Available online at www.sciencedirect.com

SCIENCE DIRECT®

Information and Computation 201 (2005) 121–159

Information and Computation

www.elsevier.com/locate/ic

Source-tracking unification[☆]

Venkatesh Choppella [a,*], Christopher T. Haynes [b]

[a] Indian Institute of Information Technology and Management—Kerala, Thiruvananthapuram, Kerala 695 581, India
[b] Computer Science Department, Indiana University, Bloomington, IN 47405, USA

Received 2 December 2003; Received 3 September 2004
Available online 10 August 2005

Abstract

We propose a path-based framework for deriving and simplifying source-tracking information for first-order term unification in the empty theory. Such a framework is useful for diagnosing unification-based systems, including debugging of type errors in programs and the generation of success and failure proofs in logic programming. The objects of source-tracking are deductions in the logic of term unification. The semantics of deductions are paths over a unification graph whose labels form the suffix language of a semi-Dyck set. Based on this idea of unification paths, two algorithms for generating proofs are presented: the first uses context-free labeled shortest-path algorithms to generate optimal (shortest) proofs in time $O(n^3)$ for a fixed signature, where n is the number of vertices of the unification graph. The second algorithm integrates easily with standard unification algorithms, entailing an overhead of only a constant factor, but generates non-optimal proofs. These non-optimal proofs may be further simplified by group rewrite rules.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Algorithms; Debugging; Formal languages; Graph theory; Logic programming; Path problems; Term unification; Type inference

[☆] This article is part of the 19th International Conference on Automated Deduction (CADE-19). [Information and Computation, Volume 199, Numbers 1/2, 2005].
[*] Corresponding author.
E-mail address: choppell@iiitmk.ac.in (V. Choppella), chaynes@cs.indiana.edu (C.T. Haynes).

0890-5401/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2004.10.013

1. Introduction

The term unification problem in the empty theory, also called the syntactic unification problem, is concerned with solving equations over syntactic terms built from variables and function symbols. Term unification has diverse applications including automated theorem proving, artificial intelligence, databases, type reconstruction in programming languages, and logic programming (Prolog).

A solution for a system of term equations, also called a *unifier*, is a substitution (a mapping from variables to terms) such that the substitution, when applied, makes each of the terms on the left- and the right-hand side of equation equal. For example, consider the term equation

$$\mathsf{f}(x, y) \stackrel{?}{=} \mathsf{f}(\mathsf{a}, z), \tag{1}$$

in which x, y, and z are variables, and f is a function symbol and a is a constant. It can be verified that the substitution

$$x \mapsto \mathsf{a}, \quad y \mapsto \mathsf{b}, \quad z \mapsto \mathsf{b},$$

where b is a constant, is a unifier of Eq. (1). The solution $x \mapsto \mathsf{a}$, $y \mapsto z$ is also a unifier.

It is also possible that a system of equations has no solutions. Consider, for example, the equation

$$\mathsf{f}(x, y) \stackrel{?}{=} \mathsf{g}(x, y). \tag{2}$$

No substitution can make the two terms involved in the equation equal. In such a case, we say that the system of term equations fails to unify, or is *non-unifiable*.

This paper is motivated by the question of determining why a system of equations fails to unify. The reason for non-unifiability of Eq. (2) is simple: the head symbols do not match. In general, however, determining the cause of failure could lead to long chains of reasoning involving many original and derived term equations. The more general problem consists of *source-tracking* unification, that is, deriving diagnostic information about unifiability or non-unifiability of a system of equations in terms of the original representation of the unification problem. Non-unifiability is related to type errors in programming languages [5,21,29,55] and unsuccessful queries in logic programs [10,17]. The reporting of this failure can be confusing and inadequate for reconstructing the error.

The origins of the unification problem can be traced to the 1930s work of Herbrand [27]. In the 1960s, Robinson coined the term "unification" and showed how it lay at the heart of resolution-based theorem proving [48]. Robinson defined the notion of a most general unifier, a unifier from which all other unifiers may be derived by applying a suitable substitution, and proposed an algorithm for computing most general unifiers. The many variants and generalizations of the unification problem (E-unification, higher-order unification, semi-unification, etc.) and their diverse applications have made unification an important area of research in theoretical computer science and artificial intelligence. Surveys of unification with its applications in other areas can be found in [31] and [2].

Traditionally, there have been two main approaches to studying the unification problem and designing algorithms for computing most general unifiers. The first is the transformational approach to unification introduced by Martelli and Montanari [37], studied by Lassez et al. [33], and