Theoretical Computer Science 737 (2018) 62–80

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Nominal essential intersection types

Maurício Ayala-Rincón [a,*], Maribel Fernández [b,*], Ana Cristina Rocha-Oliveira [a,*], Daniel Lima Ventura [c,*]

[a] Departamentos de Ciência da Computação e Matemática - Universidade de Brasília, Brazil
[b] Department of Informatics - King's College London, United Kingdom of Great Britain and Northern Ireland
[c] Instituto de Informática - Universidade Federal de Goiás, Brazil

ARTICLE INFO

Article history:
Received 16 August 2017
Received in revised form 2 May 2018
Accepted 2 May 2018
Available online 7 May 2018
Communicated by D. Sannella

Keywords:
Nominal syntax
Nominal rewriting
Binding
Essential intersection types
Subject reduction

ABSTRACT

Nominal systems are an alternative approach for the treatment of variables in computational systems, where first-order syntax is generalised to provide support for the specification of binding operators. In this work, an intersection type system is presented for nominal terms. The subject reduction property is shown to hold for a specialised notion of typed nominal rewriting, thus ensuring preservation of types under computational execution.

1. Introduction

Introducing variable binders in a language that works with names requires a mechanism to deal with α-equivalence, that is, invariance of objects modulo the renaming of bound variables. In logic, the existential and universal quantifiers are examples of constructors that need the binding engine to work. For instance, it must be possible to derive the equivalence between the formulas $\exists x : x > 1$ and $\exists y : y > 1$, despite their syntactical differences. In programming languages, two programs that differ only on the choice of variable names are considered equivalent. *Nominal theories* deal with binders using atoms (or variable names) and an abstraction construct. Atom-permutations are used to deal effectively with renamings and freshness constraints. This approach was introduced by Gabbay and Pitts [24], where the Fraenkel–Mostowski permutation model of set theory with atoms (FM-sets) is indicated as "the semantic basis of meta-logics for specifying and reasoning about formal systems involving name binding, α-conversion, etc".

Nominal syntax generalises first-order syntax by providing support for the specification of languages with binding operators. In nominal syntax, there are two kinds of variables: *atoms*, which are used to represent object-level variables and can be abstracted but not be substituted, and meta-variables, called simply *variables* or *unknowns*, which can be substituted but cannot be abstracted. Substitution of a variable by a term is closer to first-order substitution where variables act as holes to be filled by terms, possibly capturing atoms (unlike higher-order theories, where substitution is non-capturing). Moreover, β-equivalence is not a primitive notion in nominal syntax, in contrast with the higher-order and explicit substitution approaches (cf. [28,15,1]). Explicit substitution calculi are associated with higher-order rewriting systems, where substitu-

* Corresponding authors.
E-mail addresses: ayala@unb.br (M. Ayala-Rincón), maribel.fernandez@kcl.ac.uk (M. Fernández), anacrismarie@gmail.com (A.C. Rocha-Oliveira), daniel@inf.ufg.br (D.L. Ventura).

https://doi.org/10.1016/j.tcs.2018.05.008
0304-3975/

tions are manipulated explicitly; some of them use de Bruijn indices to implement the substitution operation together with α-conversion in a first-order setting (see [39]). Using a nominal rewriting system ([23]), capture-avoiding substitutions can be specified with no need to manage indices as done in some explicit substitutions calculi, since names and α-equivalence are primitive notions in nominal systems.

Type systems may help programmers to detect and avoid run-time errors in programming languages but also can be used to classify programs according to their semantics. For instance, Church's Simple Type System for the λ-calculus (see [27]) ensures $\beta\eta$-strong normalisation, i.e. termination of computations regardless of evaluation strategies, of typable terms. However, not all strongly normalisable terms are typable in the system. For example, the term $\lambda x.xx$ representing the self application function is a normal form, i.e. with no computation/reduction to be performed, and yet has no type in the type system of Hindley [27].

