Nominal Unification with Atom and Context Variables

Manfred Schmidt-Schauß
Goethe-University Frankfurt, Germany
schauss@ki.cs.uni-frankfurt.de
https://orcid.org/0000-0001-8809-7385

David Sabel
Goethe-University Frankfurt, Germany
sabel@ki.cs.uni-frankfurt.de
https://orcid.org/0000-0002-5109-3273

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 NomUnifyASD. We show that NomUnifyASD is sound and complete for this 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, NomUnifyASD runs in polynomial time, is unitary, and extends the classical problem by permitting distinct-variable constraints.

2012 ACM Subject Classification Theory of computation → Automated reasoning

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

Digital Object Identifier 10.4230/LIPIcs.FSCD.2018.28

Related Version A full version of the paper is available at [32], http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:hebis:30:3-452767.

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

---

1 Supported by the Deutsche Forschungsgemeinschaft (DFG) under grant SCHM 986/11-1.

2 Supported by the Deutsche Forschungsgemeinschaft (DFG) under grant SA2908/3-1.
Nominal Unification with Atom and Context Variables

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 $\alpha$-equivalence in quadratic time and outputs at most a single most general unifier [37, 6, 17].

Our intended application is the diagram-method, which is a syntactic proof method (e.g. [38, 12, 34]) to show properties like correctness of program transformations. As an example consider the reduction rule $(\text{cp})$ in the call-by-need lambda-calculus with let (see e.g. [2, 23, 34])

$$(\text{let } x = \lambda y. S \text{ in } C[x]) \rightarrow (\text{let } x = \lambda y. S \text{ in } C[\lambda y. S])$$

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

$$\text{let } A_1 = \lambda A_2. S_1 \text{ in } D_1[\lambda A_2. S_1] \triangleright \text{let } A_3 = \lambda A_4. S_2 \text{ in } D_2[A_3]$$

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] which in turn requires to add a further renaming to the rule to fulfill it (see Example 2.5 for details).

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), $\ell$ is a left hand side of a reduction rule, $C$ is an arbitrary context, and $\ell'$ 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 $R[\ell] \equiv C[\ell']$ can be attacked by using nominal unification in the extended language. However, for the general unification problem without any restrictions we did not find an algorithm to solve it, since it seems to be too hard (we discuss the problems in Sect. 4). As a consequence, we consider a subproblem and thus our presented algorithm solves nominal unification problems that are restricted to be linear w.r.t. context variables, so-called permutation variables do not occur in the input, and we require DVC-constraints for all equations in the input. From the perspective of applications to reasoning in calculi, these restrictions are (almost) optimal, since linearity of contexts in general holds, the DVC is also an often used assumption, and permutation variables are not required.

Results. A sound and complete algorithm NomUnifyASD for nominal unification of ASD1-unification problems is constructed. The algorithm NomUnifyASD computes in NP time a solution including a constraint (Theorem 5.8) and the collecting version produces at most exponentially many outputs. The algorithm NomFreshASD that checks satisfiability of the constraints of a solution runs in NEXPTIME (Proposition 5.7), hence solvability of ASD1-unification problems can be decided in NEXPTIME (Theorem 5.9). 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 NomUnifyASD 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 3.6). 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 3.2, Proposition 3.3, and Example 3.4).

A corollary is that classical nominal unification (with expression-variables only) is generalized by replacing freshness constraints by DVC-constraints such that unitarity and polynomial complexity still hold (Theorem 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 [37, 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 [35, 36].

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 [33, 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 NomUnifyASD. 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.

A proposal to the automated computation of overlaps and diagrams is in [31] where the unification problems are solved w.r.t. syntactic equality. This approach 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].

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 NomUnifyASD is introduced, which consists of a set of (non-deterministic) rules, and also the constraint checking algorithm NomFreshASD. In Sect. 5 the properties of the algorithms are analyzed. In Sect. 6 the special case NLaS is reconsidered, and we illustrate how the unification algorithm operates on examples. We conclude in Sect. 7. Due to space constraints, some of the proofs are omitted, but given in [32].

2 Nominal Languages and Nominal Unification

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 \( \geq 2 \). Let \( \mathcal{At} \) be the set of atoms ranged over by \( a, b \); \( \mathcal{A} \) be the set of atom-variables ranged over by
A, B, S be the set of expression-variables ranged over by S, T standing for expressions; D be the set of context-variables ranged over by D standing for single hole-contexts; and P be the set of permutation-variables ranged over by P standing for finite permutations on At, i.e. bijections π on At such that their support supp(π) = \{a ∈ At | π(a) ≠ 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 (ab).

**Definition 2.1.** The syntax of NLₐASDP is defined by the grammar

\[
X ∈ \mathcal{A}E \quad ::= \quad a \mid A \mid π.A \\
e ∈ \mathcal{E} \quad ::= \quad X \mid S \mid π.S \mid (f e_1 \ldots e_{ar(f)}) \mid λX.e \mid C[e] \\
C ∈ \mathcal{C} \quad ::= \quad [1] \mid D \mid π.D \mid (f e_1 \ldots e_n) \mid λX.[.] \mid C_1[C_2] \\
π \quad ::= \quad Φ \mid (A_1 A_2) • π \mid (A a) • π \mid (a A) • π \mid (a_1 a_2) • π \mid P • π \mid P^{-1} • π
\]

with the categories \(\mathcal{A}E\) 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 \((π_1 • A_1 π_2 • A_2)\) 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 NLₐASDP** are denoted by NLₐ₁, where \(M\) is a substring of aASDP, and the grammar is restricted accordingly. The mainly used languages are NLₐ as the ground language for the solutions, NLₐASDP as the expression language for the input, and NLₐASDP as the working language inside of the unification algorithm.

A substitution \(σ : NLₐASDP → NLₐASDP\) maps atom-variables to atom expressions, expression-variables to expressions, context-variables to contexts, permutation-variables to permutations. We identify a substitution with its extension to expressions. A ground substitution \(ρ\) is a substitution \(ρ : NLₐASDP → NLₐ\). We use permutation application • as operator and syntactic symbol, we use \(^{-1}\) as a syntactic symbol in \(P^{-1}\) and operator for inversion, and we abbreviate \(Φ V\) by \(V\) for a variable \(V\). We use the following operations and simplifications

\[
(π_1 • π_2)[e] → (π_1 • (π_2 • e)) \quad (π_1 • π_2)^{-1} → π_2^{-1} • π_1^{-1} \quad π[.] → [ ] \\
π • (f e_1 \ldots e_n) → (f • π • e_1 \ldots π • e_n) \quad π • (λX.e) → λX.π • e \quad π • C[e] → (π • C)[π • e] \\
(X_1 X_2)^{-1} → (X_1 X_2) \quad (C_1 C_2)[e] → C_1[C_2[e]]
\]

which permit standardizations: in NLₐ all permutation operations can be removed; in NLₐS, permutations can be represented as lists of swappings of length at most \(n − 1\) where \(n\) is the number of used atoms; and in NLₐASDP, the permutation operations only lead to suspensions of the form \(π • A\) and \(π • S\), \(π • D\). In all languages, permutations can be represented as a composition of lists of swappings, permutation-variables \(P\) and inverses \(P^{-1}\), and context expressions can be simplified to the form \((π • D)[e]\).

