Nominal Unification with Atom and Context Variables

Technical Report Frank-59

Institut für Informatik, Fachbereich Mathematik und Informatik

Goethe-Universität Frankfurt am Main, Germany

Manfred Schmidt-Schauß[1] and David Sabel[1]

1 **Goethe-University, Frankfurt, Germany**
`{schauss,sabel}@ki.cs.uni-frankfurt.de`

Abstract

Automated deduction in higher-order program calculi, where properties of transformation rules are demanded, or confluence or other equational properties are requested, can often be done by syntactically computing overlaps (critical pairs) of reduction rules and transformation rules. Since higher-order calculi have alpha-equivalence as fundamental equivalence, the reasoning procedure must deal with it. We define ASD1-unification problems, which are higher-order equational unification problems employing variables for atoms, expressions and contexts, with additional distinct-variable constraints, and which have to be solved w.r.t. alpha-equivalence. Our proposal is to extend nominal unification to solve these unification problems. We succeeded in constructing the nominal unification algorithm NomUnifyASC. We show that NomUnifyASC is sound and complete for these problem class, and outputs a set of unifiers with constraints in nondeterministic polynomial time if the final constraints are satisfiable. We also show that solvability of the output constraints can be decided in NEXPTIME, and for a fixed number of context-variables in NP time. For terms without context-variables and atom-variables, NomUnifyASC runs in polynomial time, is unitary, and extends the classical problem by permitting distinct-variable constraints.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases automated deduction, nominal unification, lambda calculus, functional programming

1 Introduction

Automated deduction in higher-order program calculi, where properties of transformation rules are demanded, or confluence or other equational properties are requested, can often be done by syntactically computing overlaps (critical pairs) of reduction rules and transformation rules. Since higher-order calculi have alpha-equivalence as fundamental equivalence, the reasoning procedure must deal with it. We define ASD1-unification problems, which are higher-order equational unification problems with variables for atoms, expressions, and contexts, with additional distinct-variable constraints, and which have to be solved w.r.t. alpha-equivalence. Our proposal is to extend nominal unification to solve these unification problems. The appeal of classical nominal unification is that it solves higher-order equations modulo α-equivalence in quadratic time and outputs at most a single most general unifier [36, 6, 17].

Our intended application is the *diagram-method*, which is a syntactic proof method (e.g. [37, 12, 33]) to show properties like correctness of program transformations. As an example consider the reduction rule (cp) in the call-by-need lambda-calculus with let (see e.g. [2, 23, 33]) ($\texttt{let}\ x = \lambda y.S\ \texttt{in}\ C[x]) \rightarrow (\texttt{let}\ x = \lambda y.S\ \texttt{in}\ C[\lambda y.S]$) with the restriction that x is not bound by context C. The diagram-based proof method for correctness of program transformations computes possible overlaps of the left-hand side (and in certain cases also of the right-hand side) of a program transformation with the left-hand sides of the reduction rules. An example equation is $\texttt{let}\ A_1 = \lambda A_2.S_1\ \texttt{in}\ D_1[\lambda A_2.S_1] \doteq \texttt{let}\ A_3 = \lambda A_4.S_2\ \texttt{in}\ D_2[A_3]$ where A, S, D are variables standing for concrete variables (called atoms), expressions, and contexts, respectively. The equation comes together with constraints on possible occurrences of atoms, which can be formulated using the distinct-variable condition (DVC) [5, 2].

A generalized situation for overlap computation is represented by the equation $R[\ell] = C[\ell']$, where R is a reduction context (in which reduction takes place), ℓ is a left hand side of a reduction rule, C an arbitrary context, and ℓ' is the left hand side of a transformation rule. Provided R, C are variables for reduction contexts and general contexts, respectively, solving the unification equation $R[\ell] = C[\ell']$ can be attacked by using nominal unification in the extended language. The proposal of [31] to solve such problems w.r.t. syntactic equality permits an even higher expressiveness of the language, but the support for alpha-equivalence reasoning is missing, and hence several variants of extra constraints are necessary and further reasoning needs a technically detailed analysis of renamings [29].

Results. A sound and complete algorithm NomUnifyASC for nominal unification of ASD1-unification problems is constructed. The algorithm NomUnifyASC computes in NP time a solution including a constraint (Thm. 5.9) and the collecting version produces at most exponentially many outputs. The algorithm NomFreshASC that checks satisfiability of the constraints of a solution runs in NEXPTIME (Proposition 5.8), hence solvability of ASD1-unification problems can be decided in NEXPTIME (Thm. 5.10). Since the number of context-variables is the only parameter in the exponent of the complexity, we obtain that if the number of context-variables is fixed, then the algorithm NomUnifyASC runs as a decision algorithm in NP time.

For computing diagrams (in the diagram method), it is important to obtain a complete set of unifiers. We expect that exponentiality of the number of unifiers is not a problem, since the input is usually very small. For example, the rules of the let-calculus [2] need only one context variable. We show that DVCs are a proper generalisation of freshness constraints if combined with solving equations (Proposition 3.8). A technical innovation is that decomposition for lambda-bindings can be extended to an arbitrary number of nested lambda-bindings thanks to the DVC (see Remark 3.4, Proposition 3.5, and Example 3.6).

Classical nominal unification is generalized by replacing freshness constraints by DVC-constraints such that unitarity and polynomial complexity still hold (Thm. 6.1).

Previous and Related Work. Nominal techniques [27, 26] support machine-oriented reasoning on the syntactic level supporting alpha-equivalence. Nominal unification of (syntactically presented) lambda-expressions was successfully attacked and a quadratic algorithm was developed [36, 6, 17] where technical innovations are the use of permutations on the abstract level and of freshness constraints. The approach is used in higher-order logic programming [8], and in automated theorem provers like nominal Isabelle [34, 35].

There are investigations that extend the expressive power of nominal unification problems: The restriction that bound variables are seen as atoms can be relaxed: Equivariant unification

[7, 10] permits atom-variables and permutation-variables which however, appear to add too much expressive power as mentioned in [7, 10]. A restricted language (allowing atom-variables, but without permutations at all) and its nominal unification is analyzed in [16]. An investigation of nominal unification with atom-variables and a lazy-guessing algorithm is described in [32, 15]. Nominal unification for a lambda-calculus with function constants and a recursive-let is developed in [30] and shown to run in NP time. Reasoning on nominal terms in higher-order rewrite systems and narrowing as a general, but not provably efficient method for unification is described in [4]. Nominal techniques to compute overlaps w.r.t. all term positions are in [3], but there are no context-variables and reduction strategies cannot be encoded. If there are no binders, then the problem statement can be generalized to first-order terms with arbitrary occurrences of context-variables and the unification problem is in PSPACE [14].

Classical nominal unification is strongly related to higher-order pattern unification [9, 18, 22, 28, 24] which is a decidable fragment of (undecidable) higher-order unification [13] and has most general unifiers (i.e. is unitary). A slight extension to pattern unification that is unitary and decidable is described in [19]. Another line of research for reasoning with binders is the foundation of higher-order abstract syntax [25], and extensions of higher-order pattern unification as in [1], which, however, cannot adequately deal with ASD1-unification problems, the problem class of NomUnifyASC. The use of deBruijn indices [11] has some advantages in representing lambda expressions and avoids alpha-renamings, but is not appropriate for our problem, since we have to deal with free variables and with context variables that may capture variables.

Outline. Sect. 2 contains the definitions and extensions of nominal syntax. In Sect. 3 the preparations for the nominal unification algorithm are done, and in Sect. 4 the algorithm NomUnifyASC is introduced, which consists of a set of (non-deterministic) rules, and also the constraint checking algorithm NomFreshASC. In Sect. 5 the properties of the algorithms are analyzed. In Sect. 6 the special case NL_{aS} is reconsidered, and we illustrate how the unification algorithm operates on examples. We conclude in Sect. 7. Most of the proofs are omitted in the main part, but given in the appendix.

2 Nominal Languages and Nominal Unification

We explain the nominal (meta-)languages representing higher-order languages, their semantics and also several operations.

Let $\mathcal{F}$ be a set of function symbols where each $f \in \mathcal{F}$ has a fixed arity $ar(f) \geq 0$, and $\mathcal{F}$ contains at least two function symbols, one of arity 0 (a constant) and one of arity ≥ 2. Let $\mathcal{A}t$ be the set of *atoms* ranged over by a, b; $\mathcal{A}$ be the set of *atom-variables* ranged over by A, B; $\mathcal{S}$ be the set of *expression-variables* ranged over by S, T standing for expressions; $\mathcal{D}$ be the set of *context-variables* ranged over by D standing for single hole-contexts; and $\mathcal{P}$ be the set of *permutation-variables* ranged over by P standing for finite permutations on $\mathcal{A}t$, i.e. bijections π on $\mathcal{A}t$ such that their *support* $supp(\pi) = \{a \in \mathcal{A}t \mid \pi(a) \neq a\}$ is a finite set.

The ground expressions are lambda-expressions extended by function symbols, where the lambda-variables are called atoms. Contexts in the ground language are expressions over a language extended with a symbol $[\cdot]$ the hole (occurring only once), where expressions can be plugged in. The language will be enriched by symbols for atoms, expressions, contexts, and permutations, where the latter are mappings that may change the names of atoms and are represented by lists of swappings $(a\,b)$.

▶ **Definition 2.1** (The nominal language NL_{aASCP} with atom-variables, expression-variables, context-variables, and permutation-variables). The syntax of NL_{aASCP} is defined by the grammar

$$\begin{array}{lcl} X \in \mathcal{AE} & ::= & a \mid A \mid \pi{\cdot}A \\ e \in \mathcal{E} & ::= & X \mid S \mid \pi{\cdot}S \mid (f\ e_1 \ldots e_{ar(f)}) \mid \lambda X.e \mid C[e] \\ C \in \mathcal{C} & ::= & [\cdot] \mid D \mid \pi{\cdot}D \mid (f\ e_1 \ldots [\cdot] \ldots e_n) \mid \lambda X.[\cdot] \mid C_1[C_2] \\ \pi & ::= & \emptyset \mid (A_1\ A_2)\cdot\pi \mid (A\ a)\cdot\pi \mid (a\ A)\cdot\pi \mid (a_1\ a_2)\cdot\pi \mid P\cdot\pi \mid P^{-1}\cdot\pi \end{array}$$

with the categories $\mathcal{AE}$ of atom expressions, $\mathcal{E}$ of expressions, $\mathcal{C}$ of contexts, and π of permutations. Here A, S, D, and P is an atom-variable, expression-variable, context-variable, or permutation-variable, respectively.

Nested permutations are forbidden, e.g. swappings $(\pi_1{\cdot}A_1\ \pi_2{\cdot}A_2)$ are excluded for simplicity of algorithms, and since their expressive power is already available using equations and constraints. We use positions (tree addresses) in expressions, where we ignore permutation expressions in the addressing scheme. *Sublanguages of NL_{aASCP}* are denoted by NL_M, where M is a substring of $aASCP$, and the grammar is restricted accordingly. The mainly used languages are NL_a as ground language for the solutions, NL_{aASC} as the expression language for the input, and NL_{aASCP} as the working language inside of the unification algorithm.

▶ **Definition 2.2.** A *substitution* $\sigma : NL_{aASCP} \to NL_{aASCP}$ maps atom-variables to atom expressions, expression-variables to expressions, context-variables to contexts, permutation-variables to permutations. A *ground substitution* ρ is a substitution $\rho : NL_{aASCP} \to NL_a$. The mapping on atom-variables, expression-variables, context-variables and permutation-variables determines uniquely a mapping ρ on all expressions of NL_{aASCP}.

The semantics of the symbols is now explained, which will then justify the simplifications below.We use permutation application $\cdot$ as operator and syntactic symbol. Similarly, we use $^{-1}$ as a syntactic symbol in P^{-1} and operator for inversion, and we abbreviate $\emptyset{\cdot}V$ by V for a variable V.

▶ **Definition 2.3** (Operations and Simplifications). In all the languages we use the following operations and simplifications according to the semantics:

$$\begin{array}{ccc} (\pi_1{\cdot}\pi_2)(e) \to (\pi_1{\cdot}(\pi_2{\cdot}e)) & (\pi_1{\cdot}\pi_2)^{-1} \to \pi_2^{-1}{\cdot}\pi_1^{-1} & \pi{\cdot}[\cdot] \to [\cdot] \\ \pi{\cdot}(f\ e_1 \ldots e_n) \to (f\ \pi{\cdot}e_1 \ldots \pi{\cdot}e_n) & \pi{\cdot}(\lambda X.e) \to \lambda\pi{\cdot}X.\pi{\cdot}e & \pi{\cdot}C[e] \to (\pi{\cdot}C)[\pi{\cdot}e] \\ (X_1\ X_2)^{-1} \to (X_1\ X_2) & (C_1C_2)[e] \to C_1[C_2[e]] & \end{array}$$

▶ Remark. The *simplifications* permit standardizations in the various languages:

- In NL_a, all permutation operations can be simplified away.
- In the language NL_{aS} permutations can be represented as list of swappings of length at most $|n-1|$.
- In the language NL_{aASC}, the permutation operations only lead to suspensions of the form $\pi{\cdot}A$,$\pi{\cdot}S$, and $\pi{\cdot}D$.
- In all languages, permutations can be represented as a composition of lists of swappings, permutation-variables P and their inverses P^{-1}.
- Context expressions can be simplified such that context-variables only occur in the form $(\pi{\cdot}D)[e]$.

