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.

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

surveyed by Jouannaud and Kirchner [30]. In this approach, a system of term equations is repeatedly transformed until it is in "solved form." This approach has the advantage of abstracting the representation of the unification algorithm and the control of algorithms computing unifiers. The second approach relies on an efficient representation of the unification problem as a directed acyclic graph with structure sharing. Unification is then viewed as a process of constructing a relation (the unification closure) over the vertices of the unification graph. The representation of terms as graphs with structure sharing was proposed by Boyer and Moore [8] and grew out of Robinson's tabular representation of terms [47]. The view of unification as closure computation was proposed by Paterson and Wegman in their paper on linear unification [43]. The relational view has the advantage that it pays close attention to the representation and implementation of the unification problem.

There are two other lesser cited approaches to unification in the literature. The first is the logical view of unification, proposed by Le Chenadec [34] which builds deduction systems to express unification and congruence closures. The second approach, suggested by Cox [18], views unification in terms of context-free language recognition by a pushdown automaton.

The framework presented in this paper is based on the relational approach and draws from the approaches of Le Chenadec and Cox. Its development is, however, motivated by simplicity and the desire to extract practical algorithms for source-tracking unification.

The framework consists of four components:

(1) A model theory expressing unification quotient graph connectivity in terms of *unification paths*, which are paths over the unification graph whose labels are suffixes of semi-Dyck labeled-paths over the original graph. As a special case, the problem of determining membership in the unification closure is reduced to computing paths whose labels form strings of balanced parentheses.
(2) A type system logic P^U for formalizing the reasoning involved in unification. The logic is sound and complete with respect to unification paths.
(3) A unification algorithm implementing the deduction system P^U. The algorithm is derived as a simple and inexpensive extension to the standard unification algorithm that integrates proof generation with unification.
(4) A scheme for normalizing witnesses computed by the unification source-tracking algorithm using elementary group rewriting.

Since our framework for source-tracking unification is graph-theoretic and is based on analyzing the structure of the unification graph representing the unification problem, it is closer to the relational rather than the transformational view of unification. The framework examines the unification graph's underlying labeled directed graph, obtained by orienting and labeling each edge of the unification graph. The framework rests on a fundamental property relating paths in labeled directed graphs to paths in their quotients under unification closure. The unification rule for equating subterms (downward closure) may be captured by a connectivity relation on the vertices of the underlying labeled directed graph. By suitably labeling the edges of the unification graph, we obtain a characterization of witnesses to quotient graph connectivity as paths whose labels form the suffix language of a semi-Dyck set (the language of balanced parentheses). This characterization allows the use of formal language path algorithms for witness computation [3,40]. Based on this observation, we propose an $O(n^3)$ algorithm where n is the size of the graph representing the

unification problem, for computing the optimal (smallest sized) unification paths sufficient to derive non-unifiability.

The logic P^U is a type system for well-typed *unification path expressions* whose type represents connectivity relations in the quotient graph. P^U encodes the deductions made by unification algorithms based on structure sharing. On the basis of this characterization, we define an efficient unification source-tracking algorithm that computes unification path expressions as it performs unification with an overhead of only a constant time factor.

The paths computed by the source-tracking algorithm are normalized in linear time using elementary group rewrite rules. While the simplification rules are well-known (the free group reduction rules of the equational rewrite system of Peterson and Stickel [44]), their interaction with the "typed" unification path expressions is somewhat delicate, partly due to the interplay between one-sided and two-sided cancellation laws in semi-Dyck sets. The usual subject reduction property fails to hold. Instead, types lost after a one-step rewrite are recovered at normalization. We call this phenomenon *weak subject reduction.*

The source-tracking algorithm suggested here has been implemented in Scheme [12]. The framework presented here has been the basis of the type error reporting algorithm proposed in [13].