Let tops\(e\) be the top symbol of \(e\) after simplification of permutation applications, i.e. tops\(a = \text{atom}\), tops\((X • e) = \lambda\), tops\((f e_1 \ldots e_n) = f\), tops\((π • A) = A\), tops\((π • S) = S\), and tops\((π • D)[e]\) = D. For an expression \(e\) or context \(C\) in NLₐ, we denote with FA\(e\) or
FA(C), resp., the set of free atoms, and with At(e) or At(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 NLa, α-equivalence \( \sim_{\alpha} \) is the closure by reflexivity and congruence and the rule 
\[ a \not\in FA(e') \land e \sim_{\alpha} a b \Rightarrow \lambda a. e \sim_{\alpha} \lambda b. e' \]
We also use α-equivalence for contexts: \( NL_{\alpha} \) 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., \( \lambda a. [\_] \not\sim_{\alpha} \lambda b. [\_] \), \( \lambda a. \lambda b. [\_] \sim_{\alpha} \lambda b. \lambda a. [\_] \), but \( \lambda a. \lambda b. [\_] \not\sim_{\alpha} \lambda a. \lambda a. [\_] \). For \( C_1 \sim_{\alpha} C_2 \), it suffices if \( C_1[a] \sim_{\alpha} C_2[e] \) 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 \( (f a \lambda a. []) \not\sim_{\alpha} (f a \lambda b. []) \) and \( a \not\sim_{\alpha} b \).

We explain instantiation modulo \( \alpha \) for correctly defining solvability under dvc-restrictions.

Definition 2.2 (Instantiation modulo \( \alpha \)). For testing solvability of equations, we assume that ground substitutions map into \( NL_\alpha / \sim_{\alpha} \). An equivalent method is that whenever a ground substitution \( \rho \) is applied to a variable \( S \) or \( D \), we use an \( \alpha \)-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 \( \alpha \)), and where comparison is done modulo \( \sim_{\alpha} \).

The following definition explains free/bound variables and the satisfiability of dvc-constraints of expressions in \( NL_{\alpha} / \sim_{\alpha} \) without using fresh atoms.

Definition 2.3. Let \( e \) be a normalized \( NL_{\alpha ASDP} \)-expression and \( \rho \) be a ground substitution mapping into \( NL_{\alpha} / \sim_{\alpha} \), such that \( e\rho \) is an \( NL_{\alpha} / \sim_{\alpha} \)-expression. Then we define the bound atoms \( BA(e, \rho) \), and satisfiability of the dvc of \( (e, \rho) \) as follows, where the bound atoms introduced by \( \rho \) are ignored.

1. If \( X \) is an atom \( a \) or a suspension \( \pi \). Then \( BA(X, \rho) = \emptyset \), and the dvc is satisfied.
2. \( BA(\pi \cdot S, \rho) = \emptyset \), and the dvc is satisfied.
3. \( BA(\mu e_1, \rho) = \bigcup_{1=1}^n BA(e_i, \rho) \). 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 \( (e_i, \rho) \).
4. \( BA(\pi \cdot X, \rho) = BA(e, \rho) \cup \{X\rho\} \). The dvc is satisfied, if it is satisfied for \( (e, \rho) \), and if \( X\rho \not\in BA(e, \rho) \).
5. \( BA((\pi \cdot D)[e], \rho) = BA(e, \rho) \cup ((\pi)e\rho) \cdot CA(D\rho) \). The dvc is satisfied, if it is satisfied for \( D \), i.e. \( CA(D\rho) \cap FA(D\rho) = \emptyset \) as well as \( (BA(e, \rho) \cap ((\pi)e\rho) \cdot FA(D\rho) \cup CA(D\rho)) = \emptyset \), and the dvc is satisfied for \( (e, \rho) \).

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. We use well-known freshness constraints and novel dvc-constraints.

Definition 2.4 (Freshness and dvc-Constraints). Freshness constraints in \( NL_{\alpha ASDP} \) are of the form \( a \# e \) and \( A \# e \), and dvc-constraints in \( NL_{\alpha ASDP} \) are of the form \( dvc(e, \rho) \), where \( e \) is an \( NL_{\alpha ASDP} \)-expression.

A ground substitution \( \rho \) satisfies \( A \# e \) iff \( \rho(A) \not\in FA(e\rho) \); and \( \rho \) satisfies \( a \# e \) iff \( a \not\in FA(e\rho) \). The distinct variable condition (dvc) holds for an \( NL_{\alpha} \)-expression \( e \), if all bound atoms in \( e \) are distinct, and all free atoms in \( e \) are distinct from all bound atoms in \( e \). For a ground substitution \( \rho \), \( dvc(e, \rho) \) is satisfied, iff \( (e, \rho) \) satisfies the dvc (Definition 2.3). This is equivalent to \( e\rho \) (using instantiation modulo \( \alpha \)) satisfying the dvc.
If a ground substitution \( \rho \) satisfies all constraints of a set of freshness and/or DVC-constraints, then we say that \( \rho \) is a solution of the constraint set. A set of constraints is satisfiable if there is a solution. For example, \( f \ a \ \lambda b.(g \ a \ .\ c.c) \) satisfies the DVC and \( f \ b \ \lambda b.b \) violates it. With \( \rho = \{S \mapsto \lambda c.c\} \) we have \( f(\lambda a.S)(\lambda b.S) = f(\lambda a.\lambda c_1.c_1)(\lambda b.\lambda c_2.c_2) \) and \( \rho \) satisfies DVC\( f(\lambda a.S)(\lambda b.S) \). As another example, \( \rho' = \{S \mapsto \lambda c.b\} \) violates DVC\( f(\lambda a.S)(\lambda b.S) \).

**Example 2.5.** Consider a lambda-calculus with let as in [2]. A reduction rule of the corresponding calculus is \( \text{let } x = \lambda y.s \text{ in } C[x] \rightarrow \text{let } x = \lambda y.s \text{ in } C[\lambda y.s] \) where \( C \) is a context. We represent the expressions as \( (\text{let } (\lambda A_s.D[A_s]) \ (\lambda A_y.S)) \) and \( (\text{let } (\lambda A_s.D[\lambda A_y.S]) \ (\lambda A_y.S)) \). However, \( D \) must not capture the atom represented by \( A_x \), nor free atoms from \( S \), and \( A_x \) must not occur free in \( S \). Both conditions can be captured by the constraint that \( \text{let } (\lambda A_s.D[A_s]) \ (\lambda A_y.S) \text{ and let } (\lambda A_s.D[\lambda A_y.S]) \ (\lambda A_y.S) \) 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 \( \text{let } (\lambda A_s.D[\lambda A_x.s]) \ (\lambda A_y.S) \rightarrow \text{let } (\lambda A_s.D[\lambda A_x.s[\lambda A_x.s]]) \ (\lambda A_y.S) \) and \( A_x \neq S \). Now the DVC-constraints for both expressions make sense and produce the correct conditions.

**Definition 2.6.** Let \( L \) be a sublanguage of \( \text{NL}_{a\text{ASD}} \). A nominal unification problem in \( L \) is a pair \( (\Gamma, \nabla) \) where \( \Gamma \) is a finite set of equations \( e \equiv e' \) with \( e, e' \in L \) and \( \nabla \) is a finite set of freshness and DVC-constraints, where all expressions are in \( L \). A ground substitution \( \rho \) is a solution of \( (\Gamma, \nabla) \) if for every \( (\sigma, \nabla') \) in \( (\Gamma, \nabla) \), \( \rho \) satisfies \( \nabla \) and \( e \rho \sim_{\alpha} e' \rho \) for all \( e \equiv e' \in \Gamma \). A unifier for \( (\Gamma, \nabla) \) is a pair \( (\sigma, \nabla') \) in \( (\Gamma, \nabla) \), where \( \sigma \) is a substitution and \( \nabla' \) is a set of constraints, such that \( \nabla' \) is satisfiable and for every substitution \( \gamma \) such that \( \sigma \circ \gamma \) is ground for \( \Gamma, \nabla, \nabla' \), the following implication holds: \( (\sigma \circ \gamma) \) satisfies \( \nabla' \) \( \implies \) \( (\sigma \circ \gamma) \) is a solution for \( (\Gamma, \nabla) \).