Let *tops*(e) be the top symbol of e after simplification of permutation applications, i.e. *tops*$(a) = \texttt{atom}$, *tops*$(\lambda X.e) = \lambda$, *tops*$(f\ e_1 \ldots e_n) = f$, *tops*$(\pi{\cdot}A) = A$, *tops*$(\pi{\cdot}S) = S$, and

$tops((\pi{\cdot}D)[e]) = D$. For an expression e or context C in NL_a, we denote with $FA(e)$ or $FA(C)$, resp., the set of free atoms, and with $\mathcal{A}t(e)$ or $\mathcal{A}t(C)$, resp., the set of all atoms. The set of atoms that become bound in the hole of a context C, called the *captured atoms* of C, is denoted as $CA(C)$.

In NL_a, α-equivalence $\sim_\alpha$ is the closure by reflexivity and congruence and the rule $a \notin FA(e') \wedge e \sim_\alpha (a\ b){\cdot}e' \implies \lambda a.e \sim_\alpha \lambda b.e'$.

We also use a notion of α-equivalence for contexts, defined as follows:

▶ **Definition 2.4** (α-equivalences on contexts in NL_a)**.** NL_a-contexts C_1 and C_2 are *α-equivalent*, written $C_1 \sim_\alpha C_2$, iff for all atoms a, $C_1[a] \sim_\alpha C_2[a]$ holds.

E.g., $\lambda a.[\cdot] \not\sim_\alpha \lambda b.[\cdot]$, $\lambda a.\lambda b.\lambda a.[\cdot] \sim_\alpha \lambda b.\lambda b.\lambda a.[\cdot]$, but $\lambda a.\lambda b.[\cdot] \not\sim_\alpha \lambda a.\lambda a.[\cdot]$. In Definition 2.4 it is sufficient for $C_1 \sim_\alpha C_2$, if $C_1[a] \sim_\alpha C_2[a]$ for all $a \in CA(C_1) \cup CA(C_2) \cup \{a'\}$, where a' is a fresh atom. Note that $C_1 \sim_\alpha C_2$ and $e_1 \sim_\alpha e_2$ imply $C_1[e_1] \sim_\alpha C_2[e_2]$, but the reverse is wrong: $(f\ a\ \lambda a.a) \sim_\alpha (f\ a\ \lambda b.b)$, but $(f\ a\ \lambda a.[\cdot]) \not\sim_\alpha (f\ a\ \lambda b.[\cdot])$ and $a \not\sim_\alpha b$. We explain instantiation modulo α for correctly defining solvability under DVC-restrictions.

▶ **Definition 2.5** (Instantiation modulo α)**.** For testing solvability of equations, we assume that ground substitutions map into $NL_a/{\sim_\alpha}$. An equivalent method is that whenever a variable S or D is replaced by ρ, we use an α-renamed copy of $S\rho$ or $D\rho$, respectively, where the renaming is done by fresh atoms that do not occur elsewhere (called instantiation modulo α), and where comparison is done modulo $\sim_\alpha$.

The following definition explains free/bound variables and the satisfiability of DVC-constraints of expression in $NL_a/{\sim_\alpha}$ without using fresh variables.

▶ **Definition 2.6.** Let e be a normalized NL_{aASCP}-expression and ρ be a ground substitution mapping into $NL_a/{\sim_\alpha}$, such that $e\rho$ is an $NL_a/{\sim_\alpha}$-expression. Then we define the bound atoms $BA(e,\rho)$, and satisfiability of the DVC of (e,ρ) as follows, where the bound atoms introduced by ρ are ignored.

1. If X is an atom a or a suspension $\pi{\cdot}A$ where A is an atom-variable then $BA(X,\rho) = \emptyset$, and the DVC is satisfied.
2. $BA(\pi{\cdot}S,\rho) = \emptyset$, and the DVC is satisfied.
3. $BA(f\ e_1 \dots e_n,\rho) = \bigcup_{i=1}^{n} BA(e_i,\rho)$. The DVC is satisfied, if for all $i \neq j$, $BA(e_i,\rho) \cap (FA(e_j\rho) \cup BA(e_j,\rho)) = \emptyset$ and for all i, the DVC holds for (e_i,ρ).
4. $BA(\lambda X.e,\rho) = BA(e,\rho) \cup \{X\rho\}$. The DVC is satisfied, if it is satisfied for (e,ρ), and if $X\rho \notin BA(e,\rho)$.
5. $BA((\pi{\cdot}D)[e],\rho) = BA(e,\rho) \cup ((\pi)\rho){\cdot}(CA(D\rho))$. The DVC is satisfied, if it is satisfied for D, i.e. $CA(D\rho) \cap FA(D\rho) = \emptyset$ as well as $(BA(e,\rho) \cap ((\pi)\rho){\cdot}((FA(D\rho)) \cup (CA(D\rho))) = \emptyset$, and the DVC is satisfied for (e,ρ).

Note that $BA(e,\rho) \neq BA(e\rho)$, since for $e = \lambda A.S, \rho = \{A \mapsto a, S \mapsto \lambda b.a\ b\}$, we have $BA(e\rho) = \{a,b\}$, but $BA(e,\rho) = \{a\}$ (where we do not distinguish between an atom and the α-equivalence class of an atom).

Nominal unification is connected with formulating and solving constraints. One form of constraints are freshness constraints:

▶ **Definition 2.7** (Freshness-Constraints in NL_{aASCP})**.** *Freshness constraints* in NL_{aASCP} are of the form $a\,\#\,e$ and $A\,\#\,e$. The semantics of freshness constraints is as follows: For a ground substitution ρ, we say that

- ρ *satisfies* $A\,\#\,e$ iff $\rho(A) \notin FA(\rho(e))$;

- ρ *satisfies* $a \# e$ iff $a \notin FA(\rho(e))$;

A set of freshness constraints is *satisfiable* iff there exists a ground substitution ρ in NL_{aASCP} that satisfies all constraints. If ρ satisfies all constraints of a set of freshness constraints, then we say that ρ is a *solution* of the constraint set.

Another form of constraints is derived from the distinct variable convention.

▶ **Definition 2.8** (DVC-Constraints)**.** The *distinct variable condition (*DVC*)* holds for an NL_a-expression e, if all bound atoms in e are distinct, and all free atoms in e are distinct from all bound atoms in e.

A DVC-*constraint* is of the form DVC(e), where e is an expression. A ground substitution ρ *satisfies* DVC(e), iff the distinct variable condition holds for $\rho(e)$ where $\rho(e)$ is generated by using instantiation modulo α (Def. 2.5). A DVC-constraint is *satisfiable* if there exists a satisfying ground substitution ρ. If ρ satisfies all constraints of a set of DVC-constraints, then we say that ρ is a solution of the constraint set.

For example, $f\,a\,\lambda b.(g\,a\,\lambda c.c)$ satisfies the DVC and $f\,b\,\lambda b.b$ violates it. With $\rho = \{S \mapsto \lambda c.c\}$ we have $(f\,(\lambda a.S)\,(\lambda b.S))\rho = f\,(\lambda a.\lambda c_1.c_1)\,(\lambda b.\lambda c_2.c_2)$ and thus ρ satisfies DVC$(f\,(\lambda a.S)\,(\lambda b.S))$. As another example, $\rho' = \{S \mapsto \lambda c.b\}$ violates DVC$(f\,(\lambda a.S)\,(\lambda b.S))$.Finally, note that the constraint DVC$(f\ (\lambda a.S)\ (\lambda a.S))$ is not satisfiable.

▶ Remark. We will prove below in Proposition 3.8 that freshness constraints can be expressed as DVC-constraints if they occur in unification problems.

Although freshness constraints are redundant under the assumptions above, we still use them, since it is an important special case in nominal unification. Later we show, that DVC-constraints are a proper generalization of freshness constraints, dependent on the used language.

We now give an extendedexample of the use of constraints.

▶ **Example 2.9.** A reduction rule of the corresponding calculus is`let` $x{=}\lambda y.s$ `in` $C[x] \rightarrow$ `let` $x{=}\ \lambda y.s$ `in` $C[\lambda y.s]$ where C is a context. (Note that we do not exactly represent the reduction rule from [2] here.) We represent the expressions as

$$(\texttt{let}\ (\lambda A_x.D[A_x])\ (\lambda A_y.S))\ \text{and}\ (\texttt{let}\ (\lambda A_x.D[\lambda A_y.S])\ (\lambda A_y.S)).$$

However, D is not permitted to capture the atom represented by A_x, nor free atoms from S, and S is not permitted to have free occurrences of A_x. Both conditions can be captured by the constraint that instances of `let` $(\lambda A_x.D[A_x])\ (\lambda A_y.S)$ and `let` $(\lambda A_x.D[\lambda A_y.S])\ (\lambda A_y.S)$ have to satisfy the DVC. However, the latter violates the DVC in every case due to the two occurrences of the binder A_y. Hence, we add a renaming to the rule and represent it as

$$(\texttt{let}\ (\lambda A_x.D[A_x])\ (\lambda A_y.S)) \rightarrow (\texttt{let}\ (\lambda A_x.D[(A_y\ A_z){\cdot}\lambda A_y.S])\ (\lambda A_y.S))\ \text{and}\ A_z \# S,$$

which is simplified to

$$(\texttt{let}\ (\lambda A_x.D[A_x])\ (\lambda A_y.S)) \rightarrow (\texttt{let}\ (\lambda x.D[\lambda A_z.(A_y\ A_z){\cdot}S])\ (\lambda A_y.S))\ \text{and}\ A_z \# S.$$

Now the DVC-constraints for both expressions makes sense andproduces the correct conditions.

▶ **Definition 2.10.** Let L be a sublanguage of NL_{aASCP}. A *nominal unification problem* in L is a pair (Γ, ∇) where Γ is a finite set of equations $e \doteq e'$ with $e, e' \in L$ and ∇ is a finite set of freshness and DVC-constraints, where all expressions are in L. A ground substitution ρ is a *solution* of (Γ, ∇) iff ρ satisfies ∇ and $e\rho \sim_\alpha e'\rho$ for all $e \doteq e' \in \Gamma$.

A *unifier* for (Γ, ∇) is a pair (σ, ∇') in L, where σ is a substitution and ∇' is a set of constraints, such that ∇' is satisfiable and for every substitution γ such that $\sigma \circ \gamma$ is ground for Γ, ∇, ∇':

$$(\sigma \circ \gamma) \text{ satisfies } \nabla' \implies (\sigma \circ \gamma) \text{ is a solution for } (\Gamma, \nabla)$$

For a nominal unification problem (Γ, ∇) in L, a set M of unifiers is *complete*, iff for every solution ρ of (Γ, ∇), there is a unifier $(\sigma, \nabla') \in M$ such that there is a ground substitution γ with $A\sigma\gamma{=}\rho(A)$, $S\sigma\gamma \sim_\alpha \rho(S)$, $D\sigma\gamma \sim_\alpha \rho(D)$, and $P\sigma\gamma{=}\rho(P)$ for all atom-variables A, expression-variables S, context-variables D and permutation-variables P occurring in (Γ, ∇) (we say (σ, ∇') *covers* ρ). A unifier (σ, ∇') is a *most general unifier* of (Γ, ∇), iff $\{(\sigma, \nabla')\}$ is a complete set of unifiers for (Γ, ∇).

▶ **Theorem 2.11** ([36, 6, 18, 17]). *The nominal unification problem in NL_{aS}, with ∇ consisting of freshness constraints only, is solvable in quadratic time and is unitary: For a solvable nominal unification problem (Γ, ∇), there exists a most general unifier of the form (σ, ∇') which can be computed in polynomial time.*

In Theorem 6.1 we show that our unification algorithm solves the same problem where DVC-constraints are permitted, also in polynomial time.

▶ **Definition 2.12.** An *ASD1-unification problem* is a set of NL_{aASC}-equations, where context variables occur at most once, all top-expressions in equations have a DVC-constraint, and the equations have to be solved w.r.t. α-equivalence.

3 Preparations for NL_{aASC}-Unification

In this section we will analyze properties of contexts and expressions, where we will see that the use of DVC-constraints is of considerable help. This is a preparation for the unification rules for treating unification equations of the form $D_1[e_1] \doteq D_2[e_2]$ decomposing expressions with context-variables, and it is a natural requirement for overlaps of (the instances of) rules and transformations.