Intersection types were originally introduced by Coppo and Dezani-Ciancaglini [13] as an extension of the simply typed λ-calculus, intended to characterise strong normalisation in the λ-calculus (see [8]). A term t may have two types σ and τ in a system with intersection types, denoted by $t : \sigma \cap \tau$; thus, the term $\lambda x.xx$ is typeable in an intersection system, by assuming $x : (\tau \to \sigma) \cap \tau$. Indeed, Coppo et al. [14] showed that any solvable term has a meaningful type, and Barendregt et al. [7], provided a (filter) model for the calculus itself. Hence, the fixed point combinator $\lambda f.(\lambda x.f\,(xx))\;(\lambda x.f\,(xx))$, which is solvable[1] although non-terminating, is also typeable in such systems. Intersection types have been successfully applied in the characterisation of termination properties beyond the λ-calculus ([9,30,31,36]). Their finitary polymorphism (as opposed to the polymorphism in System F, see [26]), allows one to obtain typing systems with computationally relevant properties, such as principal typings ([42]), also known as principal pairs, and decidable typing systems ([32]).

Various notions of typing have been proposed for nominal theories, in order to classify terms or simply to avoid undesirable syntactic constructions. Adapting a type system proposed for λ-terms to the nominal syntax is not straightforward, since the notions of substitution are different in each system (nominal substitutions, like first-order substitutions, may capture atoms since they simply replace a variable with a term). For instance, in the simply typed λ-calculus, the well-known substitution lemma ensures that $\Gamma \vdash t[x \mapsto s] : \sigma$ holds whenever $\Gamma, x : \gamma \vdash t : \sigma$ and $\Gamma \vdash s : \gamma$ hold (see [8], Proposition 1.2.5.). This property holds because the free variables of s are not captured. In a nominal system, one also must take into account the types assigned to atoms in the leaves of the corresponding type derivation.

This paper presents an Essential Intersection Type System for nominal terms, inspired by Bakel [3], Bakel and Fernández [6], which addresses the specificities of the nominal framework and provides results of preservation of typings for α-equivalent terms and subject reduction for a notion of typed nominal rewriting. Throughout Section 5, examples are given to show the necessity of the conditions added in typed matching, typed rewriting and, finally, in the theorem of subject reduction. The restrictions imposed on the nominal typed rewriting relation are inspired by the polymorphic nominal type system in Fairweather and Fernández [19].

1.1. Related work

Intersection types have been applied in a variety of systems, with a variety of (semantic) investigation purposes. They are used in the characterisation of strong normalisation for explicit substitution calculi in Lengrand et al. [34], Bernadet and Lengrand [9], while intersection type systems are presented for several explicit calculi with de Bruijn indices in Ventura et al. [41], all proved to have the subject reduction property. Characterisation of termination properties were also obtained through intersection types for computational interpretations of focused sequent calculi, e.g. [25,17,31]. They are also applied in investigations of termination properties for the π-calculus ([36]) and the semantics of session types ([10,35]), when combined with union types ([16]).

A restriction of the type system from Barendregt et al. [7] was introduced by Bakel [3], called Essential Intersection Type System, while preserving its main properties. Syntax-directed systems such as the one presented in Bakel [3] have at most one typing derivation of a given typing, as opposed to the multiplicity of derivations in the system of Barendregt et al. [7]. Bakel and Fernández [6] presented an Essential Intersection Type System for Curryfied Term Rewrite Systems, based on the typing system in Bakel [3]. With a few restrictions on the rewrite rules, the authors were able to prove subject reduction for such systems. The system proposed in the present work is based on Bakel [3], Bakel and Fernández [6] with respect to the intersection type features.

On type systems for nominal syntax, Fernández and Gabbay [22] define a rank 1 polymorphic type system that explores, for the first time, syntax-directed type inference for nominal terms. A principal type function is presented that applies to a term with a type environment and a freshness context, and returns the most general type for the given parameters. Subject reduction holds for a specialised notion of rewrite step involving types.

Fairweather [18] follows the presentation of Fernández and Gabbay [22] and defines simple type systems *à la* Church (where α-equivalence and freshness are redefined to take into account the typed syntax) and *à la* Curry, which are then extended to include ML-style polymorphism, and dependent types. In the latter case, an extended syntax is used, where non-capturing atom-substitution is a primitive notion. Fairweather et al. [20] presents a preliminary version of the polymorphic system; the dependent type system is described in Fairweather et al. [21]. Typed nominal rewriting and nominal

[1] Corresponding to terms with head normal form in the λ-calculus.