A set \( M \) of unifiers is complete, iff for every solution \( \rho \) of \( (\Gamma, \nabla) \), there is a unifier \( (\sigma, \nabla') \in M \) such that there is a ground substitution \( \gamma \) 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 variables \( A, S, D, P \) occurring in \( (\Gamma, \nabla) \) (we say \( (\sigma, \nabla') \) covers \( \rho \)). A unifier \( (\sigma, \nabla') \) is a most general unifier of \( (\Gamma, \nabla) \), iff \( \{ (\sigma, \nabla') \} \) is a complete set of unifiers for \( (\Gamma, \nabla) \).

**Theorem 2.7** ([37, 6, 18, 17]). The nominal unification problem in \( \text{NL}_{a\text{S}} \), with \( \nabla \) consisting of freshness constraints only, is solvable in quadratic time and is unitary: For a solvable nominal unification problem \( (\Gamma, \nabla) \), there exists a most general unifier of the form \( (\sigma, \nabla') \) which can be computed in polynomial time.

## 3 Preparations for \( \text{NL}_{a\text{ASD}} \)-Unification

As a preparation for the unification rules treating equations of the form \( D_1[e_1] \equiv D_2[e_2] \), we analyze properties of contexts and expressions in this section. Clearly, for every \( \text{NL}_{a} \)-expression \( e \) there is some \( e' \) with \( e \sim_{\alpha} e' \) s.t. \( e' \) satisfies the DVC. If \( e \) satisfies the DVC, then \( \pi \cdot e \) also satisfies the DVC for any permutation \( \pi \).

**Lemma 3.1.** Let \( e_1, e_2 \) be two expressions in \( \text{NL}_{a} \) that satisfy the DVC (separately). Then \( \pi \sim_{\alpha} \pi \) is equivalent to the condition that there exists a permutation \( \pi \) with \( e_1 = \pi \cdot e_2 \), where \( \text{supp}(\pi) \subseteq (\text{Alt}(e_1) \cup \text{Alt}(e_2)) \setminus (\text{FA}(e_1) \cup \text{FA}(e_2)) \).

**Proof.** If \( e_1, e_2 \) satisfy the DVC and \( e_1 = \pi \cdot e_2 \) where \( \pi \) 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 \( \pi \) 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 \(\pi\). 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 \(e \cdot (a \ b) \cdot e'_2\) satisfy the dvc, by induction hypothesis, \(e'_1 = \pi' \cdot (a \ b) \cdot e'_2\), for a permutation \(\pi'\) where \(\pi'(a) = a\). Let \(\pi\) be the permutation \(\pi'(a \ b)\). Then \(\pi'(a \ b) \cdot b = a\). Hence \(\pi \cdot e_2 = e_1\). If \(e_1 = f \ e_1, \ldots e_{1,n}\), and \(e_2 = f \ e_{2,1}, \ldots e_{2,n}\), then by induction there are permutations \(\pi_i\) 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 \(\pi = \pi_1 \ldots \pi_n\) as the required permutation. ▶

**Remark 3.2.** The inductive definition of \(\sim_\alpha\) for abstractions is

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

where \(a\) may be equal to \(b\) or different. Generalizing this for arbitrary contexts results in the situation \(C_1 = \lambda a_1 \ldots \lambda a_n . [\cdot]\), \(C_2 = \lambda b_1 \ldots \lambda b_n . [\cdot]\), and the rule

\[
CA(C_1) \# C_2[s_2], \exists \pi : (C_1 \sim_\alpha \pi \cdot C_2, s_1 \sim_\alpha \pi \cdot s_2) \Rightarrow C_1[s_1] \sim_\alpha C_2[s_2]
\]

where \(\pi \cdot b_i = a_i\) for all \(i\), \(CA(C_1) = \{a_1, \ldots, a_n\}\), and \(CA(C_2) = \{b_1, \ldots, b_n\}\). 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. For an \(NL_\alpha\)-context \(C\), we denote with \(\text{CAO}(C)\) 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 \(C = \lambda a.\lambda a.( f \ (\lambda b.\lambda c. b) \ (\lambda b.\lambda c. [\cdot]))\), then \(\text{CAO}(C) = (a, b)\).

**Proposition 3.3.** Let \(C_1, C_2\) be \(NL_\alpha\)-contexts and \(e_1, e_2\) be \(NL_\alpha\)-expressions, such that \(C_1\) and \(C_2\) have identical hole positions, and \(C_1[e_1]\) as well as \(C_2[e_2]\) satisfy the dvc. Then \(C_1[e_1] \sim_\alpha C_2[e_2]\) is equivalent to

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

**Proof.** We show “ \(\Rightarrow\)” by induction on the length of the hole path of \(C_1\). If the length is 0, then the claim is trivial. For the induction step, let the length be strictly greater than 0.

- If \(C_1 = f \ e_1 \ldots C'_1 \ldots e_n\) then \(C_2 = f \ e'_1 \ldots C'_2 \ldots e'_n\). The capture condition holds, since \(CA(C'_1) = CA(C_1)\). The assumption and the congruence property of \(\sim_\alpha\) imply \(e_1 \sim_\alpha e'_1\) for all \(i \neq k\). By the induction hypothesis there is a permutation \(\pi_k\) 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 \(\alpha\)-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 \(C'_1 \sim_\alpha \pi'(a \ b) \cdot C'_2\), and \(e_1 \sim_\alpha \pi'(a \ b) \cdot e_2\), 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 $(a\ b)\ C_2'[e_2]$. This implies $\forall c \in CA(C_1): c \# C_2'[e_2]$. The application $\pi'(a\ b)\ b$ results in $a$, since $\pi'$ does not change $a$. Hence the required permutation is $\pi = \pi'(a\ b)$.

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

**Example 3.4.** The DVC is required in Proposition 3.3: Let $C_1 = f\ a\ \lambda a.[\ ]$ and $C_2 = f\ a\ \lambda b.[\ ]$, which implies $C_1[a] = (f\ \lambda a.\ a) \sim_\alpha (f\ \lambda b.\ b) = C_2[b]$. The DVC is violated for the left expression. We see that there does not exist a (common) permutation $\pi$ with $C_1 \sim_\alpha \pi\ C_2$ and $e_1 \sim_\alpha \pi\ e_2$.

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, which will refine the information.

**Definition 3.5.** Let $\rho$ be a ground substitution. The unification algorithm uses the following further constraint components:

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

**Proposition 3.6.** In NL-languages containing expression-variables, freshness constraints can in linear time be encoded as DVC-constraints by translating $a\#e$ into DVC($(f\ (\lambda A.\ S))$), plus the equation $S \doteq e$; and $A\#e$ into DVC($(f\ (\lambda A.\ S))$) plus the equation $S \doteq e$ where $f$ is a binary function symbol, and in both cases, $S$ is a new expression-variable.

