(φ,ν) requires an additional notion of *syntactic support* given in Table 4. Furthermore, we write $x{\leq}i[\,y\,]$ as a shortcut for $x{\leq}f(y_1,\ldots,y_n)$ where $y_1,\ldots,y_n$ are fresh variables except for $y_i = y$.

Definition 22. We call a set $D \subseteq \{1,...,n\}^*$ *domain closed* if D is prefixed-closed and satisfies the following property for all $\pi \in \{1,...,n\}^*$: if $i,j \in \{1,...,n\}$ and $\pi i \in D$ then $\pi j \in D$. The *domain closure dc*(D) is the least domain closed set containing D.

Definition 23 (Saturation). Let φ be a constraint over $\Sigma = \{\bot, f, \top\}$ which contains variables x, y. Let $\nu \in \mathbf{A}^*_\Sigma$ where $\mathbf{A}_\Sigma = \{1,\ldots,\mathsf{ar}_f\}$.

For every $z \in \{x,y\}$ and $\pi \in$ *dc*$(\{\nu 1,\ldots,\nu n\})$ let q^z_π be a fresh variable and *W*(φ,ν) the collection of these fresh variables. The saturation *sat*(φ,ν) of φ at path ν is the constraint of minimal size satisfying properties **a**.–**g**. of Table 5.

Lemma 24. *If φ is closed, $\varphi \wedge x{\leq}y$ is satisfiable (over finite resp. possible infinite trees), and ν is the minimal word in $\mathbf{A}^*_\Sigma \setminus \mathcal{L}(\mathcal{P}_l(\varphi \models^? x{\leq}y))$ then the saturation sat(φ,ν) is satisfiable.*

Proof. We have to prove that the saturation *sat*(φ,ν) satisfiable. First note that *sat*(φ,ν) is closed. We omit the proof for lack of space. We will exploit the fact that a closed constraint is satisfiable if and only if it does not contain any clash according to Table

label clash 1	$x{=}\top \wedge y{=}\bot$ *in* φ and $\varphi \vdash x{\leq}y$
label clash 2	$x{=}\top \wedge x{\leq}f(x_1,\ldots,x_n)$ *in* φ
label clash 3	$x{=}\bot \wedge f(x_1,\ldots,x_n){\leq}x$ *in* φ
cycle clash	$x_1{=}x_{n+1}$ and forall $i \leq n:\ x_i{\leq}f(...,x_{i+1},...) \wedge f(...,x_{i+1},...){\leq}x_i$ *in* φ

Table 6. Clashes: the **cycle clash** has to be omitted in the case of infinite trees

6. Beside of the usual label clashes, there is a cycle clash expressing the finiteness of trees, which has to be removed in the case of infinite trees.

For illustration, we elaborate one of the more interesting cases of the proof of clash-freeness. Assume that there were a **label clash** containing $q^x_\nu{=}\top$ that was introduced by saturation (rule **d.** of Table 5). Such a clash must of the form **label clash1** or **label clash2**. We treat the second case only. There exist $w, w_1, \ldots, w_n \in V_{sat(\varphi,\nu)}$ such that: $q^x_\nu{=}\top$ and $q^x_\nu{\leq}f(w_1,\ldots,w_n)$ *in* $sat(\varphi,\nu)$.

We have to distinguish all possible rules of Definition 23 which may have added $q^x_\nu{\leq}f(w_1,\ldots,w_n)$ to $sat(\varphi,\nu)$; the candidates are **b**, **e**, and **g**. Rule **b** cannot be applied since it requires $\pi < \nu$. Rule **e** imposes $\varphi \vdash x{\leq}\nu i(w_i)$; thus π is l-safe which a contradiction to our assumption $\nu \notin \mathcal{L}(\mathcal{P}_l(\varphi \models^? x{\leq}y))$. Finally consider rule **g**. This rule requires that there is a variable v in V_φ with: $\varphi \vdash x{\leq}o(v)$ and $\varphi \vdash o'i(v){\leq}z'$.

- **Case** $z' = y$ **and** $o = o'i$**:** We have $\varphi \vdash x{\leq}o(v)$ and $\varphi \vdash o(v){\leq}y$. The **reflexivity** rule in the automaton construction yields $o \in \mathcal{L}(\mathcal{P}_l)$ and thus $\nu \in \mathcal{L}(\mathcal{P}_l)$ - a contradiction.
- **Case** $z' = y$ **and** $o = o'i\sigma$ **where** $\sigma \neq \varepsilon$**:** We have $\varphi \vdash x{\leq}o'i\sigma(v)$ and $\varphi \vdash o'i(v){\leq}y$. We can identify a transition $\mathcal{P}_l \vdash (x,y) \xrightarrow{o'i} (s,v) \xrightarrow{\sigma} (v,t)$, where s and t are arbitrary tokens. According to the automaton construction there is a left P-edge which validates all paths $o'i\,\sigma'$ with $o'i\,\sigma' < o'i\,\sigma\sigma'$ to be in $\mathcal{L}(\mathcal{P}_l)$. Since $o'i{\leq}\nu$ and $o'i\sigma{\leq}\nu$ it holds $\nu \in \mathcal{L}(\mathcal{P}_l)$ - a contradiction.
- The remaining cases where $z' = x$ or $o \neq o'i\pi$ are analogous.