The rest of this article is divided into the following sections: Section 2 outlines the main ideas with an example and informally shows how non-unifiability can be captured via specially labeled-paths in the unification graph. Section 3 defines the machinery of unification in terms of labeled directed graphs. Formal definitions for term graphs, unification graphs, labeled graphs, unification closure, and quotient graphs are presented. The basic result of Paterson and Wegman [43] about unification closure and unifiability is recalled. The notation used in the rest of the paper is summarized. Section 4 identifies source tracking as the problem of tracking the source of paths in the quotient graph in terms of paths in the original graph. It introduces the notion of unification paths and their fundamental role in unification source-tracking. Section 5 introduces a deduction system P^U for expressing proofs of membership in the unification closure of a graph. It also defines unification path expressions as the well-typed terms of this deduction system. Section 6 shows how to inexpensively integrate the construction of unification path expressions into the unification algorithm and how to accommodate some standard optimizations of the unification algorithm with the construction source-tracking information. Section 7 shows how to simplify the unification expressions using elementary group rewriting. Section 8 discusses related work. The related work section is slanted towards the application of unification in type inference. Section 9 concludes with suggestions for future work.

An earlier version of this work was reported in [15].

2. Motivating example

The main ideas of this paper are informally introduced using the set of term equations shown in Fig. 1. These equations are type equations whose derivation, with source-tracking as the central concern, is reported elsewhere [13,14].

The equations $f : t_6 \stackrel{?}{=} t_7 \to t_4$ and $h : t_6 \stackrel{?}{=} \texttt{int} \to \texttt{int}$, together imply

$$t_7 \to t_4 \stackrel{?}{=} \texttt{int} \to \texttt{int}.$$

$$a : t_0 \stackrel{?}{=} t_1 \rightarrow t_2 \quad b : t_2 \stackrel{?}{=} t_4 \quad c : t_3 \stackrel{?}{=} \texttt{bool}$$
$$d : t_4 \stackrel{?}{=} t_5 \quad e : t_3 \stackrel{?}{=} t_1 \quad f : t_6 \stackrel{?}{=} t_7 \rightarrow t_4$$
$$g : t_5 \stackrel{?}{=} t_1 \quad h : t_6 \stackrel{?}{=} \texttt{int} \rightarrow \texttt{int} \quad i : t_7 \stackrel{?}{=} t_1$$

Fig. 1. Example of a non-unifiable set of term equations.

Equating subterms yields the derived equations

$$j : t_7 \stackrel{?}{=} \texttt{int},$$
$$k : t_4 \stackrel{?}{=} \texttt{int}.$$

The equations j, i, e, and c form a chain of equality

$$\texttt{int} \stackrel{?}{=} t_7 \stackrel{?}{=} t_1 \stackrel{?}{=} t_3 \stackrel{?}{=} \texttt{bool}$$

implying $\texttt{int} \stackrel{?}{=} \texttt{bool}$, which is a *symptom* of non-unifiability of the original set of equations. Also, k, d, g, e and c form the chain

$$\texttt{int} \stackrel{?}{=} t_4 \stackrel{?}{=} t_5 \stackrel{?}{=} t_1 \stackrel{?}{=} t_3 \stackrel{?}{=} \texttt{bool}$$

again yielding the symptom $\texttt{int} \stackrel{?}{=} \texttt{bool}$.

The system of term equations is represented by a *unification graph*, as shown in Fig. 2. Variables and function symbol occurrences in the system of term equations are vertices in the unification graph. Thick *branch edges* distinguished with solid arrowheads represent the relation between a term and its immediate subterms. Thin *equational edges* distinguished with open arrowheads represent equations between terms. Each equational edge is oriented in an arbitrary direction. Solid edges are *original* equations. Dashed edges are *derived* equations, that is, equations which are inferred by the process of unification (j and k in the example).