### 4 The Unification Algorithm NomUnifyASD for NL$_{\alpha\ ASDP}$

The nominal unification problem in the language NL$_{\alpha\ ASDP}$ without any restriction seems to be too hard (at least we did not find an algorithm to solve it). We discuss the reasons that make the problem so hard, and which restrictions we introduce to handle 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 is permutation-variables, since nominal unification with permutation-variables but without contexts, known as equivariant unification [7, 10], is known to be solvable in EXPTIME. However, 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. For this reason, we forbid permutation variables in the input problem (however allow them during execution of the unification algorithm) and consider the specific class of so-called ASD1-unification problems:

**Definition 4.1 (ASD1-Unification Problem).** An ASD1-unification problem is a nominal unification problem $(\Gamma, \nabla)$ in NL$_{\alpha\ ASD}$ (see Definition 2.6), where each context variable $D$ occurs at most once in $\Gamma$ and all top-expressions $e_i$ of equations $(e_1 \doteq e_2) \in \Gamma$ have a DVC-constraint $DVC(e_i) \in \nabla$ (for $i = 1, 2$).
We describe a unification algorithm to solve ASD1-unification problems. During the execution of the algorithm the invariant, that context variables occur only once in the equations, must be kept and thus instantiations within the equations (which could duplicate occurrences of context variables) are not permitted. For example, transitions using replacement starting with \( S \models D[\ldots], S \models e_1, S \models e_2, \ldots \) would introduce two occurrences of \( D \), since the result is \( D[\ldots] \models e_1, D[\ldots] \models e_2, \ldots \). As a consequence we propose a Martelli-Montanari-style algorithm [21] that avoids instantiations within the unsolved equations.

The state during unification is a tuple \((\Gamma, \nabla, \theta, \Delta)\) using expressions from \(NL_{a,ASDP}\), where \(\Gamma\) is a set of sets of expressions, so-called multi-equations, \(\nabla\) is a set of freshness and dvc-constraints and further constraints of Definition 3.5, \(\theta\) is a substitution in triangle-form, represented as a set of components, and \(\Delta\) is a set of context-variables that are assumed to be nonempty. We omit the component \(\Delta\), if it is not changed by the unification rules.

Multi-equations \(M = \{e_1, \ldots, e_n\}\) will sometimes be written as \(e_1 \models e_2 \models \ldots \models e_n\), and we write \(\pi\cdot M\) to apply permutation \(\pi\) to all expressions in the multi-set, i.e. \(\pi\cdot M\) is the multi-equation \(\{\pi e_1, \ldots, \pi e_n\}\). We will assume that the expressions in \(\Gamma\) 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. However, to keep the presentation simple, we do not mention this compression and operations in the unification rules.

**Definition 4.2.** The input of the non-deterministic algorithm NomUnifyASD is an ASD1-unification problem \((\Gamma, \nabla)\), where \(\Gamma\) is a set of equations and \(\nabla\) a set of dvc- and freshness constraints, both over \(NL_{a,ASDP}\). The internal data structure is a tuple \((\Gamma, \nabla, \theta, \Delta)\), over \(NL_{a,ASDP}\). The algorithm finishes either with Fail, or, if \(\Gamma\) is empty and no failure rule applies, with a tuple \((\nabla', \theta, \Delta')\). The rules of the algorithm NomUnifyASD are shown in Figs. 2, 3, 4, 5, 6, which are partitioned into the following rule sets: The rule (flatten) in Fig. 1 is applied until no longer applicable. The variable-replacement and usual decomposition rules are in Fig. 2, the rules for decomposing multi-equations with expressions of the form \(D[\ldots]\), and with function symbols or \(\lambda\) as top symbol are in Fig. 3, the decomposition rules for multi-equations of expressions \(D[\ldots]\) are in Fig. 4, where the starred rules (DDPRm*) and (DDFrk*) are not used directly; the rules for guessing context-variables as empty or nonempty are in Fig. 5, and the failure rules are in Fig. 6. Rules (ID), (AD) make one (parallel) decomposition step, where rule (ID) first guesses a common first level of the hole positions of all \(D_i\). Rule (DDPrf) guesses that one context is a prefix of the others; rule (DDPRm) guesses a (maximal) common prefix of the contexts \(D_i\) (such that it is a proper prefix of all \(D_i\)) and rule (DDFrk*) 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.

---

3 A substitution in triangle-form is a shared representation of a substitution, e.g., the substitution in triangle-form \(\{x \mapsto f(y, z), y \mapsto a, z \mapsto \lambda b. b\}\) 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.
Nominal Unification with Atom and Context Variables

(flaten) \[
\Gamma \cup \{C[e] \doteq M\} \quad \text{where } C \neq [], e \text{ is neither } \pi \cdot S, \text{ nor } \pi \cdot A_i
\]
and \(S\) is a fresh variable.

\begin{itemize}
\item **Figure 1** The flatten-rule.
\end{itemize}

\begin{itemize}
\item \((\text{Elm1})\) \[\frac{\Gamma \cup \{e \doteq e \doteq M\}, \nabla, \theta}{\Gamma \cup \{e \doteq M\}, \nabla, \theta}\]
\item \((\text{Elm2})\) \[\frac{\Gamma \cup \{\{e\}\}, \nabla, \theta}{(\Gamma, \nabla, \theta)}\]
\end{itemize}

\begin{itemize}
\item \((\text{Slv1})\) \[\frac{(\Gamma \cup \{\pi_1 \cdot S \doteq \pi_2 \cdot V \doteq M\}, \nabla, \theta)}{\text{if } S \neq V, V \text{ is an } S- \text{ or } A- \text{variable or atom, and}}\]
\[\frac{(\Gamma \cup \{\pi_1 \cdot A \doteq \pi_2 \cdot X \doteq M\}, \nabla, \theta)}{\text{if } X \neq A \text{ is an atom or atom}-}\]
\item \((\text{Slv2})\) \[\frac{\Gamma \cup \{\pi_1 \cdot X_1 \doteq \pi_2 \cdot X_2 \doteq M\}, \nabla, \theta)}{\text{if } X_1, X_2 \text{ are atom-variables s.t.}}\]
\item \((\text{Slv3})\) \[\frac{(\Gamma \cup \{\pi_1 \cdot X_1 \doteq M\}, \nabla)}{\text{if } X_1, X_2 \text{ are fresh}}\]
\item \((\text{Slv4})\) \[\frac{(\Gamma \cup \{e \doteq M\}, \nabla, \theta \cup \{S \mapsto \pi^{-1} e\})}{\text{if } S \text{ does not occur in } M, e \text{ or } \Gamma}\]
\item \((\text{Mrg})\) \[\frac{(\Gamma \cup \{\pi_1 \cdot S \doteq M_1\} \cup (\pi_2 \cdot S \doteq M_2\}, \nabla, \theta)}{(\Gamma \cup \{S \doteq \pi^{-1} \cdot M_1 \doteq \pi^{-1} \cdot M_2\}, \nabla, \theta)}\]
\item \((\text{Slv5})\) \[\frac{(\Gamma \cup \{S \doteq M\}, \nabla)}{(\Gamma \cup \{\text{supp}(\pi) \# S\}, \nabla)}\]
\item \((\text{Abstr})\) \[\frac{\Gamma \cup \{\lambda X, e_1 \doteq M\}, \nabla, \theta)}{(\Gamma \cup \{\lambda A_1, e_1 \doteq M\}, \nabla, \theta)}\]
\item \((\text{Abb})\) \[\frac{\Gamma \cup \{\lambda A_1, e_1 \doteq M\}, \nabla)}{(\Gamma \cup \{\lambda A_1 \cdot e_1 \doteq M\}, \nabla)}\]
\end{itemize}