▶ **Lemma 3.1.** *Let e be an NL_a-expression. Then there is an expression e' that satisfies the* DVC *with $e \sim_\alpha e'$.*

Proof. The argument is a simple bottom-up algorithm that renames all bound atoms by fresh ones. ◀

▶ **Lemma 3.2.** *Let e be an NL_a-expression that satisfies the* DVC *and π be a permutation. Then $\pi{\cdot}e$ also satisfies the* DVC.

Proof. This holds, since a permutation is a bijection on the atoms. ◀

▶ **Lemma 3.3.** *Let e_1, e_2 be two expressions in NL_a that satisfy the* DVC *(separately). Then $e_1 \sim_\alpha e_2$ is equivalent to the condition that there exists a permutation π with $e_1 = \pi{\cdot}e_2$, where $supp(\pi) \subseteq (\mathcal{A}t(e_1) \cup \mathcal{A}t(e_2)) \setminus (FA(e_1) \cup FA(e_2))$.*

Proof. If e_1, e_2 satisfy the DVC and $e_1 = \pi{\cdot}e_2$ where π does not change free atoms of e_1, e_2, then clearly $e_1 \sim_\alpha e_2$. We prove the other direction of the claim by induction on the size.

- For constants and atoms, this is trivial, since π does not change free atoms.
- If $e_1 = \lambda a.e'_1$, and $e_2 = \lambda a.e'_2$, then $e'_1 \sim_\alpha e'_2$, hence $e'_1 = \pi{\cdot}e'_2$, for a (minimal) permutation π. Hence also $e_1 = \pi{\cdot}e_2$.
- Let $e_1 = \lambda a.e'_1$, and $e_2 = \lambda b.e'_2$, with $a \neq b$. Then $a\#e_2$, $a\#e'_2$, and $e'_1 \sim_\alpha (a\ b){\cdot}e'_2$. The expressions e'_1 and $(a\ b){\cdot}e'_2$ satisfy the DVC, hence by induction hypothesis, $e'_1 = \pi'{\cdot}(a\ b){\cdot}e'_2$, for a permutation π' where $\pi'(a) = a$. Let π be the permutation $\pi'{\cdot}(a\ b)$. Then $\pi'{\cdot}(a\ b){\cdot}b = a$. Hence $\pi{\cdot}e_2 = e_1$.
- If $e_1 = f\ e_{1,1} \dots e_{1,n}$, and $e_2 = f\ e_{2,1} \dots e'_{2,n}$, then by induction there are permutations π_i such that $\pi_i{\cdot}e_{2,i} = e_{1,i}$ for all i. Since the permutations can be chosen minimal and are only determined by the binders, and since the DVC is assumed, the permutations are disjoint. Thus we can compose (i.e. union) the permutations, and obtain $\pi = \pi_1 \dots \pi_n$ as the required permutation. ◀

▶ **Remark 3.4.** *We motivate the decomposition lemma. The inductive definition of $\sim_\alpha$ for abstractions that usually consists of two rules can be joined into a single rule that characterizes α-equivalence:*

$$\frac{a \# \lambda b.s_2, s_1 \sim_\alpha (a\ b){\cdot}s_2}{\lambda a.s_1 \sim_\alpha \lambda b.s_2}$$

where a may be equal to b or different. Generalizing this for arbitrary contexts results in the next, slightly more complicated situation where $C_1 = \lambda a_1 \dots \lambda a_n.[\cdot]$, $C_2 = \lambda b_1 \dots \lambda b_n.[\cdot]$, and the rule

$$\frac{CA(C_1) \# C_2[s_2], \exists \pi : (C_1 \sim_\alpha \pi{\cdot}C_2, s_1 \sim_\alpha \pi{\cdot}s_2)}{C_1[s_1] \sim_\alpha C_2[s_2]}$$

where π is a permutation with $\pi{\cdot}b_i = a_i$ for all i, $CA(C_1) = \{a_1, \dots, a_n\}$, and $CA(C_2) = \{b_1, \dots, b_n\}$. We assume that $a_i \neq a_j$, $b_i \neq b_j$ for $i \neq j$, but $a_i = b_j$ for some i, j may hold. We show below, that the latter rule is already the general one, provided the DVC holds for $C_1[s_1]$ and $C_2[s_2]$.

We now analyze the decomposition of context applications $C[e]$ under the assumption that the DVC holds. Note that contexts like $\lambda a.\lambda a.[\cdot]$ violate the DVC, and so are excluded. For an NL_a-context C, we denote with $CAO(C)$ the ordered tuple of the atoms in $CA(C)$, where the atom ordering is according to the nesting of active bindings: The outermost bound atom comes first. For example if $C = \lambda a.\lambda a.(f\ (\lambda b.\lambda c.b)\ (\lambda b.\lambda b.[\cdot]))$, then $CAO(C) = (a, b)$.

▶ **Proposition 3.5.** *Let C_1, C_2 be NL_a-contexts and e_1, e_2 be NL_a-expressions, such that C_1 and C_2 have identical hole positions, and such that $C_1[e_1]$ as well as $C_2[e_2]$ satisfy the DVC. Then the following are equivalent:*

1. *$C_1[e_1] \sim_\alpha C_2[e_2]$.*
2. *$\forall a \in CA(C_1)$: $a \# C_2[e_2]$ and there is a permutation π with $C_1 \sim_\alpha \pi{\cdot}C_2$ and $e_1 \sim_\alpha \pi{\cdot}e_2$, where π does not change free atoms in $C_2[e_2]$, π maps $CAO(C_2)$ to $CAO(C_1)$, and $supp(\pi) \subseteq CA(C_2) \cup CA(C_1)$.*

▶ **Example 3.6.** The DVC is required in Proposition 3.5: Let $C_1 = (a, \lambda a.[\cdot])$ and $C_2 = (a, \lambda b.[\cdot])$, which implies $C_1[a] = (a, \lambda a.a) \sim_\alpha (a, \lambda b.b) = C_2[b]$. The DVC is violated for the left expression. We see that there does not exist a (common) permutation π with $C_1 \sim_\alpha \pi{\cdot}C_2$ and $e_1 \sim_\alpha \pi{\cdot}e_2$.

Note that exploiting the general decomposition property of context-variables above in a unification algorithm, even under strong restrictions, requires permutation-variables, and also constraints of the form $CA(D)\,\#\,e$. There are investigations on nominal unification permitting variable permutations [7, 10], but an extension to context-variables is open. We will use further components in the constraint set that has NL_{aASCP} as its working language, which will refine the information.

▶ **Definition 3.7** (Further Constraint Components)**.** Let ρ be a ground substitution. The unification algorithm uses the following further constraint components:

- $A \neq e$, where e is an atom expression. (short form of $A\#e$)
- $CA(D)\#e$, which is satisfied by ρ, if for all atoms $a \in CA(D\rho)$: $a\#e\rho$.
- $supp(\pi)\#e$, which is satisfied by ρ, if for all atoms $a \in supp((\pi)\rho)$: $a\#e\rho$.
- $supp(\pi) \subseteq CA(C_1) \cup CA(C_2)$ which is satisfied by ρ, if for all atoms $a \in supp((\pi)\rho)$: $a \in CA(C_1\rho) \cup CA(C_2\rho)$.
- $C \neq \emptyset$, which is satisfied by ρ, if $C\rho$ is not the trivial context.

▶ **Proposition 3.8.** *In NL-languages containing expression-variables, freshness constraints can in linear time be encoded as* DVC*-constraints by translating $a\#e$ into* DVC$((f\ (\lambda a.a)\ S))$*, plus the equation $S \doteq e$; and $A\#e$ into* DVC$((f\ (\lambda A.A)\ S))$ *plus the equation $S \doteq e$ where f is a binary function symbol, and in both cases, S is a new expression-variable.*

Proof. We consider the first case $(a\,\#\,e)$. If the freshness constraint is satisfied by ρ, then a does not occur free in $e\rho$, hence the modulo-alpha-instantiation of S by ρ can be done without use of the atom a, and then the constraint is satisfied. To prove the converse, assume that the DVC-constraint is satisfied by ρ. Then $e\rho$ cannot have a free atom a. Similar arguments hold for the other case. ◀

4 The Unification Algorithm NomUnifyASC for NL_{aASC}

The nominal unification problem in the language NL_{aASCP} without any restrictions seems to be too hard (at least we did not find an algorithm to solve it). One hint is that already context unification for first-order terms is a quite hard problem. Its solvability was open for decades and recently shown to be in PSPACE [14]. That is why we restrict the input and allow only single occurrences of the same context-variable. A further complication are permutation-variables, since nominal unification with permutation-variables but without contexts, known as equivariant unification [7, 10], is known to be solvable in EXPTIME. If context-variables and permutations are combined, then it is unclear how to do constraint solving, since the (to be guessed) support of the permutations depends on the set of captured atoms occurring in the instance of context-variables, and it is unknown how to bound this number of atoms.

Thus, we describe a unification algorithm for ASD1-unification problems, where we permit additional freshness constraints.

Since double occurrences of context variables must be avoided during the execution of the algorithm, instantiations within the equations are not permitted: For example transitions using replacement starting with $S \doteq D[\ldots], S \doteq e_1, S \doteq e_2, \ldots$ may introduce two occurrences of D, since the result is $D[\ldots] \doteq e_1, D[\ldots] \doteq e_2, \ldots$. Thus we propose a Martelli-Montanari-style algorithm [21] that avoids instantiations within the unsolved equations.

$$\text{(flatten)}\ \frac{\Gamma \uplus \{C[e] \doteq e'\}}{\Gamma \uplus \{C[S] \doteq e'\} \uplus \{S \doteq e\}} \ \text{where } S \text{ is a fresh variable}$$

Figure 1 The flatten-rule

$$\text{(ME1)}\ \frac{(\Gamma \uplus \{e \doteq e \doteq M\}, \nabla, \theta)}{(\Gamma \uplus \{e \doteq M\}, \nabla, \theta)} \qquad \text{(ME2)}\ \frac{(\Gamma \uplus \{\{e\}\}, \nabla, \theta)}{(\Gamma, \nabla, \theta)}$$

$$\text{(ME3)}\ \frac{(\Gamma \uplus \{\pi_1{\cdot}S \doteq \pi_2{\cdot}V \doteq M\}, \nabla, \theta)}{(\Gamma\sigma \uplus \{\pi_2{\cdot}V \doteq M\sigma\}, \nabla\sigma, \theta \cup \sigma)} \ \text{if } S \neq V,\ V \text{ is an } S\text{- or } A\text{-variable or atom, and } \sigma = \{S \mapsto \pi_1^{-1}{\cdot}\pi_2{\cdot}V\}$$

$$\text{(ME4a)}\ \frac{(\Gamma \uplus \{\pi_1{\cdot}A \doteq \pi_2{\cdot}X \doteq M\}, \nabla, \theta)}{(\Gamma\sigma \uplus \{\pi_2{\cdot}X \doteq M\sigma\}, \nabla \cup \{A = \pi_1^{-1}{\cdot}\pi_2{\cdot}X\}, \theta \cup \sigma)} \ \text{if } X \neq A \text{ is an atom or atom-variable, and } \sigma = \{A \mapsto \pi_1^{-1}{\cdot}\pi_2{\cdot}X\}$$

$$\text{(ME4b)}\ \frac{(\Gamma \uplus \{\pi_1{\cdot}X_1 \doteq \pi_2{\cdot}X_2 \doteq M\}, \nabla, \theta)}{(\Gamma \uplus \{\pi_1{\cdot}X_1 \doteq M\}, \nabla \cup \{X_1 = \pi_1^{-1}{\cdot}\pi_2{\cdot}X_2\}, \theta)} \ \text{if } X_1, X_2 \text{ are atom-variables s.t. } X_1 = X_2, \text{ or } X_1, X_2 \text{ are atoms}$$

$$\text{(ME5)}\ \frac{(\Gamma \uplus \{\pi{\cdot}S \doteq e \doteq M\}, \nabla, \theta)}{(\Gamma \uplus \{e \doteq M\}, \nabla, \theta \cup \{S \mapsto \pi^{-1}{\cdot}e\})} \ \text{if } S \text{ does not occur in } M,\ e \text{ or } \Gamma$$

$$\text{(ME6)}\ \frac{(\Gamma \uplus \{\pi_1{\cdot}S \doteq M_1\} \uplus \{\pi_2{\cdot}S \doteq M_2\}, \nabla, \theta)}{(\Gamma \uplus \{S \doteq \pi_1^{-1}{\cdot}M_1 \doteq \pi_2^{-1}{\cdot}M_2\}, \nabla, \theta)}$$

$$\text{(FP)}\ \frac{(\Gamma \uplus \{S \doteq \pi{\cdot}S \doteq M\}, \nabla, \theta)}{(\Gamma \uplus \{S \doteq M\}, \nabla \uplus \{\mathit{supp}(\pi) \# S\}, \nabla, \theta)}$$

$$\text{(Df)}\ \frac{(\Gamma \uplus \{(f\ e_1 \ldots e_{ar(f)}) \doteq (f\ e'_1 \ldots e'_{ar(f)}) \doteq M\}, \nabla, \theta)}{(\Gamma \uplus \{(f\ e_1 \ldots e_{ar(f)}) \doteq M\} \uplus \{e_1 \doteq e'_1, \ldots, e_{ar(f)} \doteq e'_{ar(f)}\}, \nabla, \theta)}$$

$$\text{(Abstr)}\ \frac{(\Gamma \uplus \{\lambda X.e_1 \doteq M\}, \nabla, \theta)}{(\Gamma \uplus \{\lambda A'_1.e_1 \doteq M\}, \nabla \uplus \{A'_1 = X\}, \theta)} \ \text{if } X \text{ is of the form } \emptyset{\cdot}a,\ \pi{\cdot}a, \text{ or } \pi{\cdot}A \text{ and } \pi \text{ is not trivial, and } A'_1 \text{ is fresh}$$

$$\text{(Dlam)}\ \frac{(\Gamma \uplus \{\lambda A_1.e_1 \doteq \lambda A_2.e_2 \doteq M\}, \nabla, \theta)}{(\Gamma \uplus \{\lambda A_1.e_1 \doteq M, e_1 \doteq (A_1\ A_2){\cdot}e_2\}, \nabla \uplus \{A_1 \# \lambda A_2.e_2\}, \theta)}$$