9 Restrictions of Constructed Automata

Constructed cap automata satisfy restrictions that we need to translate back (Sec 11).

Definition 25. We call a cap automaton $\mathcal{P}$ over $\mathbf{A}$ *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: For all transitions $\mathcal{P} \vdash q_1 \xrightarrow{i} \underline{\underline{q_2}},\ i \in \mathbf{A}$ from a non-final state q_1 it holds that q_2 is universal: for all $\pi \in \mathbf{A}^*$ there is q_3 with $\mathcal{P} \vdash \underline{\underline{q_2}} \xrightarrow{\pi} \underline{\underline{q_3}}$.

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 $\mathcal{P} \vdash {>}q_0$ is the initial state and $q \neq q'$ then the language $\{\pi' \mid \pi\pi' \in \mathcal{L}(\mathcal{P})\}$ is universal.

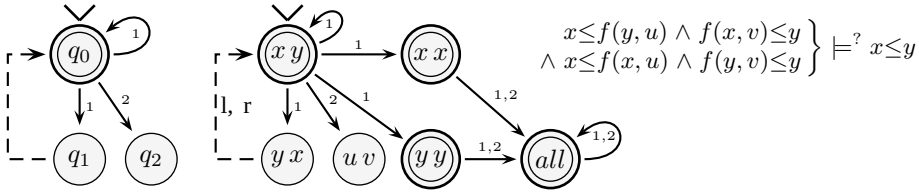


Fig. 6. An unshuffled cap automaton (on the left) and a shuffled extension (in the middle).

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 that is built on top of a deterministic finite automaton 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.

A cap automaton where the word 1 violates the shuffle condition is given on the left-hand of Table 6. A shuffled extension of this automaton is displayed in the middle of Table 6. This automaton is constructed from an entailment judgment displayed to the right. This example illustrates that shuffling simulates the interaction between multiple lower and upper bounds in constraints.

Proposition 26. *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 licence 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 1 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 1), $\mathcal{P}_l \vdash (x, y) \xrightarrow{\pi} (u, u)$ 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 licenced 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.

10 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 27. We call a cap set expression over alphabet $\mathbf{A}$ *restricted* if it is a shuffled expression generated by the following grammar, where R_1, R_2, R range over regular expressions over $\mathbf{A}$:

$$F ::= pr(R_1 R_2^\circ) \mid \ R\mathbf{A}^* \mid \ F_1 \cup F_2$$

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

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

Proposition 28. *Universality of restricted cap set expressions and restricted cap automata are equivalent modulo deterministic polynomial time transformations.*

Proof. It is easy to see that the cap automata that are constructed for restricted cap expressions (proof or Proposition 12) are gap-universal, strictly cap, and shuffled. They can be made strictly ε-free by a post-processing step (Lemma 30).

Conversely, given a restricted cap automaton $\mathcal{P}$, we can express the regular part of $\mathcal{P}$ by a restricted cap set expression $pr(R_1\varepsilon^\circ) \cup R_2\mathbf{A}^*$, 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.

Our next goal is 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 29. *If a cap automaton $\mathcal{P}$ has a unique initial state, then this state can be normalized (while preserving the language gap-universality, strict cap, and shuffle).*

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 30. *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.*

Proof. First, we eliminate ε-edges in the underlying finite automaton. This yields a cap automaton which may have more than one initial states. W.l.o.g. 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 29.

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.

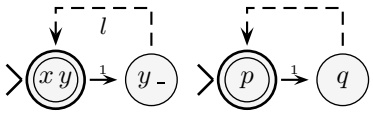
11 Back Translation for Restricted Cap Automata

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

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 $\mathcal{P}$.

$x{\leq}f(y) \models^{?} x{\leq}y$



$$\left.\begin{array}{c} l(p){\leq}f(l(q)) \\ \wedge\ f(r(q)){\leq}r(p) \\ \wedge\ l(p){\leq}f(r(p)) \end{array}\right\} \models^{?} l(p){\leq}r(p)$$

Fig. 7. A judgment, its pair of cap automata, and the back translation of the left cap automaton.

Definition 31. 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 automaton $\mathcal{P}$ with initial state $\mathcal{P}\vdash\ {>}q_0$ 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 $\mathbf{A}^*$.

Proposition 32 (Correctness). *Every complete and restricted cap automaton $\mathcal{P}$ with initial state $\mathcal{P}\vdash\ {>}q_0$ over alphabet $\mathbf{A}$ satisfies:*

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

Proof. 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 $\pi\in\mathbf{A}^*$ 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 $\mathcal{P}_r(J(\mathcal{P}))$.