\begin{itemize}
\item **Figure 2** Rules of NomUnifyASD for Variables and Decomposition.
\end{itemize}

\begin{itemize}
\item \((\text{ed})\) \[\frac{(\Gamma \cup \{e_1 \doteq (\pi \cdot D)[e_2] \doteq M\}, \nabla, \theta, \Delta)}{\text{if tops}(e_1) \text{ is a atom variable, atom,}}\]
\item \((\text{ed})\) \[\frac{(\Gamma \cup \{e_1 \doteq v_2 \doteq M\}, \nabla, \theta, \Delta)}{\text{if tops}(e_1) \text{ is an atom variable, atom, or tops}(e_1) \text{ is fresh,}}\]
\item \((\text{id})\) \[\frac{(\Gamma \cup \{f_1 \ldots e_n \doteq (\pi_1 \cdot D_1)[e'_1] \doteq \ldots \doteq (\pi_m \cdot D_m)[e'_m]\}, \nabla, \theta, \Delta)}{\text{if } \forall i: D_i \in \Delta, \text{ and where context variables } D_{i,1} \text{ for } i = 1, \ldots, m \text{ are fresh and}}\]
\item \((\text{id})\) \[\frac{(\Gamma \cup \{X, e_0 \doteq (\pi_1 \cdot D_1)[e_1] \doteq \ldots \doteq (\pi_m \cdot D_m)[e_m]\}, \nabla, \theta, \Delta)}{\text{if } \forall i: D_i \in \Delta \text{ and } X \text{ is an atom or atom-variable}}\]
\end{itemize}

\begin{itemize}
\item **Figure 3** Rules of NomUnifyASD for F-D-Decomposition.
\end{itemize}
(DDPRm) First apply (DDPRm*); then apply rule (DDFrk) to the resulting multi-equation.

(DDFrk) Apply (DDFrk*), then remove all introduced variables $S_{i,j}$ using (Slv).

\[
\begin{align*}
&\text{(DDPR)} \quad (\Gamma \cup \{ (\pi_1 \cdot D_1)[e_1] \doteq \ldots \doteq (\pi_n \cdot D_n)[e_n] \}, \nabla, \theta, \Delta) \quad \text{if } \forall i : D_i \in \Delta \\
&\quad (\Gamma \cup \{ e_1 \doteq P_2 \cdot ((\pi_2 \cdot D_2_2)[e_2]) \doteq \ldots \doteq P_n \cdot ((\pi_n \cdot D_n_2)[e_n]) \}, \\
&\quad \nabla \cup \{ CA(D_i) \# \pi_i^{-1}.((\pi_i \cdot D_i)[e_i]), i = 2, \ldots, n \} \\
&\quad \cup \{ \text{supp}(P_i) \subseteq (CAO(\pi_1 \cdot D_1) \cup CAO(\pi_i \cdot D_i_1)) \mid i = 2, \ldots, n \}, \\
&\quad \theta \cup \{ D_i \mapsto D_{i,1}, D_{i,2}, D_{i,1} \mapsto \pi_i^{-1} \cdot P_i^{-1} \cdot \pi_1 \cdot D_1 \mid i = 2, \ldots, n \}, \\
&\quad \Delta \cup \{ D_{i,1} \mid i = 2, \ldots, n \} \text{ where } D_{i,1}, D_{i,2}, P_2, \ldots, P_n \text{ are fresh.}
\end{align*}
\]

\[
\begin{align*}
&\text{(DDFrk*)} \quad (\Gamma \cup \{ (\pi_1 \cdot D_1)[e_1] \doteq \ldots \doteq (\pi_n \cdot D_n)[e_n] \}, \nabla, \theta, \Delta) \quad \text{if } \forall i : D_i \in \Delta \\
&\quad (\Gamma \cup \{ (\pi_i \cdot D_i')[e_i] \mid i \in M_1 \} \cup \{ \pi_i : S_{i,1} \mid i \notin M_1 \} \\
&\quad \cup \ldots \cup \\
&\quad \{ (\pi_i \cdot D_i')[e_i] \mid i \in M_n \} \cup \{ \pi_i \cdot S_{i,m} \mid i \notin M_m \} \}, \nabla, \theta, \sigma, \Delta)
\end{align*}
\]

where $f$ with $ar(f) \geq 2$ and the index positions $j(i)$ for $i=1, \ldots, n$ are guessed such that $|\{j(i) \mid 1 \leq i \leq n\}| \geq 2$; and $M_k := \{ h \mid j(h) = k \}$ for $k = 1, \ldots, m$; and $\sigma = (D_i \mapsto \{ f : S_{i,1} \ldots S_{i,m} \mid 1 \leq i \leq n \}$ where $D_i', S_{i,v}$ are fresh.

\section*{Figure 4} Rules of NomUnifyASD for D-D-decomposition.

\[
\begin{align*}
&(\Gamma, \nabla, \theta, \Delta) \\
&\text{(GuessDEmpty)} \quad (\Gamma[[/D], \nabla[[/D]], \theta \cup \{ D \mapsto [/D] \}, \Delta)) \quad \text{If } D \notin \Delta, D \text{ occurs in } \Gamma
\end{align*}
\]

\[
\begin{align*}
&(\Gamma, \nabla, \theta, \Delta) \\
&\text{(GuessDNonEmpty)} \quad (\Gamma, \nabla, \theta, \Delta \cup \{ D \}) \quad \text{If } D \notin \Delta, D \text{ occurs in } \Gamma
\end{align*}
\]

\section*{Figure 5} Rules of NomUnifyASD for guessing $D$ empty or nonempty.

\[
\begin{align*}
&(\Gamma \cup \{ e_1 \doteq e_2 \doteq M \}, \nabla, \theta, \Delta) & \quad \text{if } tops(e_1) \text{ and } tops(e_2) \text{ are different atoms; or } \quad \text{if } tops(e_1) \quad \text{and } \text{tops(e_2)} \text{ are atom, } \lambda \text{ or a function symbol, and } \\
&\quad \text{Fail} & \quad \text{tops(e_1)} \neq \text{tops(e_2)}; \text{ or } \quad \text{tops(e_1)} \text{ is an atom or atom-variable, and } \text{tops(e_2)} \text{ is } \lambda \text{ or } f \in F.
\end{align*}
\]

\[
\begin{align*}
&(\Gamma \cup \{ S_1 \doteq e_1 \doteq M_1, \ldots, S_n \doteq e_n \doteq M_n \}, \nabla, \theta, \Delta) & \quad \text{Fail} \quad \text{if all } e_i \text{ are neither variables nor suspensions, all context variables occurring in } \\
&\quad \text{in } e_i \text{ are in } \Delta, S_{i+1} \text{ occurs in } e_i \text{ for } i = 1, \ldots, n - 1, \text{ and } S_1 \text{ occurs in } e_n \\
&(e\text{DFail}) \quad (\Gamma \cup \{ e_1 \doteq (\pi \cdot D)[e_2] \doteq M \}, \nabla, \theta, \Delta) & \quad \text{Fail} \quad \text{if } \text{tops(e_1)} \text{ is atom or an atom-variable,} \\
&\quad \text{if } \text{tops(e_1)) = S \text{ and } S \text{ occurs in } e_2; D \in \Delta \quad \text{or } \text{tops(e_1)} = \text{S and } S \text{ occurs in } e_2; D \in \Delta
\end{align*}
\]
Example 4.3. For $f\ a\ S_2\ S_1 = f\ S_1\ (\lambda a.\ S_3)\ S_3$ there is a substitution that equates the expressions. However, if there are DVC-constraints for the top expressions then these cannot be satisfied, since for example, the instantiation $\rho(S_3) = a$ cannot be $\alpha$-renamed. Another example is $f\ S\ S = f\ (\lambda a.\ a)\ (\lambda b.\ b)$, which is solvable by $\{S \mapsto \lambda a.\ a\}$: it does not lead to a DVC-violation, since it is treated as instantiation modulo $\alpha$.