Figure 2 Rules of NomUnifyASC for Variables and Decomposition

$$\text{(CD0)}\ \frac{(\Gamma \uplus \{e_1 \doteq (\pi{\cdot}D)[e_2] \doteq M\}, \nabla, \theta, \Delta)}{(\Gamma \uplus \{e_1 \doteq e_2 \doteq M\}, \nabla[[\cdot]/D], \theta \cup \{D \mapsto [\cdot]\}, \Delta)} \ \text{if } \mathit{tops}(e_1) \text{ is an atom variable, } \texttt{atom}, \text{ or } \mathit{tops}(e_1) = S,\ S \text{ occurs in } e_2;\ D \notin \Delta$$

$$\text{(CD1)}\ \frac{(\Gamma \uplus \{f\ e_1 \ldots e_n \doteq (\pi_1{\cdot}D_1)[e'_1] \doteq \ldots \doteq (\pi_m{\cdot}D_m)[e'_m]\}, \nabla, \theta, \Delta)}{\begin{array}{l}(\Gamma \uplus \{\{e_k\} \cup \{D_{i,1}[e'_i] \mid k = j(i), i \in \{1,..,m\}\} \mid k = 1, \ldots, n\},\\ \quad \nabla, \theta \cup \{D_i \mapsto \pi_i^{-1}{\cdot}(f\ e_1 \ldots \underbrace{D_{i,1}}_{j(i)} \ldots e_n) \mid i = 1, \ldots, m\}, \Delta)\end{array}}$$

if $\forall i : D_i \in \Delta$, and where context variables $D_{i,1}$ for $i = 1, \ldots, m$ are fresh and where for all $i = 1, \ldots, m$, the index position $j(i)$ of D_i is guessed.

$$\text{(CD2)}\ \frac{(\Gamma \uplus \{\lambda X.e_0 \doteq (\pi_1{\cdot}D_1)[e_1] \doteq \ldots \doteq (\pi_m{\cdot}D_m)[e_m]\}, \nabla, \theta, \Delta)}{\begin{array}{l}(\Gamma \uplus \{e_0 \doteq (X\ A_1){\cdot}(D_{1,1}[e_1]) \doteq \ldots \doteq (X\ A_m){\cdot}(D_{m,1}[e_m])\},\\ \quad \nabla \cup \{X \# \lambda A_i.D_{i,1}[e_i] \mid i = 1, \ldots, m\}, \theta \cup \{D_i \mapsto \pi_i^{-1}{\cdot}(\lambda A_i.D_{i,1})\}, \Delta)\end{array}}$$

if $\forall i : D_i \in \Delta$ and X is an atom or atom-variable and $A_i, D_{i,1}$ are fresh

Figure 3 Rules of NomUnifyASC for F-D-Decomposition

(DCPref) $$\frac{(\Gamma \mathbin{\dot\cup} \{(\pi_1{\cdot}D_1)[e_1] \doteq (\pi_2{\cdot}D_2)[e_2] \doteq \ldots \doteq (\pi_n{\cdot}D_n)[e_n]\}, \nabla, \theta, \Delta)}{\begin{array}{l}(\Gamma \mathbin{\dot\cup} \{e_1 \doteq P_2{\cdot}((\pi_2{\cdot}D_{2,1})[e_2]) \doteq \ldots \doteq P_n{\cdot}((\pi_n{\cdot}D_{n,1})[e_n])\},\\ \nabla \cup \{CA(D_1) \# \pi_1^{-1}{\cdot}((\pi_i{\cdot}D_i)[e_i]), i = 2, \ldots, n\}\\ \quad \cup\{supp(P_i) \subseteq (CAO(\pi_1{\cdot}D_1) \cup CAO(\pi_i{\cdot}D_{i,1})) \mid i = 2, \ldots, n\},\\ \theta \cup \{D_i \mapsto D_{i,1}D_{i,2}, D_{i,1} \mapsto \pi_i^{-1}{\cdot}P_i^{-1}{\cdot}\pi_1{\cdot}D_1 \mid i = 2, \ldots, n\},\\ \Delta \cup \{D_{i,1} \mid i = 2, \ldots, n\}) \text{ where } D_i \in \Delta, D_{i,1}, D_{i,2}, P_2, \ldots, P_n \text{ are fresh.}\end{array}}$$

(DCPRem) First apply (DCPRem*); then apply the guessing rules (GuessDEmpty), (GuessDNonEmpty) to $D_{j,1}$ for all j. If all $D_{j,1}$ are guessed to be nonempty, then the rule (DCFork) must be applied to the resulting multi-equation.

(DCFork) Apply (DCFork*), then remove all introduced variables $S_{i,j}$ using (ME5)

The helper rules (DCPRem*) and (DCFork*):

(DCPRem*) $$\frac{(\Gamma \mathbin{\dot\cup} \{(\pi_1{\cdot}D_1)[e_1] \doteq (\pi_2{\cdot}D_2)[e_2] \doteq \ldots \doteq (\pi_n{\cdot}D_n)[e_n]\}, \nabla, \theta, \Delta)}{\begin{array}{l}(\Gamma \mathbin{\dot\cup} \{(\pi_1{\cdot}D_{1,1})[e_1] \doteq P_2{\cdot}((\pi_2{\cdot}D_{2,1})[e_2]) \doteq \ldots \doteq P_n{\cdot}((\pi_n{\cdot}D_{n,1})[e_n])\},\\ \nabla \cup \{CA(D_{1,0}) \# (\pi_2{\cdot}D_2)[e_2], \ldots, CA(D_{n,0}) \# (\pi_n{\cdot}D_n)[e_n]\}\\ \quad \cup\{supp(P_i) \subseteq (CAO(D_{1,0}) \cup CAO(D_{i,0})) \mid i = 2, \ldots, n\},\\ \theta \cup \{D_1 \mapsto (\pi_1^{-1}{\cdot}D_{1,0})D_{1,1}, \ldots, D_n \mapsto (\pi_n^{-1}{\cdot}D_{n,0})D_{n,1},\\ \qquad D_{2,0} \mapsto P_2^{-1}{\cdot}D_{1,0}, \ldots, D_{n,0} \mapsto P_n^{-1}{\cdot}D_{1,0}\},\\ \Delta \cup \{D_{i,0} \mid i = 1, \ldots, n\}) \text{ where } P_i, D_{i,j} \text{ are fresh.}\end{array}} \quad \text{if } \forall i : D_i \in \Delta$$

(DCFork*) $$\frac{(\Gamma \mathbin{\dot\cup} \{(\pi_1{\cdot}D_1)[e_1] \doteq \ldots \doteq (\pi_n{\cdot}D_n)[e_n]\}, \nabla, \theta, \Delta)}{\begin{array}{l}(\Gamma \mathbin{\dot\cup} \{\{(\pi_i{\cdot}D'_i)[e_i] \mid i \in M_1\} \cup \{\pi_i{\cdot}S_{i,1} \mid i \notin M_1\}\} \mathbin{\dot\cup} \ldots \mathbin{\dot\cup}\\ \{\{(\pi_i{\cdot}D'_i)[e_i] \mid i \in M_m\} \cup \{\pi_i{\cdot}S_{i,m} \mid i \notin M_m\}\}, \nabla, \theta \cup \sigma, \Delta)\end{array}} \quad \text{if } \forall i\text{: } D_i \in \Delta$$

where f with $ar(f) \geq 2$ and the index positions $j(i)$ for $i{=}1, \ldots, n$ are guessed s.t. $|\{j(i) \mid 1 \leq i \leq n\}| \geq 2$; and $\sigma{=}\{D_i {\mapsto} (f\ S_{i,1} \ldots \underbrace{D'_i[\cdot]}_{j(i)} \ldots S_{i,m}) \mid 1{\leq}i{\leq}n\}$ and $D'_i, S_{i,i'}$ are fresh; and $M_k := \{h \mid j(h) = k\}$ for $k = 1, \ldots, m$.

Figure 4 Rules of NomUnifyASC for D-D-decomposition.

(GuessDEmpty) $$\frac{(\Gamma, \nabla, \theta, \Delta)}{(\Gamma[[\cdot]/D], \nabla[[\cdot]/D], \theta \cup \{D \mapsto [\cdot]\}, \Delta)} \quad \text{If } D \notin \Delta, D \text{ occurs in } \Gamma$$

(GuessDNonEmpty) $$\frac{(\Gamma, \nabla, \theta, \Delta)}{(\Gamma, \nabla, \theta, \Delta \cup \{D\})} \quad \text{If } D \notin \Delta, D \text{ occurs in } \Gamma$$

Figure 5 Rules of NomUnifyASC for guessing D empty or nonempty

(Clash) $\dfrac{(\Gamma \uplus \{e_1 \doteq e_2 \doteq M\}, \nabla, \theta)}{\mathit{Fail}}$ if $\mathit{tops}(e_1), \mathit{tops}(e_2)$ are different atoms; or $\mathit{tops}(e_1), \mathit{tops}(e_2)$ is `atom`, λ or a function symbol, and $\mathit{tops}(e_1) \neq \mathit{tops}(e_2)$; or $\mathit{tops}(e_1)$ is an atom or atom-variable, and $\mathit{tops}(e_2)$ is λ or $f \in \mathcal{F}$.

(Cycle) $\dfrac{(\Gamma \uplus \{S_1 \doteq e_1 \doteq M_1, \ldots, S_n \doteq e_n \doteq M_n\}, \nabla, \theta, \Delta)}{\mathit{Fail}}$
if all e_i are neither variables nor suspensions, all context variables occurring in e_i are in Δ, S_{i+1} occurs in e_i for $i = 1, \ldots, n-1$, and S_1 occurs in e_n

(CD0Fail) $\dfrac{(\Gamma \uplus \{e_1 \doteq (\pi{\cdot}D)[e_2] \doteq M\}, \nabla, \theta, \Delta)}{\mathit{Fail}}$ if $\mathit{tops}(e_1)$ is `atom` or an atom-variable, or $\mathit{tops}(e_1){=}S$ and S occurs in e_2; $D{\in}\Delta$

Figure 6 Failure Rules of NomUnifyASC

4.1 The Data Structures and Rules of NomUnifyASC

The state during unification is $(\Gamma, \nabla, \theta, \Delta)$ from NL_{aASCP}, where Γ is a set of sets of expressions, so-called multi-equations, ∇ is a set of freshness and DVC-constraints and further constraints of Definition 3.7, θ is a substitution in triangle-form[1], represented as a set of components, and Δ is a set of context-variables that are assumed to be nonempty. We explicitly write Δ if needed.

Multi-equations $M = \{e_1, \ldots, e_n\}$ will sometimes be written as $e_1 \doteq e_2 \doteq \ldots \doteq e_n$. We will assume that the expressions in Γ are flattened, using iteratively the rule (flatten) in Fig. 1, i.e., in $(f\ e_1 \ldots e_n), \lambda\pi{\cdot}X'.e$, and in $D[e]$, the expressions e_i, e are of the form $\pi{\cdot}X$, where X is an atom or expression-variable. For permutations, we assume that there is a compression scheme implementing sharing using an SLP [20] where a permutation is a composition (i.e. like a string) of the basic components $(a\ b), (A\ a), (a\ A), (A\ B), P, P^{-1}$, and the expansion of a representation may be exponentially long, and where the required operations on the permutations like composition and inverting can be done in polynomial time and space. However, to keep the presentation simple, we do not mention this compression and operations in the description of the unification rules.

▶ **Definition 4.1** (NomUnifyASC)**.** The input of the non-deterministic algorithm NomUnifyASC is an ASD1-unification problem (Γ, ∇), where Γ is a set of equations and ∇ a set of DVC- and freshness constraints, both over NL_{aASC}. The internal data structure is a tuple $(\Gamma, \nabla, \theta, \Delta)$, over NL_{aASCP}. The algorithm finishes either with Fail, or, if Γ is empty and no Failure rule applies, with a tuple $(\nabla', \theta, \Delta')$.