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. 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 $\underline{\underline{\textit{all}}}$. 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 $\underline{\underline{\textit{all}}}$. 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 the cap automaton $\mathcal{P}_l(x{\leq}f(y) \ \models^? \ x{\leq}y)$ given in Table 7. 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 33. *Let $\mathcal{P}$ be a restricted cap automaton with initial state $\mathcal{P} \vdash \ {>}q$. The constructed constraint $\varphi_{\mathcal{P}}$ is then closed and $\varphi_{\mathcal{P}} \wedge l(q){\leq}r(q)$ is satisfiable over finite and infinite trees.*

Theorem 34 (Back translation). *Universality of restricted cap automata over the alphabet $\{1, \ldots, \mathsf{ar}_f\}$ can be reduced in polynomial time to NSSE with signature $\{\bot, f, \top\}$ (both over finite 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 32). Since $\varphi_{\mathcal{P}}$ is closed and clash-free (Lemma 33), the latter is equivalent to that NSSE holds for the judgment $J(\mathcal{P})$ (Theorem 13).

12 Equivalence of Variants of NSSE

We prove Corollary 2 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 35. *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 addition

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

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

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

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 iff $\mathcal{L}(\mathcal{P}')$ is universal. We encode a letter of A in two letters of A' to the base n via the standard encoding $d : A \to A' \times A'$:

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

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 $\mathcal{L}(\mathcal{P})$ iff 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 non-constant function symbol 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. We finally note that cap automata seem to be related to tree automata with equality tests (tuple reduction automata).

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. R. M. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, September 1993.
2. F. Baader and K. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. In *Journal of Symbolic Computation*, volume 21, pages 211–243, 1996.
3. V. Durnev. Unsolvability of positive $\forall\exists^3$-theory of free groups. In *Sibirsky mathematichesky jurnal*, volume 36(5), pages 1067–1080, 1995. In Russian, also exists in English translation.
4. J. Eifrig, S. Smith, and V. Trifonow. Sound polymorphic type inference for objects. In *ACM Conference on Object-Oriented Programming: Systems, Languages, and Applications*, 1995.
5. J. Eifrig, S. Smith, and V. Trifonow. Type inference for recursively constrained types and its application to object-oriented programming. *Elec. Notes in Theor. Comp. Science*, 1, 1995.
6. Y. Fuh and P. Mishra. Type inference with subtypes. *Theo. Comp. Science*, 73, 1990.
7. F. Henglein and J. Rehof. The complexity of subtype entailment for simple types. In *Proceedings of the 12th IEEE Symposium on Logic in Computer Science*, pages 362–372, 1997.
8. F. Henglein and J. Rehof. Constraint automata and the complexity of recursive subtype entailment. In *25th Int. Conf. on Automata, Languages, & Programming*, LNCS, 1998.
9. D. Kozen, J. Palsberg, and M. I. Schwartzbach. Efficient inference of partial types. *Journal of Computer and System Sciences*, 49(2):306–324, 1994.
10. D. Kozen, J. Palsberg, and M. I. Schwartzbach. Efficient recursive subtyping. *Mathematical Structures in Computer Science*, 5:1–13, 1995.
11. G. Makanin. The problem of solvability of equations in a free semigroup. *Math. USSR Sbornik*, 32, 1977. (English translation).
12. J. C. Mitchell. Type inference with simple subtypes. *The Journal of Functional Programming*, 1(3):245–285, July 1991.
13. M. Müller, J. Niehren, and R. Treinen. The first-order theory of ordering constraints over feature trees. In *IEEE Symposium on Logic in Computer Science*, pages 432–443, 1998.
14. J. Niehren, M. Müller, and J.-M. Talbot. Entailment of atomic set constraints is PSPACE-complete. In *14th IEEE Symposium on Logic in Computer Sience*, pages 285–294, 1999.
15. J. Niehren and T. Priesnitz. Entailment of non-structural subtype constraints. In *Asian Computing Science Conference*, LNCS, pages 251–265. Springer-Verlag, Berlin, 1999.
16. J. Palsberg, M. Wand, and P. O'Keefe. Type Inference with Non-structural Subtyping. *Formal Aspects of Computing*, 9:49–67, 1997.
17. W. Plandowski. Satisfiability of word equations with constants is in PSPACE. In *Proc. of the 40th IEEE Symp. on Found. of Comp. Science*, pages 495–500, 1999.
18. F. Pottier. Simplifying subtyping constraints. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*, pages 122–133, 1996.
19. F. Pottier. A framework for type inference with subtyping. In *Proc. of the third ACM SIGPLAN International Conference on Functional Programming*, pages 228–238, 1998.
20. F. Pottier. *Type inference in the presence of subtyping: from theory to practice*. PhD thesis, Institut de Recherche d'Informatique et d'Automatique, 1998.
21. J. Rehof. Minimal typings in atomic subtyping. In *ACM Symposium on Principles of Programming Languages*. ACM Press, 1997.
22. J. Rehof. *The Complexity of Simple Subtyping Systems*. PhD thesis, DIKU, Copenh., 1998.
23. K. U. Schulz. Makanin's algorithm for word equations – two improvements and a generalization. In *Word Equations and Related Topics*, LNCS 572, pages 85–150, 1991.
24. Y. Vazhenin and B. Rozenblat. Decidability of the positive theory of a free countably generated semigroup. In *Math. USSR Sbornik*, volume 44, pages 109–116, 1983.