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 $\text{NomUnifyASD}$, since without this information a decision algorithm appears to be impossible since permutation-variables as well as context-variables appear in the constraint.

Definition 4.4. The algorithm $\text{NomFreshASD}$ operates on the output $(\nabla, \theta, \Delta)$ of $\text{NomUnifyASD}$ 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 $\theta$, by a constant $c$ from the signature.

(III) Construct $M_\infty$ as a set of $|M_A| \times (d!)^2$ atoms by extending the set $M_A$ by further fresh atoms, where $d$ is the number of context-variables in $\Gamma$. Guess for every context-variable $D$ that occurs in $\Gamma, \nabla, \theta$ and that is not instantiated by $\theta$, the ordered set of captured atoms from the given set of atoms.

(IV) Guess the permutation-variables as bijections on the set $M_\infty$.

(V) Test the freshness constraints, equality, disequality, extended freshness constraints, and non-emptiness constraints, which are now immediately computable. To test whether the $\theta$ violates the DVC in $\Gamma$, 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 $\theta$. Then test the DVC-property, which is possible in polynomial time.

5 Properties of $\text{NomUnifyASD}$ and $\text{NomFreshASD}$

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

Lemma 5.1. If the input is $(\Gamma, \nabla)$, then the application of (flatten) to a subexpression of $e$ of the equations $\Gamma$ does not change the set of solutions.

Proposition 5.2. For a nominal unification problem $(\Gamma, \nabla)$ in $\text{NL_{\alpha ASD}}$ as input, the non-deterministic algorithm $\text{NomUnifyASD}$ terminates after a polynomial number of steps.
Proof. Let \( \text{maxArity} \) be the maximal arity of function symbols in the signature. Let \( \mu_1 := \mu_{1,1} + 2 \cdot \text{maxArity} \cdot \mu_{1,2} \) where \( \mu_{1,1} \) is the number of expressions in \( \Gamma \) and \( \mu_{1,2} \) is the number of occurrences of function symbols and \( \lambda \)-s in \( \Gamma \). Let \( \mu_2 \) be \( \mu_{1,1} \) minus the number of multi-equations. Let \( \mu_3 \) be the pair of the number of occurrences of context-variables in \( \Gamma \) and the number of context-variables in \( \Gamma \) which are not in \( \Delta \). Let \( \mu_4 \) be the number of occurrences of expressions \( \lambda X \), where \( X \) is \( \emptyset \cdot a \), \( \pi \cdot a \), or \( \pi \cdot A \) and \( \pi \) is nontrivial. The following table lists the relation between \( \Gamma \) before and after the application of the rule or subalgorithm, where GD(N)E abbreviates rules (GuessDEmpty) and (GuessDNonEmpty).

<table>
<thead>
<tr>
<th>rule</th>
<th>( \mu_1 )</th>
<th>( \mu_2 )</th>
<th>( \mu_3 )</th>
<th>rule</th>
<th>( \mu_1 )</th>
<th>( \mu_2 )</th>
<th>( \mu_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>any rule of Fig. 2 except Abstr &gt;</td>
<td>= = &gt;</td>
<td>= = &gt;</td>
<td>= = &gt;</td>
<td>DDFrk = = &gt;</td>
<td>DPPm = = &gt;</td>
<td>DDFr = = &gt;</td>
<td></td>
</tr>
<tr>
<td>Abstr</td>
<td>= = &gt;</td>
<td>= = &gt;</td>
<td>= = &gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eD</td>
<td>= = &gt;</td>
<td>= = &gt;</td>
<td>= = &gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It can be verified for (Elm1), (Elm2), (Slv1), (Slv2), (Slv3), (Slv4), (Mrg), (Slv5), (ff), (Abstr), (\( \lambda \lambda \)), and (eD) by a simple check. It is correct for (\( \text{ID} \)) and (\( \lambda \text{D} \)), since \( f e_1 \ldots e_m \) is removed, which counts \( 2^*\text{maxArity} \). (DDFrk) removes one occurrence of a context-variable. (DDFrk) 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 (DDPrm), it suffices to check (DDFrk). The measure \( \mu_3 \) 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.

\[ \triangleright \text{Lemma 5.3.} \text{ If all top expressions of the initial set of equations } \Gamma \text{ are restricted by the DVC, then this also holds for all top-expressions in the sequence of rule executions of the algorithm NomUnifyASD.} \]

\[ \triangleright \text{Proposition 5.4.} \text{ Inspecting the details of all rules of NomUnifyASD shows soundness: The solutions of the final data structure are also solutions of the input. The following rules of the algorithm NomUnifyASD do not lose any solutions, i.e. for every solution } \rho \text{ of the data structure } Q \text{ before application, there is a solution } \rho' \text{ of the output data structure } Q', \text{ such that } \rho(X) \sim_\alpha \rho'(X) \text{ for all atom-, expression-, context- and permutation-variables } X \text{ occurring in } Q; \text{ Rules from Fig. 2, rules (eD), (\( \lambda \text{D} \)) from Fig. 3, and the failures rules.} \]

\[ \triangleright \text{Proposition 5.5.} \text{ The algorithm NomUnifyASD is complete: If } \rho \text{ is a solution of the intermediate data structure } Q, \Gamma \text{ is not empty and no failure rule applies, then there is a possible rule application, such that there is solution } \rho' \text{ of the output } Q', \text{ and } \rho(X) \sim \rho'(X) \text{ for all atom-, expression-, context- and permutation-variables } X \text{ occurring in } Q. \]

We now consider the correctness and complexity of NomFreshASD.

\[ \triangleright \text{Lemma 5.6.} \text{ Let } H \text{ be an execution of NomUnifyASD starting with } S_0 := (\Gamma_0, \nabla_0, \theta_0, \Delta_0) \text{ where } \Gamma_0 = \Gamma, \text{ context variables occur at most once, and } \theta_0, \Delta_0 \text{ are trivial or empty. Let the sequence } H \text{ end with } S_{\text{out}} = (\emptyset, \nabla_{\text{out}}, \theta_{\text{out}}, \Delta_{\text{out}}), \text{ and let } \rho \text{ be a solution of the input as well as of the output } S_{\text{out}}. \text{ Then there is also a solution } \rho' \text{ that uses only a set of atom } VA_{\infty} \text{ with } \vert VA_{\infty} \vert \leq \vert VA \vert \cdot ((d + 1)^2), \text{ where the visible set } VA \text{ of atoms } VA = \{ a \mid a \text{ occurs in } H \} \cup \{ A p \mid A \text{ occurs in } H \}, \text{ and where } d \text{ is the number of context-variables in } \Gamma. \]

\[ \triangleright \text{Proposition 5.7.} \text{ Let } (\nabla_{\text{out}}, \theta_{\text{out}}, \Delta_{\text{out}}) \text{ be the output of NomUnifyASD for input } (\Gamma, \nabla). \text{ The algorithm NomFreshASD decides satisfiability of the output in NEXPTIME in the size} \]
Nominal Unification with Atom and Context Variables

of the input, where the main components are $|\langle \Gamma, \nabla \rangle| \ast (|d|!)^2$, where $d$ is the number of context-variables in $\Gamma$. For a fixed upper bound on the number of context-variables, satisfiability can be checked in NP time.