NomUnifyASC uses rules in rule sets: The rule (flatten) in Fig. 1 is applied until no longer applicable. Then the variable-replacement and usual decomposition rules in Fig. 2, the rules for decomposing multi-equations with expressions of the form $D[\ldots]$, and with function symbols or λ as top symbol in Fig. 3, the decomposition rules for multi-equations of expressions $D[\ldots]$ in Fig. 4, where the starred rules (DCPrem*) and (DCFork*) are not used directly; the rules for guessing context-variables as empty or nonempty in Fig. 5, and the failure rules in Fig. 6. Rules (CD1), (CD2) make one (parallel) decomposition step, where (CD1) first guesses a common first level of the hole positions of all D_i. (DCPref) guesses that one context is a prefix of the others; (DCPRem*) guesses a (maximal) common prefix of

[1] A *substitution in triangle-form* is a shared representation of a substitution, e.g., $\{x \mapsto (f\ y\ z), y \mapsto a, z \mapsto \lambda b.b\}$, is the substitution $\{x \mapsto f\ a\ (\lambda b.b), y \mapsto a, z \mapsto \lambda b.b\}$, i.e. the substitution itself is idempotent.

the contexts D_i, and (DCFork*) guesses that D_i fork and the first level of the hole positions of D_i.

The priorities of rule application are the sequence as above, i.e. the rules in Fig. 2, 3, 4, 5, 6. Within the rule sets, for rules in Figs. 2, 3, the priority is the sequence as given in the figures. For the rules in Figs. 4 and 5, within the rule sets the priority is the same. The failure rules can be applied at any time.

▶ **Example 4.2.** For $f\, a\, S_2\, S_1 \doteq f\, S_1\, (\lambda a.S_3)\, S_3$ there is a substitution that equates the expressions. However, if there are DVC-constraints for the top expressions, which must be satisfied by unifiers, i.e. $\{\mathrm{DVC}(f\, S_1\, S_2\, S_1), \mathrm{DVC}(f\, a\, (\lambda a.S_3)\, S_3)\} \subseteq \nabla$, then these cannot be satisfied, since for example, the instantiation $\rho(S_3) = a$ cannot be α-renamed. Another example is $f\, S\, S \doteq f\, (\lambda a.a)\, (\lambda b.b)$, which is solvable by $\{S \mapsto \lambda a.a\}$: it does not lead to a DVC-violation, since it is treated as instantiation modulo α.

4.2 The Algorithm for Satisfiability of the Output

We define the non-deterministic algorithm that checks satisfiability of the output constraints, where we give the justification later in Section 5. The algorithm exploits the execution sequence of NomUnifyASC, since without this information a decision algorithm appears to be impossible since permutation-variables as well as context-variables appear in the constraint. I.e., we were unable to find an algorithm checking solvability of arbitrary constraints permitted by the syntax, i.e. with context-variables and permutation-variables. Thus we have to use the knowledge on the execution trace of the algorithm in order to decide satisfiability of the final constraints.

▶ **Definition 4.3.** The algorithm NomFreshASC operates on the output (∇, θ, Δ) of NomUnifyASC and uses the set of all atoms and atom-variables occurring in the execution sequence H leading to this output, and the number d of context-variables in the input. It performs the following steps:

(I) Iteratively guess the solution of atom-variables, i.e. for an atom-variable A guess that A is mapped by the solution to an already used atom in H, or to a fresh one, and replace the atom-variable A accordingly in H. In the next iteration the fresh atom is among the used ones. Let H' be the adjusted execution sequence. Note that the exact names of fresh atoms are irrelevant. Thus there is only a linear number (w.r.t. the number of used atoms) of possibilities for every atom-variable. Let M_A be the set of all atoms in the execution sequence H'.

(II) Replace every expression-variable S that occurs in H' and that is not instantiated by θ, by a constant c from the signature.

(III) Construct M_∞ as a set of $|M_A| * (d!)^2$ atoms by extending the set M_A by further fresh atoms, where d is the number of context-variables in Γ. Guess for every context-variable D that occurs in Γ, ∇, θ and that is not instantiated by θ, the ordered set of captured atoms from the given set of atoms.
(Note that the number of potentially generated atoms is at most exponential, see Lemma 5.7).

(IV) Guess the permutation-variables as bijections on the set M_∞.

(V) Test the freshness constraints, equality, disequality, extended freshness constraints, and non-emptiness constraints, which are now immediately computable. To test whether the θ violates the DVC in Γ, use dynamic programming to compute the sets FA, BA for every expression- and context-variable, and $CA(D)$ for the context-variables D that are not instantiated by θ. Then test the DVC-property, which is possible in polynomial time.

5 Properties of NomUnifyASC and NomFreshASC

Let `maxArity` be the maximal arity of function symbols in the signature.

▶ **Lemma 5.1.** *The number of occurrences of any context-variable in Γ in every state of* NomUnifyASC *is at most 1.*

An example which shows that implicitly requiring the DVC is not stable, in contrast to requiring explicit DVC-constraints, is $\{S_1 \doteq f\ (\lambda a.a)\ (\lambda a.a)\}$. It does not satisfy the DVC, but after applying (flatten) to position 1 of the right hand side, we obtain $\{S_1 \doteq (f\ S\ (\lambda a.a)), S \doteq \lambda a.a\}$ which has the solution $\{S \mapsto \lambda a.a, S_1 \mapsto (f\ (\lambda a'.a')\ (\lambda a.a))$. If the initial set ∇ contains DVC$(f\ (\lambda a.a)\ (\lambda a.a))$, then there is no solution before and after flattening, since (flatten) cannot be applied to expressions within ∇.

▶ **Lemma 5.2.** *If the input is (Γ, ∇), then the application of (flatten) to a subexpression of e of the equations Γ does not change the set of solutions.*

Proof. Obvious, since $\sim_\alpha$ is a congruence. ◀

▶ **Proposition 5.3.** *For an NL_{aASC}-input Γ, ∇, the non-deterministic algorithm* NomUnifyASC *terminates after a polynomial number of steps.*

Proof. First we define the measure:

- $\mu_1 := \mu_{1,1} + 2 * \texttt{maxArity} * \mu_{1,2}$, where $\mu_{1,1}$ is the number of expressions in Γ, and $\mu_{1,2}$ is the number of occurrences of function symbols and λ-s in Γ;
- μ_2 is $\mu_{1,1}$ minus the number of multi-equations. Note that $0 \leq \mu_2 \leq \mu_1$.
- μ_3 is the pair consisting of the number of occurrences of context-variables D in Γ and the number of context-variables D occurring in Γ which are not in Δ.
- μ_4 is the number of occurrences of expressions λX, where X is of the forms $\emptyset{\cdot}a$, $\pi{\cdot}a$, or $\pi{\cdot}A$ and where π is nontrivial.

The following table lists the relation between Γ before and after the application of the rule or subalgorithm.

rule	μ_1	μ_2	μ_3
MEi	>		
FP	>		
Df	>		

rule	μ_1	μ_2	μ_3	μ_4
Abstr	=	=	=	>
Dlam	>			
CD0	=	=	>	

rule	μ_1	μ_2	μ_3
CD1	>		
CD2	>		
GD(N)E	=	=	>

rule	μ_1	μ_2	μ_3
DCPref	=	=	>
DCPRem	=	>	
DCFork	=	>	

It can be verified for (MEi), (FP), (Df), (Abstr), (Dlam), and (CD0) by a simple check. It is correct for (CD1) and (CD2), since the expression $f\ e_1 \ldots e_m$ is removed, which counts 2*`maxArity`. (DCPref) removes one occurrence of a context-variable. (DCFork) splits the context-directions, and first introduces expression-variables S_i, which are then removed. Hence the number of expressions in multi-equations is the same, but there are more multi-equations. For (DCPRem), it suffices to check (DCFork). The measure μ_3 is not increased by any rule, and $\mu_2 \leq \mu_1$, and $\mu_4 \leq \mu_1$, hence the number of executions of rules is polynomial. Since there are multiple sub-steps, we have to argue that a single rule application can be done in polynomial time. The number of steps within the rules is polynomial due to the strict decrease w.r.t. the orderings. We add a (standard, SLP) compression scheme using sharing, which implies that all operations on the permutations like flattening, inverting and composition can be done in polynomial time and space. ◀

The assumptions on the DVC-constraints ensure that for every solution ρ and input expression e in Γ, the instance $e\rho$ satisfies the distinct variable condition.

▶ **Lemma 5.4.** *If all top expressions of the initial set of equations Γ are restricted by the* DVC*, then this also holds for all top-expressions in the equations in the sequence of rule executions of the algorithm* NomUnifyASC.

▶ **Proposition 5.5.** *Inspecting the details of all rules of* NomUnifyASC *shows soundness: The solutions of the final data structure are also solutions of the input. The following rules of the algorithm* NomUnifyASC *do not lose any solutions, i.e. for every solution ρ of the data structure Q before application, there is a solution ρ' of the output data structure Q', such that $\rho(X) \sim_\alpha \rho'(X)$ for all atom-, expression-, context- and permutation-variables X occurring in Q: These are rules in Fig. 2, and rules (CD0), (CD2) from Fig. 3. In addition the failures rules do not lose any solutions, since they are only applicable, if there are no solutions.*

▶ **Proposition 5.6.** *Assume that for all top-expressions e in the input Γ, there is a constraint* DVC(e) *in ∇. Then the algorithm* NomUnifyASC *is complete: If ρ is a solution of the intermediate data structure Q, Γ is not empty and no Failure rule applies, then there is a possible rule application, such that there is solution ρ' of the output Q', and $\rho(X) \sim \rho'(X)$ for all atom-, expression-, context- and permutation-variables X occurring in Q.*

We now consider the correctness and complexity of NomFreshASC.

▶ **Lemma 5.7.** *Let H be an execution of* NomUnifyASC *starting with $S_0 := (\Gamma_0, \nabla_0, \theta_0, \Delta_0)$ where $\Gamma_0 = \Gamma$, context variables occur at most once, and θ_0, Δ_0 are trivial or empty. Let the sequence H end with $S_{out} = (\emptyset, \nabla_{out}, \theta_{out}, \Delta_{out})$, and let ρ be a solution of the input as well as of the output S_{out}. Then there is also a solution ρ' that uses only a set of atom VA_∞ with $|VA_\infty| \leq |VA| * ((d!)^2)$, where the visible set VA of atoms $VA = \{a \mid a \text{ occurs in } H\} \cup \{A\rho \mid A \text{ occurs in } H\}$, and where d is the number of context-variables in Γ.*

▶ **Proposition 5.8.** *Let $(\nabla_{out}, \theta_{out}, \Delta_{out})$ be the output of* NomUnifyASC *for input (Γ, ∇). The algorithm* NomFreshASC *decides satisfiability of the output in NEXPTIME in the size of the input, where the main components are $|(\Gamma, \nabla)| * ((d!)^2)$, where d is the number of context-variables in Γ. For a fixed upper bound on the number of context-variables, satisfiability can be checked in NP time.*

Proof. This mainly follows from Lemma 5.7. ◀

▶ Remark. There is a complexity jump between freshness constraints and DVC-constraints in the language NL_{aSC}, since satisfiability of freshness constraints in NL_{aSC} is in PTIME whereas satisfiability of DVC-constraints in NL_{aSC} is NP-hard.

Combining Propositions 5.6, 5.3, and 5.8 shows:

▶ **Theorem 5.9.** *For Γ, ∇ as input the algorithm* NomUnifyASC *terminates and is sound and complete. The computation of some output $(\nabla', \theta', \Delta')$ can be done in NP-time, and the collecting version of the algorithm produces at most exponentially many outputs $(\nabla', \theta', \Delta')$. Decidability of solvability of output constraints, and hence of the input, is in NEXPTIME, and if the number of context-variables is fixed, then in NP time.*

▶ **Theorem 5.10.** *Solvability of ASD1-unification problems is in NEXPTIME.*

6 Specializations, Applications and Examples

We consider nominal unification in NL_{aS} with freshness and DVC-constraints extending the result of [36] (see Theorem 2.11) by allowing DVC-constraints and by restricting NomUnifyASC

▶ **Theorem 6.1.** *The nominal unification problem in NL_{aS} where freshness- and* DVC*-constraints are permitted in ∇ is solvable in polynomial time. Moreover, for solvable (Γ, ∇), there exists a most general unifier of the form (∇', θ) which can be computed in polynomial time, i.e., the problem class is unitary.*

The application of NomUnifyASC to NL_{AS} also yields at most one most general unifier, however, the complexity to check solvability is increased, since already the solvability of freshness constraints in NL_{AS} is NP-hard [32].

We now consider applications and examples. Most reduction and transformation rules in the application domain of functional programming languages require freshness and/or DVC-constraints to exclude invalid instances of the rules.

