Journal of Symbolic Computation ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

Nominal unification with atom-variables ☆

Manfred Schmidt-Schauß, David Sabel, Yunus D.K. Kutz

Goethe-University Frankfurt am Main, Germany

ARTICLE INFO

Article history:
Received 9 March 2017
Accepted 25 June 2017
Available online xxxx

Keywords:
Program transformations
Nominal unification
Functional languages
Atom-variables
Correctness

ABSTRACT

The problem of nominal unification where variables are allowed for atoms, and computing a complete set of unifiers is considered. The complexity is shown to be NP-complete, while for special cases there are polynomial time algorithms. The main result is a novel algorithm to compute a complete set of unifiers which performs lazy guessing of equality or disequality of atom-variables, runs in NP time, and the collecting variant has more chances to keep the complete set of unifiers small. Applications of this algorithm are in reasoning about program transformations in higher order functional languages. We also present a variant of the unification algorithm that delays guessing and checking solvability, and produces a single most general unifier.

1. Introduction

Motivation, applications and goals The initial motivation to introduce and investigate nominal reasoning and unification was the observation that formalizations of proofs in higher-order deduction systems like Isabelle (Nipkow et al., 2002; Urban and Kaliszyk, 2012) would profit from a more machine-oriented formalization of reasoning about α-equivalence and for arguments in automated proofs that employ renaming of bound variables by fresh names. This leads to the development and application of nominal theories and techniques (a recent tutorial is Pitts, 2016, a broad overview is

☆ The first author is supported by the Deutsche Forschungsgemeinschaft (DFG) under grant under grant SCHM 986/11-1. The second author is supported by the Deutsche Forschungsgemeinschaft (DFG) under grant SA 2908/3-1.

E-mail addresses: schauss@ki.informatik.uni-frankfurt.de (M. Schmidt-Schauß), sabel@ki.informatik.uni-frankfurt.de (D. Sabel), kutz@ki.informatik.uni-frankfurt.de (Y.D.K. Kutz).

https://doi.org/10.1016/j.jsc.2018.04.003
0747-7171/

Pitts, 2013) and also of nominal unification (Urban et al., 2003; Calvès and Fernández, 2008; Levy and Villaret, 2010).

An often used assumption in the technique of nominal reasoning is that variable names (called atoms) in higher-order binders are concrete, and not abstract, and that modifying the terms is done by applying atom permutations to terms and replace atoms, irrespective of their binding status (see Urban et al., 2003).

In the application field of verifying program transformations in higher order languages using syntactical reasoning and the operational semantics (for instance, defined by a small step reduction relation), a foundational reasoning task is to compute overlaps between left hand sides of small-step-reduction rules and transformation rules employing a unification algorithm. This task is similar to the computation of critical pairs in term rewrite systems (variants for nominal syntax were recently investigated by Ayala-Rincón et al., 2016). A difference is that instead of all overlaps only those overlaps have to be taken into account which match the reduction strategy given by the operational semantics.

An algorithm for computing overlaps for a class of higher-order program calculi is described in Schmidt-Schauß and Sabel (2016): It is an algorithm for unification of higher-order expressions with meta-variables, where (among others) expression-variables and variable-variables are used. As a consequence, in that approach, there are no concrete atoms, but "abstract" atoms. For this application it is indispensable to compute complete sets of unifiers. As correctness criterion, syntactical equality is used instead of alpha-equivalence. The treatment of alpha-equivalence and binding constraints uses so-called non-capture constraints. Further reasoning on correctness and executing reductions and/or transformations requires to also use and reason about alpha-renamings, which is, however, not supported by the unification method in Schmidt-Schauß and Sabel (2016).

In this paper we focus on a combination of the approaches and investigate nominal unification with the feature of abstract atoms (called *atom-variables*).

We focus on unification of nominal terms with atom-variables, since it is a basic reasoning task required by several reasoning algorithms. Hence, the goal of the paper is to construct a unification algorithm for higher-order meta-expressions which comprise atom-variables. In contrast to Schmidt-Schauß and Sabel (2016), we leave the task of combining the methods also with extra constructs like context variables, letrec-constructs and binding chains (which is required for sophisticated reasoning in call-by-need lambda calculi) for further research.

We illustrate the differences between nominal unification with concrete atoms and on the other hand with atom-variables. As a quite small example consider the equation $\lambda x.\lambda y.x \doteq \lambda x.\lambda y.y$. If x, y are atoms, then the equation is ground on both sides (there are no unification variables) and there is no solution: nominal unification only checks alpha-equivalence which results in the unsolvable equation $x \doteq y$. But, if x, y represent unification variables which can be instantiated by atoms, then there exists a solution which instantiates x, y by the same atom a. Furthermore, nominal unification with atom-variables generalizes usual nominal unification, if so called freshness constraints are allowed in the input, since they can enforce disequality of instantiations of atom-variables. For instance, the freshness constraint $x\#y$ ensures that atom-variables x and y must not be instantiated by the same atom. Thus, the equation $\lambda x.\lambda y.x \doteq \lambda x.\lambda y.y$ together with freshness constraint $x\#y$ represents usual nominal unification and thus has no solution.

As a motivating example for the usefulness of atom-variables in applications, we consider the following instance of a call-by-value beta reduction rule with 3-ary lambda expressions where the arguments are variables, which may be enforced by the syntax. As a small-step reduction rule one would usually write

$$(\lambda(x_1, x_2, x_3).e)\ (y_1, y_2, y_3) \to e[y_1/x_1, y_2/x_2, y_3/x_3]$$

where y_1, y_2, y_3 are variables and implicitly the binders x_1, x_2, x_3 are meant to be different. To (syntactically) reason on such a rule, it has to be represented by using meta-syntax. However, with usual nominal syntax (with atoms but without atom-variables), it is *insufficient* to represent the rule as

$$(\lambda(a_1, a_2, a_3).S)\ (b_1, b_2, b_3) \to S[b_1/a_1, b_2/a_2, b_3/a_3],$$

Please cite this article in press as: Schmidt-Schauß, M., et al. Nominal unification with atom-variables. J. Symb. Comput. (2018), https://doi.org/10.1016/j.jsc.2018.04.003