Remark. There is a complexity jump between freshness constraints and $dvc$-constraints in $NLASD$, since satisfiability of freshness constraints in $NLASD$ is in PTIME whereas satisfiability of $dvc$-constraints in $NLASD$ is NP-hard.

Combining Propositions 5.5, 5.2, and 5.7 shows:

Theorem 5.8. For $\Gamma, \nabla$ as input the algorithm NomUnifyASD 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 5.9. Solvability of ASD1-unification problems is in NEXPTIME.

6 Specializations, Applications and Examples

We consider nominal unification in $NLAS$ with freshness and $dvc$-constraints extending the result of [37] (see Theorem 2.7) by allowing $dvc$-constraints and by restricting NomUnifyASD

Theorem 6.1. The nominal unification problem in $NLAS$ where freshness- and $dvc$-constraints are permitted in $\nabla$ is solvable in polynomial time. Moreover, for solvable $(\Gamma, \nabla)$, there exists a most general unifier of the form $(\nabla', \theta)$ which can be computed in polynomial time, i.e., the problem class is unitary.

Proof. We assume that the algorithm computes in polynomial time a unifier $(\nabla', \theta)$ consisting of a substitution $\theta$ and a constraint set $\nabla'$, where in addition we assume that the output substitution $\theta$ is represented in triangle-form and that it is of polynomial size (see [33] for the technique). Soundness and completeness of computing only a single execution path follows from Proposition 5.4 since there are no permutation-variables and no context-variables.

For the final satisfiability test, we instantiate the expression-variables in the codomain of $\theta$ 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 $\alpha$. Then the check for every constraint $dvc(e)$, whether $e\theta$ satisfies the $dvc$, can be performed in polynomial time. 

The application of NomUnifyASD to $NLAS$ also yields at most one most general unifier, however, the complexity to check solvability is increased, since already the solvability of freshness constraints in $NLAS$ is NP-hard [33].

We now consider applications and examples.

Example 6.2. As a first example, we consider the equation $\lambda A_1. \lambda A_2. A_1 \doteqdot \lambda A_1. \lambda A_2. A_2$ together with $dvc$-constraints for both expressions. The algorithm NomUnifyASD finds a potential candidate for a solution, which sets $A_1 \mapsto A_2$. However, constraint checking using NomFreshASD fails, since for any instantiation which instantiates $A_1$ and $A_2$ with the same atom, the $dvc$ does not hold.

As a second example, consider the input equation $\lambda A_1. \lambda A_2. A_1 \doteqdot \lambda A_2. \lambda A_1. A_2$ together with $dvc$-constraints for both expressions. The algorithm NomUnifyASD finds a potential
candidate for a solution, which is the identity substitution for $A_1$ and $A_2$. Algorithm NomFRESHASD shows satisfiability of the dvc-constraints, where a requirement is that $A_1$ and $A_2$ are set to different atoms.

As a third simple example, consider the equation $D_1[A] \equiv D_2[B]$ with dvc-constraints for the input and the additional constraint $A \neq B$. We consider the execution that applies rule (DDPrf): This results in the potential solution which sets $A \mapsto P_2 \cdot B$, $D_2 \mapsto D_{2,1}$, $D_{2,1} \mapsto P_2^{-1} \cdot D_1$ and adds the constraints $CA(D_1) \# D_2[B]$, $supp(P_2) \subseteq CAO(D_1) \cup CAO(D_{2,1})$ to $\forall$. Algorithm NomFRESHASD detects satisfiability, for instance, by instantiating $A \mapsto a$, $B \mapsto b$, and then guessing $CAO(D_1) = \{a\}$ and $P_2 = (a \cdot b)$.

We now consider reductions and transformation rules. Most of them in the application domain of functional programming languages require freshness and/or dvc-constraints to exclude invalid instances of the rules.

► Example 6.3. The rule $\text{app (\lambda A.S)} \ S' \to \text{let } A = S' \text{ in } S$ is a sharing-variant of $\beta$-reduction. It needs the constraint $\text{dvc(app (\lambda A.S) S')}$ if $\text{let}$ is recursive, to ensure that for the instances, there are no free occurrence of $A \rho$ in $S' \rho$.

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

► Example 6.4. We describe an exemplary unification problem that occurs in correctness proofs of program transformations. A reduction rule in the let-calculus of [2] is $\text{let } A_x = (\text{let } A_y = S_y \text{ in } S_x) \text{ in } S_r \to \text{let } A_y = S_y \text{ in } \text{let } A_x = S_x \text{ in } S_r$ with dvc-constraint $\text{dvc(let } A_x = (\text{let } A_y = S_y \text{ in } S_x) \text{ in } S_r)$ 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 (nontrivial) overlap of the left hand side of the rule with itself (as a transformation) we form the unification equation $\text{let } A_x = (\text{let } A_y = S_y \text{ in } S_x) \text{ in } S_r \simeq D[\text{let } A'_x = (\text{let } A'_y = S'_y \text{ in } S'_x) \text{ in } S'_r]$ where the context-variable $D$ is intended as a representation of the reduction strategy. For correct application of the rule, the dvc-constraints $\text{dvc(let } A_x = (\text{let } A_y = S_y \text{ in } S_x) \text{ in } S_r)$ and $\text{dvc(D[let } A'_x = (\text{let } A'_y = S'_y \text{ in } S'_x) \text{ in } S'_r])}$ are required. We omit the case that $D \mapsto []$ and analyze the instantiation which sets $D \mapsto \text{(let } A_x = D_1 \text{ in } S_r)$. We obtain the equation $\text{let } A_y = S_y \text{ in } S_r \simeq D_1[\text{let } A'_x = (\text{let } A'_y = S'_y \text{ in } S'_x) \text{ in } S'_r]$. Guessing $D_1 \mapsto []$ results in $\text{let } A_y = S_y \text{ in } S_r \simeq \text{let } A'_x = (\text{let } A'_y = S'_y \text{ in } S'_x) \text{ in } S'_r$. Guessing $A_y \mapsto A'_y$, we obtain as solution $S_y \simeq (\text{let } A'_y = S'_y \text{ in } S'_x)$, $S_x \simeq S_x$. If we alternatively guess $A_y \neq A'_y$, we obtain as solution $S_y \simeq (\text{let } A'_y = S'_y \text{ in } S'_x)$ and $S_x \simeq (A_x \ A'_y) \cdot S'_x$ together with the constraint $A_y \# S'_r$.

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, where unification problems consist of unification equations, freshness and dvc-constraints. 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.

---

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


A Detailed Proofs

A.1 Completeness of NomUnifyASD

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) $[\cdot], i = 1, \ldots, n$ where every hole occurs at most once. For $n \geq 1$ and two $n$-contexts $C_1, C_2$ the relation $C_1 \sim_a C_2$ holds, if for all $n$-tuples $a_1, \ldots, a_n$ of (perhaps equal) atoms $a_i$, it holds that $C_1[a_1, \ldots, a_n] \sim_a C_2[a_1, \ldots, a_n]$ as expressions, which is a generalization from contexts to multi-contexts.