▶ **Example 6.2.** We provide rules that can be described by our language. The rule $\texttt{app}\ (\lambda A.S)\ S' \to \texttt{let}\ A = S'\ \texttt{in}\ S$ is a sharing-variant of β-reduction. It needs the constraint $\textsc{dvc}(\texttt{app}\ (\lambda A.S)\ S')$ if `let` is recursive, to ensure that for the instances, there are no free occurrence of $A\rho$ in $S'\rho$.

The rule $\texttt{let}\ A = S\ \texttt{in}\ D[A] \to \texttt{let}\ A = S\ \texttt{in}\ D[S]$ copies a single expression to some position of the bound atom. The expression is represented by the expression-variable S, and the target position by the context-variable D. The constraint $\textsc{dvc}(\texttt{let}\ A = S\ \texttt{in}\ D[A])$ prevents capturing in instances, s.t. $A\rho$ is not captured by $D\rho$. The constraint $\textsc{dvc}(\texttt{let}\ A = S\ \texttt{in}\ D[S])$ prevents that $D\rho$ captures atoms that are free in $S\rho$. Since instances $S\rho$ are α-renamed in $(\texttt{let}\ A = S\ \texttt{in}\ D[S])\rho$, these constraints suffice.

▶ **Example 6.3.** We describe an exemplary unification problem that occurs in correctness proofs of program transformations. A reduction rule in the let-calculus of [2] is $\texttt{let}\,A_x{=}(\texttt{let}\,A_y{=}S_y\,\texttt{in}\,S_x)\,\texttt{in}\,S_r \to \texttt{let}\,A_y{=}S_y\,\texttt{in}\,\texttt{let}\,A_x{=}\,S_x\,\texttt{in}\,S_r$ with DVC-constraint $\textsc{dvc}(\texttt{let}\,A_x{=}(\texttt{let}\,A_y{=}S_y\,\texttt{in}\,S_x)\,\texttt{in}\,S_r)$ that prevents the occurrence of A_y as free atom in S_r, and thus an unwanted capture in the right hand side. To check whether there is a (preferably nontrivial) overlap of the left hand side of the rule with itself (as a transformation) we form the unification equation $\texttt{let}\,A_x{=}(\texttt{let}\,A_y{=}S_y\,\texttt{in}\,S_x)\,\texttt{in}\,S_r \doteq D[\texttt{let}\,A'_x{=}(\texttt{let}\,A'_y{=}S'_y\,\texttt{in}\,S'_x)\,\texttt{in}\,S'_r]$ where the context-variable D is intended as a representation of the reduction strategy[2]. For correct application, the constraints $\textsc{dvc}(\texttt{let}\,A_x{=}(\texttt{let}\,A_y{=}S_y\,\texttt{in}\,S_x)\,\texttt{in}\,S_r)$ and $\textsc{dvc}(D[\texttt{let}\,A'_x{=}(\texttt{let}\,A'_y{=}S'_y\,\texttt{in}\,S'_x)\,\texttt{in}\,S'_r])$ are required. We omit the case that $D \mapsto [\cdot]$ and analyze the instantiation $D \mapsto (\texttt{let}\,A_x{=}D_1\,\texttt{in}\,S_r)$. We obtain the equation $\texttt{let}\,A_y{=}S_y\,\texttt{in}\,S_x \doteq D_1[\texttt{let}\,A'_x{=}(\texttt{let}\,A'_y{=}S'_y\,\texttt{in}\,S'_x)\,\texttt{in}\,S']$. Guessing $D_1 \mapsto [\cdot]$ results in $\texttt{let}\,A_y{=}S_y\,\texttt{in}\,S_x \doteq \texttt{let}\,A'_x{=}(\texttt{let}\,A'_y{=}S'_y\,\texttt{in}\,S'_x)\,\texttt{in}\,S'_r$. Guessing $A_y \mapsto A'_x$, we obtain as solution $S_y \doteq (\texttt{let}\,A'_y{=}S'_y\,\texttt{in}\,S'_x)$, $S_x \doteq S'_r$. If we alternatively guess $A_y \neq A'_x$, we obtain as solution $S_y \doteq (\texttt{let}\,A'_y{=}S'_y\,\texttt{in}\,S'_x)$ and $S_x \doteq (A_x\ A'_y){\cdot}S'_r$ together with the constraint $A_y \# S'_r$.

▶ **Example 6.4.** An example with two occurrences of an S-variable is the copy rule [2, 23, 33] $\texttt{let}\,A_x{=}\lambda A_y.S\,\texttt{in}\,D[A_x] \to \texttt{let}\,A_x{=}\lambda A_y.S\,\texttt{in}\,D[\lambda A_z.(A_y\ A_z){\cdot}S]$ with $\textsc{dvc}(\texttt{let}\,A_x{=}\lambda A_y.S\,\texttt{in}\,D[A_x])$ and $\textsc{dvc}(\texttt{let}\,A_x{=}\lambda A_y.S\,\texttt{in}\,D[\lambda A_z.(A_y\ A_z){\cdot}S])$ as constraints, which imply that $A_y\rho$ is not captured in $D\rho$, and that $A_z\#S$ is valid. The diagram proof technique [33] also needs to overlap right-hand sides of program transformations with left hand sides of reduction rules. We overlap the right hand side and the left hand side of the copy rule at the top position. The problem (Γ, ∇) in NL_{ASC} with $\Gamma = \{D_0[\texttt{let}\,A_{x,1}{=}\lambda A_{y,1}.S_1\,\texttt{in}\,D_1[\lambda A_{z,1}.(A_{y,1}\ A_{z,1}){\cdot}S_1]]$

[2] In general, the reduction strategy has to be represented by context classes as in [31].

$\doteq$ $\texttt{let}\, A_{x,2}{=}\lambda A_{y,2}.S_2\,\texttt{in}\,D_2[A_{x,2}]\}$ and $\textsc{dvc}(\texttt{let}\,A_{x,2}{=}\lambda A_{y,2}.S_2\,\texttt{in}\,D_2[A_{x,2}])\}$ as well as $\textsc{dvc}(D_0[\texttt{let}\,A_{x,1}{=}\lambda A_{y,1}.S_1\,\texttt{in}\,D_1[\lambda A_{z,1}.(A_{y,1}\;A_{z,1})\cdot S_1]]$ contained in ∇ represents this overlap. Execution of the rules of NomUnifyASC leads to $D_0 \mapsto [\cdot]$, $A_{x,1}{=}A_{x,2}$, and $\lambda A_{y,1}.S_1 \doteq \lambda A_{y,2}.S_2$, $D_1[\lambda A_{z,1}.(A_{y,1}\;A_{z,1})\cdot S_1] \doteq D_2[A_{x,2}]$ as equations. Assume that the next guesses are $A_{y,1}{=}A_{y,2}$ leading to $S_1{=}S_2$. For the last equation there are 4 possibilities for the relative position of the holes of D_1, D_2: If the hole paths in the instances of D_1 and D_2 fork then a solution (using rule (DCFork)) comes with the mappings $D_1 \mapsto D_3[\texttt{app}\,D_4\,D_5[A_{x,2}]]$ and $D_2 \mapsto D_3[\texttt{app}\,D_4[\lambda A_{z,1}.(A_{y,1}\;A_{z,1})\cdot S_1]\,D_5]$. The possibility that the hole-path of $D_2\rho$ is a prefix of the hole-path $D_1\rho$ can be ruled out since $A_{x,2}\rho$ is an atom. Further forms of knowledge may be built into the unification mechanism which would require an extension of the formalism like adding grammars for context classes.

7 Conclusion and Further Work

We described and analyzed a nominal unification algorithm for a language with higher-order expressions and variables for atoms, expressions and contexts.

Further work is to extend and adapt the unification and constraint solution method to more constructs of higher-order languages, like a recursive-let, or context-classes.

References

1 Andreas Abel and Brigitte Pientka. Higher-order dynamic pattern unification for dependent types and records. In C.-H. Luke Ong, editor, *Proc. 10th TLCA 2011*, volume 6690 of *LNCS*, pages 10–26. Springer, 2011. `doi:10.1007/978-3-642-21691-6_5`.

2 Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. A call-by-need lambda calculus. In *Proceedings of POPL 1995*, pages 233–246, San Francisco, CA, 1995. ACM Press.

3 Mauricio Ayala-Rincón, Maribel Fernández, Murdoch James Gabbay, and Ana Cristina Rocha Oliveira. Checking overlaps of nominal rewriting rules. *Electr. Notes Theor. Comput. Sci.*, 323:39–56, 2016.

4 Mauricio Ayala-Rincón, Maribel Fernández, and Daniele Nantes-Sobrinho. Nominal narrowing. In Delia Kesner and Brigitte Pientka, editors, *Proc. FSCD 2016*, volume 52, pages 11:1–11:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl.

5 H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics.* North-Holland, Amsterdam, New York, 1984.

6 Christophe Calvès and Maribel Fernández. A polynomial nominal unification algorithm. *Theor. Comput. Sci.*, 403(2-3):285–306, 2008.

7 James Cheney. The complexity of equivariant unification. In *Proceedings of ICALP 2004*, volume 3142 of *LNCS*, pages 332–344. Springer-Verlag, 2004.

8 James Cheney. *Nominal Logic Programming.* PhD thesis, Cornell University, Ithaca, NY, August 2004.

9 James Cheney. Relating higher-order pattern unification and nominal unification. In *Proceedings of UNIF 2005*, pages 104–119, 2005.

10 James Cheney. Equivariant unification. *JAR*, 45(3):267–300, 2010.

11 Nicolaas Govert de Bruijn. Lambda calculus notation with nameless dummies: A tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

12 Maribel Fernández and Albert Rubio. Nominal completion for rewrite systems with binders. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Proc.*

39th ICALP Part II, volume 7392 of *LNCS*, pages 201–213. Springer, 2012. doi:10.1007/978-3-642-31585-5_21.

13 Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1):27–57, 1975. doi:10.1016/0304-3975(75)90011-0.

14 Artur Jez. Context unification is in PSPACE. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Proceedings of ICALP 2014 Part II*, volume 8573 of *LNCS*, pages 244–255. Springer, 2014.

15 Yunus Kutz and Manfred Schmidt-Schauß. Most general unifiers in generalized nominal unification. In *Informal Proceedings of UNIF 2017*, 2017.

16 Matthew R. Lakin. Constraint solving in non-permutative nominal abstract syntax. *Logical Methods in Computer Science*, 7(3), 2011.

17 Jordi Levy and Mateu Villaret. An efficient nominal unification algorithm. In Christopher Lynch, editor, *Proc. RTA 2010*, volume 6 of *LIPIcs*, pages 209–226. Schloss Dagstuhl, 2010.

18 Jordi Levy and Mateu Villaret. Nominal unification from a higher-order perspective. *ACM Trans. Comput. Log.*, 13(2):10, 2012. doi:10.1145/2159531.2159532.

19 Tomer Libal and Dale Miller. Functions-as-constructors higher-order unification. In Delia Kesner and Brigitte Pientka, editors, *Proceedings of FSCD 2016*, volume 52 of *LIPIcs*, pages 26:1–26:17. Schloss Dagstuhl, 2016. doi:10.4230/LIPIcs.FSCD.2016.26.

20 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.

21 Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

22 Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991. doi:10.1093/logcom/1.4.497.

23 Andrew K. D. Moran, David Sands, and Magnus Carlsson. Erratic fudgets: A semantic theory for an embedded coordination language. In *Proceedings of Coordination 1999*, volume 1594 of *LNCS*, pages 85–102. Springer-Verlag, 1999.

24 Tobias Nipkow. Functional unification of higher-order patterns. In *Proceedings of LICS 1993*, pages 64–74. IEEE Computer Society, 1993. doi:10.1109/LICS.1993.287599.

25 Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In Richard L. Wexelblat, editor, *Proceedings of PLDI 1988*, pages 199–208. ACM, 1988.

26 Andrew Pitts. Nominal techniques. *ACM SIGLOG News*, 3(1):57–72, 2016. doi:10.1145/2893582.2893594.

27 Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, New York, NY, USA, 2013.

28 Zhenyu Qian. Linear unification of higher-order patterns. In *Proceedings of TAPSOFT 1993*, pages 391–405. Springer-Verlag, 1993. URL: http://dl.acm.org/citation.cfm?id=646618.697260.

29 David Sabel. Alpha-renaming of higher-order meta-expressions. In Brigitte Pientka and Wim Vanhoof, editors, *Proceedings of PPDP 2017*, pages 151–162, New York, NY, USA, 2017. ACM. doi:10.1145/3131851.3131866.

30 Manfred Schmidt-Schauß, Temur Kutsia, Jordi Levy, and Mateu Villaret. Nominal unification of higher order expressions with recursive let. In Manuel V. Hermenegildo and Pedro López-García, editors, *Proc. LOPSTR 2016*, volume 10184 of *LNCS*, pages 328–344. Springer, 2016.

31 Manfred Schmidt-Schauß and David Sabel. Unification of program expressions with recursive bindings. In German Vidal, editor, *Proceedings of PPDP 2016*, pages 160–173, New York, NY, USA, 2016. ACM.

32 Manfred Schmidt-Schauß, David Sabel, and Yunus Kutz. Nominal unification with atom-variables. *J. Symbolic Comput.*, 2017. accepted for publication.

33 Manfred Schmidt-Schauß, Marko Schütz, and David Sabel. Safety of Nöcker's strictness analysis. *J. Funct. Programming*, 18(04):503–551, 2008. `doi:10.1017/S0956796807006624`.

34 Christian Urban. Nominal techniques in Isabelle/HOL. *J. Autom. Reasoning*, 40(4):327–356, 2008. `doi:10.1007/s10817-008-9097-2`.

35 Christian Urban and Cezary Kaliszyk. General bindings and alpha-equivalence in nominal Isabelle. *Log. Methods Comput. Sci.*, 8(2), 2012.

36 Christian Urban, Andrew M. Pitts, and Murdoch Gabbay. Nominal unification. In *Proc. CSL 2003, EACSL 2003, and KGC 2003*, volume 2803 of *LNCS*, pages 513–527. Springer, 2003.

37 Joe B. Wells, Detlef Plump, and Fairouz Kamareddine. Diagrams for meaning preservation. In Robert Nieuwenhuis, editor, *Proceedings of RTA 2003*, volume 2706 of *LNCS*, pages 88 –106. Springer, 2003.

A Detailed Proofs

A.1 Multi-Contexts

▶ **Definition A.1.** We introduce n-contexts for some $n \geq 1$ (also called multi-contexts) in NL_a. These are expressions of NL_a extended by constants (holes) $[\cdot]_i, i = 1, \dots, n$, where every hole occurs at most once. For $n \geq 1$ and two n-contexts C_1, C_2 the relation $C_1 \sim_\alpha C_2$ holds, iff for all n-tuples $a_1, \dots, a_n$ of (perhaps equal) atoms a_i, it holds that $C_1[a_1, \dots, a_n] \sim_\alpha C_2[a_1, \dots, a_n]$ as expressions, which is a generalization from contexts (Definition 2.4) to multi-contexts.

The following lemma helps in the completeness proof of rule (DCFork).

▶ **Lemma A.2.** *Let $n \geq 1$, C_1, C_2 be NL_a-n-contexts, and $e_{1,i}, e_{2,i}, i = 1, \dots, n$ be NL_a-expressions, such that the corresponding hole positions of C_1 and C_2 are identical, and such that $C_1[e_{1,1}, \dots, e_{1,n}]$ and $C_2[e_{2,1} \dots, e_{2,n}]$ satisfy the* DVC. *Then $C_1[e_{1,1}, \dots, e_{1,n}] \sim_\alpha C_2[e_{2,1}, \dots, e_{2,n}]$ iff the following holds:*

$CA(C_1) \,\#\, C_2[e_{2,1} \dots, e_{2,n}]$, and there is a permutation π that does not change free atoms in $C_2[e_{2,1} \dots, e_{2,n}]$ with $C_1 \sim_\alpha \pi{\cdot}C_2$ and $e_{1,i} \sim_\alpha \pi{\cdot}e_{2,i}$ for all i, π maps $CAO(C_2)$ to $CAO(C_1)$, and $supp(\pi) \subseteq CAO(C_1) \cup CAO(C_2)$.

A.2 Properties of DVC-Constraints

▶ **Lemma 3.3.** *Let e_1, e_2 be two expressions in NL_a that satisfy the* DVC *(separately). Then $e_1 \sim_\alpha e_2$ is equivalent to the condition that there exists a permutation π with $e_1 = \pi{\cdot}e_2$, where $supp(\pi) \subseteq (\mathcal{A}t(e_1) \cup \mathcal{A}t(e_2)) \setminus (FA(e_1) \cup FA(e_2))$.*

Proof. If e_1, e_2 satisfy the DVC and $e_1 = \pi{\cdot}e_2$ where π does not change free atoms of e_1, e_2, then clearly $e_1 \sim_\alpha e_2$. We prove the other direction of the claim by induction on the size. For constants and atoms, this is trivial, since π does not change free atoms. If $e_1{=}\lambda a.e_1'$, and $e_2{=}\lambda a.e_2'$, then $e_1' \sim_\alpha e_2'$, hence $e_1' = \pi{\cdot}e_2'$, for a (minimal) permutation π. Hence $e_1 = \pi{\cdot}e_2$. Let $e_1 = \lambda a.e_1'$, and $e_2 = \lambda b.e_2'$, with $a \neq b$. Then $a \# e_2$, $a \# e_2'$, and $e_1' \sim_\alpha (a\ b){\cdot}e_2'$. The expressions e_1' and $(a\ b){\cdot}e_2'$ satisfy the DVC, by induction hypothesis, $e_1' = \pi'{\cdot}(a\ b){\cdot}e_2'$, for a permutation π' where $\pi'(a) = a$. Let π be the permutation $\pi'{\cdot}(a\ b)$. Then $\pi'{\cdot}(a\ b){\cdot}b = a$. Hence $\pi{\cdot}e_2 = e_1$. If $e_1 = f\ e_{1,1} \dots e_{1,n}$, and $e_2 = f\ e_{2,1} \dots e_{2,n}'$, then by induction there are permutations π_i such that $\pi_i{\cdot}e_{2,i} = e_{1,i}$ for all i. Since the permutations can be chosen minimal and are only determined by the binders, and since the DVC is assumed, the permutations are disjoint. Thus we can compose (i.e. union) the permutations, and obtain $\pi = \pi_1 \dots \pi_n$ as the required permutation. ◀

A.3 Decomposition of Context-Applications

▶ **Proposition 3.5.** *Let C_1, C_2 be NL_a-contexts and e_1, e_2 be NL_a-expressions, such that C_1 and C_2 have identical hole positions, and such that $C_1[e_1]$ as well as $C_2[e_2]$ satisfy the* DVC. *Then the following are equivalent:*

1. *$C_1[e_1] \sim_\alpha C_2[e_2]$.*
2. *$\forall a \in CA(C_1)$: $a \,\#\, C_2[e_2]$ and there is a permutation π with $C_1 \sim_\alpha \pi{\cdot}C_2$ and $e_1 \sim_\alpha \pi{\cdot}e_2$, where π does not change free atoms in $C_2[e_2]$, π maps $CAO(C_2)$ to $CAO(C_1)$, and $supp(\pi) \subseteq CAO(C_2) \cup CAO(C_1)$.*

Proof. We show " $\implies$ " by induction on the length of the hole path of C_1.

- If the length is 0, then the claim is trivial.
- If $C_1 = f\ e_1 \ldots \underbrace{C_1'}_{k} \ldots e_n$ then $C_2 = f\ e_1' \ldots \underbrace{C_2'}_{k} \ldots e_n'$. The capture condition holds, since $CA(C_i') = CA(C_i)$. The assumption and the congruence property of $\sim_\alpha$ imply $e_i \sim_\alpha e_i'$ for all $i \neq k$. By the induction hypothesis there is a permutation π_k satisfying the theorem, which is the required permutation.
- If $C_1 = \lambda a.C_1'$ and $C_2 = \lambda a.C_2'$, then the capture condition holds, $C_1'[e_1]$ and $C_2'[e_2]$ are α-equivalent, and we can apply the induction hypothesis.
- If $C_1 = \lambda a.C_1'$ and $C_2 = \lambda b.C_2'$, then $a\#C_2'[e_2]$, and $C_1'[e_1] \sim_\alpha (a\ b){\cdot}C_2'[e_2]$. The DVC also holds for $(a\ b){\cdot}C_2'$, hence we can apply the induction hypothesis, and obtain $C_1' \sim_\alpha \pi'{\cdot}(a\ b){\cdot}C_2'$, and $e_1 \sim_\alpha \pi'{\cdot}(a\ b){\cdot}e_2$, and $\forall c \in CA(C_1')$: $c\#(a\ b){\cdot}C_2'[e_2]$. The equation $c = a$ is not possible, since $C_1[\cdot]$ satisfies the DVC; $c = b$ may be possible, but since $a\#C_2'[e_2]$, and due to the application of $(a\ b)$ there are no free occurrences of b in $(a\ b){\cdot}C_2'[e_2]$. This implies $\forall c \in CA(C_1)$: $c\#C_2[e_2]$. The application $\pi'{\cdot}(a\ b){\cdot}b$ results in a, since π' does not change a. Hence the required permutation is $\pi = \pi'{\cdot}(a\ b)$.

The reverse is easy: if there are two expressions $e_1 \sim_\alpha e_2$, $C_1[e_1], C_2[e_2]$ satisfy the DVC, and there is a permutation π that does not change free atoms in e_2, and $C_1 \sim_\alpha \pi{\cdot}C_2$, then $C_1[e_1] \sim_\alpha C_2[e_2]$. ◀

A.4 Completeness of NomUnifyASC

▶ **Proposition 5.6.** *Assume that for all top-expressions e in the input Γ, there is a constraint* DVC(e) *in ∇. Then the algorithm* NomUnifyASC *is complete: If ρ is a solution of the intermediate data structure Q, Γ is not empty and no Failure rule applies, then there is a possible rule application, such that there is solution ρ' of the resulting data structure Q', and $\rho(X) \sim \rho'(X)$ for all atom-, expression-, context- and permutation-variables X occurring in Q.*

Proof. Let ρ be a solution of the current state. We scan the cases:

1. We can assume that all multi-equations have at least two expressions since otherwise rule (ME1) is applicable.
2. We can also assume that all context-variables that occur in Γ are contained in Δ, by applying either (CD0), or one of the rules (GuessDEmpty) or (GuessDNonEmpty), where the choice is directed by the solution.
3. If there is a multi-equation that has only context-variables as top symbols, then one of the rules from Fig. 4 is applicable, depending on the solution ρ, and there is a solution ρ' of the output that extends ρ. The condition that top-expressions in the input satsify the DVC and that the input are ASD1-unification problems is necessary for the application of Proposition 3.5.
4. If there is a multi-equation such that all but one expression have context-variables as top symbols, then there are several possibilities: Since $D \in \Delta$ for all D, one of the rules from Fig. 3 is applicable, since either the context-variables' instances have a common prefix or not. Proposition 3.5 and the knowledge on permutations show that the execution of the rules is possible. In any case, there will be a solution ρ' after the application that is an extension of ρ (on the variables of Q).
5. For the other cases there are at least two expressions in the multi-equation, which do not have a context-variable as top-symbol. The failure rules are not applicable, since otherwise, there is no solution. If the top symbols of two expressions are λ, or function symbols, then rules (Df), (Abstr) or (Dlam) can be applied and there is a solution ρ'

extending ρ after the application. If there are two expressions of the form $\pi{\cdot}X$, where X is an atom or expression-variable, then (ME3), (ME4a), (ME4b), or (ME5) is applicable, and there is a solution ρ' extending ρ after the application. Similar for the case where $tops(.)$ yields `atom` twice. If one expression is of the form $\pi{\cdot}X$, and the other is not of this form, then we apply (ME5) if this is possible.

6. The final case is that for all equations, the equation pattern permits only applications of rule (ME5), all multi-equations have only two expressions, but the conditions on non-containment of S in rule (ME5) prevent this. Defining a quasi-ordering on the expression-variables generated by the containment ordering over equations shows that there are no finite instance-expressions for some expression-variable, which is impossible, since we have assumed that there are solutions. Indeed in this case a failure rule would apply. ◀

A.5 Satisfiability of Constraints of NomUnifyASC

▶ **Lemma 5.7.** *Let H be a sequence of executions of* NomUnifyASC *starting with $S_0 := (\Gamma_0, \nabla_0, \theta_0, \Delta_0)$ where $\Gamma_0 = \Gamma$, the condition on the input are as stated in Definition 4.3, and θ_0, Δ_0 are trivial or empty. Let the sequence H end with $S_{out} = (\emptyset, \nabla_{out}, \theta_{out}, \Delta_{out})$, and let ρ be a solution of the input as well as of the output S_{out}. Then there is also a solution ρ' that uses only a set of atom VA_∞ with $|VA_\infty| \leq |VA| * ((d!)^2)$, where the visible set VA of atoms $VA = \{a \mid a \text{ occurs in } H\} \cup \{A\rho \mid A \text{ occurs in } H\}$, and where d is the number of context-variables in Γ.*

Proof. We show a stronger claim by induction on the steps of the execution: Let $\mathcal{D}_F$ and $\mathcal{D}_P$ be the set of context-variables D_1 in the applications of (DCPref) in H and $D_{1,0}$ in the application of (DCPRem) in H, and P_i, $i = 2, \ldots, n$ in the applications of (DCPref) and (DCprem), resp. Let us call these the *focused* context-variables and the *focused* permutation-variables in the respective rule applications. These are the set of context-variables that are moved to the codomain of θ. The permutation-variables are exactly all the generated ones in H. Let $VA = \{a \mid a \text{ occurs in } H\} \cup \{A\rho \mid A \text{ occurs in } H\}$. Then the size of VA is polynomial, since the execution sequence H can be generated in polynomial time according to Proposition 5.3. The claim is: there is a solution ρ' that uses only the set VA_∞ of atoms, where $VA_0{=}VA$, and VA_i is constructed below, such that $|VA_{i+1}| \leq |VA_i|{*}d^2$, where d is the number of context-variables in the input, and where $|VA_{i+1}|{>}|VA_i|$ only if rule (DCPref) or (DCprem) was executed. The construction implies $VA_i \subseteq VA_{i+1}$. The final VA_∞ is defined as the final VA_i.