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

Lemma A.2. Let $n \geq 1$, $C_1, C_2$ be $NL_a$-$n$-contexts, and $e_1, e_i, i = 1, \ldots, n$ be $NL_a$-expressions, such that the corresponding hole positions of $C_1$ and $C_2$ are identical, and such that $C_1[e_1, \ldots, e_1,n]$ and $C_2[e_2, \ldots, e_2,n]$ satisfy the DVC. Then $C_1[e_1, \ldots, e_1,n] \sim_a C_2[e_2, \ldots, e_2,n]$ if the following holds:

- $CA(C_1) \neq CA(C_2)[e_2, \ldots, e_2,n]$, and there is a permutation $\pi$ that does not change free atoms in $C_2[e_2, \ldots, e_2,n]$ with $C_1 \sim_a C_2$ and $e_{1,i} \sim_a e_{2,i}$ for all $i$, $\pi$ maps $CAO(C_2)$ to $CAO(C_1)$, and $\supp(\pi) \subseteq CAO(C_1) \cup CAO(C_2)$.

Proposition 5.5. The algorithm NomUnifyASD is complete: If $\rho$ is a solution of the intermediate data structure $Q$, $\Gamma$ is not empty and no failure rule applies, then there is a possible rule application, such that there is solution $\rho'$ 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 $\rho$ 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 (Elm1) is applicable.
2. We can also assume that all context-variables that occur in $\Gamma$ are contained in $\Delta$, by applying either (cD), 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 $\rho$, and there is a solution $\rho'$ of the output that extends $\rho$. The condition that top-expressions in the input satisfy the DVC and that the input are ASD1-unification problems is necessary for the application of Proposition 3.3.
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 3.3 and the knowledge on permutations show that the execution of the rules is possible. In any case, there will be a solution $\rho'$ after the application that is an extension of $\rho$ (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 $\lambda$, or function
symbols, then rules (ff), (Abstr) or (λλ) can be applied and there is a solution $\rho'$ extending $\rho$ after the application. If there are two expressions of the form $\pi.X$, where $X$ is an atom or expression-variable, then (Slv1), (Slv2), (Slv3), or (Slv4) is applicable, and there is a solution $\rho'$ extending $\rho$ after the application. Similar for the case where $\text{tops}(\cdot)$ yields atom twice. If one expression is of the form $\pi.X$, and the other is not of this form, then we apply (Slv4) if this is possible.

6. The final case is that for all equations, the equation pattern permits only applications of rule (Slv4), all multi-equations have only two expressions, but the conditions on non-containment of $S$ in rule (Slv4) 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.2 Satisfiability of Constraints of NomUnifyASD

Lemma 5.6. Let $H$ be a sequence of executions of $\text{NomUnifyASD}$ starting with $S_0 := (\Gamma_0, \Delta_0, \theta_0)$ where $\Gamma_0 = \Gamma$, the condition on the input are as stated in Definition 4.4, and $\theta_0, \Delta_0$ are trivial or empty. Let the sequence $H$ end with $S_{\text{out}} = (\emptyset, \Delta_{\text{out}}, \theta_{\text{out}})$, and let $\rho$ be a solution of the input as well as of the output $S_{\text{out}}$. Then there is also a solution $\rho'$ that uses only a set of atom $VA_{\infty}$ with $|VA_{\infty}| \leq |VA|*2^2$, 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 $\Gamma$.

Proof. We show a stronger claim by induction on the steps of the execution: Let $D_F$ and $D_F$ be the set of context-variables $D_0$ in the applications of (DDPrf) in $H$ and $D_1,0$ in the application of (DDPRm) in $H$, and $D_i, i = 2, \ldots, n$ in the applications of (DDPrf) and (DDPRm), 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 $\theta$. 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 5.2. The claim is: there is a solution $\rho'$ that uses only the set $VA_{\infty}$ 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 (DDPrf) or (DDPRm) was executed. The construction implies $VA_i \subseteq VA_{i+1}$. The final $VA_{\infty}$ is defined as the final $VA_i$.

We define the construction: Let $i$ be an index in $H$ and $S_i = (\Gamma_i, \Delta_i, \theta_i)$ be a state in $H$ with set $VA_i$, such that the next step is (DDPrf) or (DDPRm) leading to $S_{i+1}$. Let us assume that it is the first occasion in $H$ such that a focused context-variable or permutation-variable that is in the focus of a rule application (DDPrf) or (DDPRm), uses an atom $a'$ in its instances under $\rho$, where $a' \notin VA_i$. Then the following changes are made to the solution $\rho$, resulting in $\rho'$ and a modified execution sequence $H'$.

First consider the modification concerning the rule (DDPrf):

- Using the same notation as in the rule application, we consider the instances $(CAO(\pi_1.D_1)\rho)$, $(CAO(\pi_2.D_2)\rho)$, ..., $(CAO(\pi_n.D_n)\rho)$. We can assume that $(\pi_1.D_1)\rho$, $(\pi_2.D_2)\rho$, ..., $(\pi_n.D_n)\rho$ only consist of $\lambda_{k,1}, \ldots, \lambda_{k,m}\ldots$ where $m = |CAO(\pi_1.D_1)|$, by modifying the solution $\rho$, which is without effect on the further execution of the algorithm $\text{NomUnifyASD}$ 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 $(\pi_l \cdot D_{k,1})\rho$ to $CAO(\pi_l \cdot D_1)$ for $k \geq 2$, which is justified by Proposition 3.3. Hence $P_k$ does not need any extra fresh atoms in its support. Using the thus modified $\rho$, 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 $|V_i| \cdot d'$, where $d'$ is the number of context-variables in $\Gamma_i$. The number of all used atoms in the instances is at most $|V_i| \cdot d' \cdot d''$.

Let $\rho'$ be $\rho$ after these modifications. The ground substitution $\rho'$ is a solution of the state $S_{i+1}$, and such that the same execution still leads to a final state that covers $\rho'$.

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 (DDPRm). It is similar to the previous case, but we detail it, since the names of variables are different.

Consider the instances $(CAO(\pi_l \cdot D_{1,0})\rho), \ldots, (CAO(\pi_m \cdot D_{0,0})\rho)$. We can assume that $(\pi_k \cdot D_{k,0})\rho$ only consist of $\lambda a_{k,1}, \ldots, a_{k,m}[:]$ where $m = |CAO(D_{1,0})|$, by modifying the solution $\rho$, which is without effect on the execution of the algorithm NomUnifyASD 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 $\rho$, 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 $|V_i| \cdot d' \cdot d''$ where $d''$ is the number of context-variables in $\Gamma_i$.

Let $\rho'$ be $\rho$ after these modifications. The ground substitution $\rho'$ is a solution of the state $S_{i+1}$, and such that the same execution still leads to a final state that covers $\rho'$.

The number of executions of rules (DDPrf), (DDPRm) is at most the number of different context-variables. This holds, since (DDPrf) removes one context-variable, and since (DDPRm) calls (DDFrk), and (DDFrk) 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 $(CAO(\pi_l \cdot D_1)\rho)$, there can at most be $d_i \cdot |V_i|$ variables, where $d_i$ is the number of context-variables in $\Gamma_i$. Since there is an iterated multiplication, we obtain $|VA| \cdot d \cdot d \cdot (d - 1) \cdot (d - 1) \ldots$, which leads to the estimation as claimed.

This change can be iterated until there are no (DDPrf), (DDPRm)-steps having an index $j$ where completely fresh variables are used for all context-variables in the multi-equation. Finally, we have constructed $VA_\infty$, and the modified solution $\rho'$.