We define the construction: Let i be an index in H and $S_i = (\Gamma_i, \nabla_i, \theta_i, \Delta_i)$ be a state in H with set VA_i, such that the next step is (DCPref) or (DCprem) leading to S_{i+1}. Let us assume that it is the first occasion in H such that a focussed context-variable or permutation-variable that is in the focus of a rule application (DCPref) or (DCPrem), uses an atom a' in its instances under ρ, where $a' \not\in VA_i$. Then the following changes are made to the solution ρ, resulting in ρ' and a modified execution sequence H'.

- First consider the modification concerning the rule (DCPref):
  - Let us consider the instances $(CAO(\pi_1{\cdot}D_1)\rho)$, $(CAO(P_2{\cdot}\pi_2{\cdot}D_{2,1})\rho),\ldots$, $(CAO(P_n{\cdot}\pi_n{\cdot}D_{n,1})\rho)$, (same notation as in the rule application). We can assume that $(\pi_1{\cdot}D_1)\rho$, $CAO(P_2{\cdot}\pi_2{\cdot}D_{2,1}\rho),\ldots$, $CAO(P_n{\cdot}\pi_n{\cdot}D_{n,1}\rho)$ only consist of $\lambda a_{k,1}, \ldots, a_{k,m}.[\cdot]$ where $m = |CAO(\pi_1{\cdot}D_1)|$, by modifying the solution ρ, which is without effect on the further execution of the algorithm NomUnifyASC and

solvability. We show that the number of binders can be bounded: Luckily, we can also eliminate the binder at position j, if $a_{k,j}$ is not in VA_i for all k: eliminate the binder j in every context above, then modify the instantiation of the permutation-variables P_k such that these map exactly $(\pi_i{\cdot}D_{i,1})\rho$ to $CAO(\pi_1{\cdot}D_1)$ for $k \geq 2$, which is justified by Proposition 3.5. Hence P_k does not need any extra fresh atoms in its support. Using the thus modified ρ, we replace all atoms (as expressions) by the constant c at the following positions: If the atom at this position in the instance $e_i\rho$ is an atom $a_{k,j}$ for some k.

- Since binders cannot be repeated, an upper bound on the maximal number of binders for a single context-variable is $|V_i| * d'$, where d' is the number of context-variables in Γ_i. The number of all used atoms in the instances is at most $|V_i| * d' * d'$.
- Let ρ' be ρ after these modifications. The ground substitution ρ' is a solution of the state S_{i+1}, and such that the same execution still leads to a final state that covers ρ'. There is no effect on the execution of the rules of the algorithm, since the changes are only in the solution.

- Now consider the modification for the rule (DCPrem). It is similar to the previous case, but we detail it, since the names of variables are different.
  - Consider the instances $(CAO(\pi_1{\cdot}D_{1,0})\rho)$, …, $(CAO(\pi_n{\cdot}D_{n,0})\rho)$. We can assume that $(\pi_k{\cdot}D_{k,0})\rho$ only consist of $\lambda a_{k,1}, \ldots, a_{k,m}.[\cdot]$ where $m = |CAO(D_{1,0})|$, by modifying the solution ρ, which is without effect on the execution of the algorithm NomUnifyASC and solvability. We can also eliminate the binder at position j, if $a_{k,j}$ is not in VA_i for all k, as follows: eliminate the binder j in every $(\pi_k{\cdot}D_{k,0})\rho$, then modify the instantiation of the permutation-variables P_k such that these map exactly $CAO(D_{k,0})$ to $CAO(D_{1,0})$ for $k \geq 2$. Using the thus modified ρ, we replace atoms by the constant c at all the following positions: If the atom at this position in the instance $e_i\rho$ is an atom with erased binder: $a_{k,j}$ for some k.
  - An upper bound on the maximal number of binders for a single context-variable is $|V_i|{*}d'{*}d'$ where d' is the number of context-variables in Γ_i.
  - Let ρ' be ρ after these modifications. The ground substitution ρ' is a solution of the state S_{i+1}, and such that the same execution still leads to a final state that covers ρ'.

The number of executions of rules (DCPref), (DCPrem) is at most the number of different context-variables. This holds, since (DCPref) removes one context-variable, and since (DCPRem) calls (DCFork), and (DCFork) can be applied also at most as often as there are expressions with topmost context-variables. As additional argument, all other rules keep the number of context-variables, and there is never a merge of two multisets that only contain context-variables. The estimation for the maximal number of variables is that in $(CAO(\pi_1{\cdot}D_1)\rho)$, there can at most be $d_i * |V_i|$ variables, where d_i is the number of context-variables in Γ_i. Since there is an iterated multiplication, we obtain $|VA|{*}d{*}d{*}(d{-}1){*}(d{-}1)\ldots$, which leads to the estimation as claimed.

This change can be iterated until there are no (DCPref), (DCPrem)-steps having an index j where completely fresh variables are used for all context-variables in the multi-equation. Finally, we have constructed VA_∞, and the modified solution ρ'. ◀

A.6 The Algorithm NomUnifyASC for Restricted Input

▶ **Theorem 6.1.** *The nominal unification problem in NL_{aS} where freshness- and* DVC-*constraints are permitted in ∇, is solvable in polynomial time. Moreover, for a solvable* *nominal unification problem* (Γ, ∇)*, there exists a most general unifier of the form* (∇', θ) *which can be computed in polynomial time.*

Proof. We assume that the algorithm computes in polynomial time a unifier (∇', θ) consisting of a substitution θ and a constraint set ∇', where in addition we assume that the output substitution θ is represented in triangle-form and that it is of polynomial size (see [32] for the technique). Soundness and completeness of computing only a single execution path follows from Proposition 5.5 since there are no permutation-variables, and no context-variables.

For the final satisfiability test, we instantiate the expression-variables in the codomain of θ with a constant from the signature. Note that also $\lambda a.a$ could be used if there is no such constant. For expression-variables S, it is possible to compute $FA(S\theta)$ in polynomial time using dynamic programming. The bound atoms in $FA(S\theta)$ are irrelevant, since these will be renamed by the substitution process which is done modulo α. Then the check for every constraint $\textsc{dvc}(e)$, whether $e\theta$ satisfies the $\textsc{dvc}$, can be performed in polynomial time. ◀

A.7 Expressivity of Freshness vs. DVC-Constraints

We show that there is a complexity jump between freshness constraints and $\textsc{dvc}$-constraints in the language NL_{aSC}.

▶ **Proposition A.3.** *Satisfiability of a set* ∇ *of freshness constraints in* NL_{aSC} *can be decided in PTIME.*

Proof. For deciding satisfiability, we apply the following transformations on ∇:

$$\begin{array}{llll}
\{a \,\#\, b\} \uplus \nabla & \to \nabla & \{a \,\#\, f\ e_1 \ldots e_n\} \uplus \nabla & \to \{a \,\#\, e_1, \ldots, a \,\#\, e_n\} \cup \nabla \\
\{a \,\#\, \lambda b.e\} \uplus \nabla & \to \{a \,\#\, e\} \cup \nabla & \{a \,\#\, \lambda a.e\} \uplus \nabla & \to \nabla \\
\{a \,\#\, \pi{\cdot}S\} \uplus \nabla & \to \nabla & \{a \,\#\, \pi{\cdot}D[e]\} \uplus \nabla & \to \nabla
\end{array}$$

For most of the transformations $\nabla \to \nabla'$ satisfiability of ∇' obviously implies satisfiability of ∇. Hence, we only treat the exceptional cases. For the constraints $a \,\#\, \pi{\cdot}S$, removal is correct, since S can be instantiated by a fresh atom, and for constraints $a \,\#(\pi{\cdot}D)[e]$ removal is correct, since the set of conditions for a single D-variable is of the form $\{a_1 \,\#(\pi_1{\cdot}D)[e_1], \ldots, a_n \,\#(\pi_n{\cdot}D)[e_n]\}$ which can always be satisfied by instantiating D with $\lambda a'_1 \ldots a'_n.[\cdot]$ where $a'_i = \pi_i(a_i)$.

After exhaustively applying the transformation the constraint is either empty (and thus satisfiability is detected), or it contains a constraint $a\#a$ and unsatisfiability is detected. ◀

▶ **Proposition A.4.** *Satisfiability of a set of freshness and DVC-constraints in* NL_{aSC} *is NP-hard.*

Proof. To show NP-hardness we reduce 3SAT to the satisfiability problem of freshness and non-capture constraints in NL_{aSC}. Let $\mathcal{C}$ be a set of clauses. We generate a set of freshness and DVC-constraints $T(\mathcal{C})$ as follows: Let us assume that $\texttt{True}, \texttt{False}$ are fixed atoms. For every propositional variable p_j occurring in $\mathcal{C}$, there is a context-variable D_j. The idea is that the instances of D_j are mainly $\lambda\texttt{True}.[\cdot]$ or $\lambda\texttt{False}.[\cdot]$. For each D_j, we add a DVC-constraint $\textsc{dvc}(f\ D_j(c)\ (\texttt{True False}){\cdot}D_j(c))$ to $T(\mathcal{C})$, which ensures that the instance of D_j does not capture both $\texttt{True}$ and $\texttt{False}$.

For every clause $\{L_1, L_2, L_3\} \in \mathcal{C}$, the set $T(\mathcal{C})$ contains a freshness constraint $\texttt{True} \,\#\, C_1[C_2[C_3[\texttt{True}]]]$ where $C_i := D_j$ if $L_i = p_j$ and $C_i := \pi{\cdot}D_j$ if $L_i = \neg p_j$ where $\pi = (\texttt{True False})$. E.g., the clause $\{\neg p_1, p_2, \neg p_3\}$ is encoded as $\texttt{True} \,\#\, (\pi{\cdot}D_1)[D_2[(\pi{\cdot}D_3)[\texttt{True}]]]$.

We show that satisfiability of the input clause set is equivalent to satisfiability of the constructed constraint set:

If $\mathcal{C}$ is satisfiable, then there exists a valuation v which maps each variable p_i to a truth value, s.t. in each clause at least one literal becomes `True` under the valuation v. We set $\rho(D_i) := \lambda v(p_i).[\cdot]$. We have to show that the ground substitution ρ satisfies all constraints in $T(\mathcal{C})$: For the DVC-constraints we get $\rho(f\ D_j(c)\ (\texttt{True False}){\cdot}D_j(c)) = f\ (\lambda v(p_i).c)\ (\lambda\overline{v(p_i)}.c)$ (where $\overline{L}$ negates a literal). Thus the DVC-constraints are satisfied by ρ. For each freshness constraint $\texttt{True} \# C_1[C_2[C_3[\texttt{True}]]]$ corresponding to clause $\{L_1, L_2, L_3\}$, we have $FA(\rho(C_1[C_2[C_3[\texttt{True}]]])) = \{\texttt{True}\} \setminus \{CA(\rho(C_i)) \mid 1 \leq i \leq 3\} = \{\texttt{True}\} \setminus \{v(L_i) \mid 1 \leq i \leq 3\} = \{\texttt{True}\} \setminus \{\texttt{True}\} = \emptyset$ since $v(L_i) = \texttt{True}$ must hold for at least one L_i for $1 \leq i \leq 3$. Thus the freshness constraints $\texttt{True} \# C_1[C_2[C_3[\texttt{True}]]]$ hold.

For the other part, assume that $T(\mathcal{C})$ is satisfiable and ρ is a ground substitution satisfying $T(\mathcal{C})$. The DVC-constraints ensure that $CA(\rho(D_j))$ cannot contain both `True` and `False`, since otherwise binders in $\rho(D_j)$ and in $\rho((\texttt{True False}){\cdot}D_j)$ cannot be pairwise disjoint. Since ρ satisfies the freshness constraints, we have $\texttt{True} \not\in FA(\rho(C_1[C_2[C_3[\texttt{True}]]])$ for all C_1, C_2, C_3 corresponding to clause $\{L_1, L_2, L_3\}$, which implies that $\texttt{True} \in CA(\rho(C_i))$ for some $1 \leq i \leq 3$. Thus, for those i, we have $w \in CA(\rho(D_i))$ where $w = \texttt{True}$ if $L_i = p_j$ and $w = \texttt{False}$ if $L_i = \neg p_j$. This shows that each valuation which sets $v(p_j) := w$ for all such p_j satisfies the clause $\{L_1, L_2, L_3\}$ and thus is a model of $\mathcal{C}$. ◀

Frank-59