Unification may be viewed as establishing special connectivity relations between vertices in the unification graph. In the example, a proof for the derivation of the unsolvable equation

$$\texttt{int} \stackrel{?}{=} \texttt{bool}$$

from the original set of equations of Fig. 1 may be viewed as a path connecting the vertices `int` and `bool` in the unification graph, with the assumption that edges may be traversed in either direction. Traversal of an edge y in a direction opposite to its orientation is denoted y^{-1}. Thus, the paths $j^{-1}ie^{-1}c$ and $k^{-1}dge^{-1}c$ between `int` and `bool` in Fig. 2 are both witnesses to the insolubility of the given system of term equations. The edges j and k owe their existence to the downward-closure rule and the connectivity of the $\rightarrow$ nodes via edges f and h. Thus, j is derived from the path $p^{-1}f^{-1}hr$ consisting solely of original constraints. Similarly, k is derived from the path $q^{-1}f^{-1}hs$.

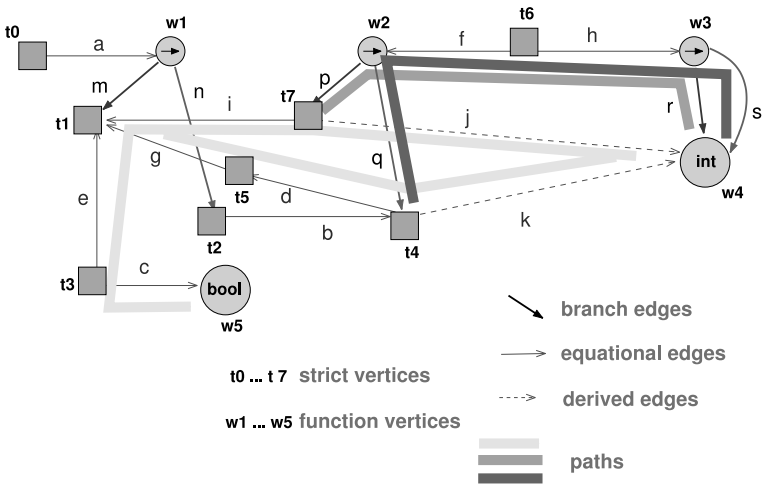

Fig. 2. Unification graph representing term equations of Fig. 1.

Replacing j with $p^{-1}f^{-1}hr$ in the path $j^{-1}ie^{-1}c$ connecting `int` with `bool` yields the path $(p^{-1}f^{-1}hr)^{-1}ie^{-1}c$. This path, containing only original edges, simplifies to $r^{-1}h^{-1}fpie^{-1}c$. Similarly, replacing k with $q^{-1}f^{-1}hs$ in the path $k^{-1}dge^{-1}c$ and simplifying yields $s^{-1}h^{-1}fqdge^{-1}c$. These two paths are different insolubility diagnoses for the original type constraints. Furthermore, these paths are minimal: no other path consisting of a proper subset of the edges in these paths connects `int` to `bool`.

2.1. Slices from paths

Programmers think in terms of *program slices* when debugging errors in their programs [56,57]. Program slices are subparts of the program that need to be fixed to remove the error. If we consider sets of term equations as programs and symptoms of non-unifiability as errors, then it is reasonable to ask how to compute program slices of a system of non-unifiable term equations that derive the symptom of non-unifiability of the term equations.

Each of the paths $r^{-1}h^{-1}fpie^{-1}c$ and $s^{-1}h^{-1}fqdge^{-1}c$ defines a program slice of the set of equations of Fig. 1 that derive the symptom $\texttt{int} \stackrel{?}{=} \texttt{bool}$. The set of edges $\{r, h\}$ corresponds to the *weakening* $t_6 \stackrel{?}{=} \texttt{int} \to \square$ of the constraint $h : t_6 \stackrel{?}{=} \texttt{int} \to \texttt{int}$. The $\square$ represents a "hole" indicating that the second occurrence of `int` is irrelevant. Each occurrence of a hole is interpreted as a variable not occurring elsewhere in the set of equations. Similarly, $\{f, p\}$ corresponds to the weakening $t_6 \stackrel{?}{=} t_7 \to \square$ obtained by replacing t_4 with $\square$ in f. The path $r^{-1}h^{-1}fpie^{-1}c$ consisting of segments $r^{-1}h^{-1}$ and fp, along with edges i, e, and c, therefore corresponds to the following set of minimally non-unifiable type equations:

$$t_6 \stackrel{?}{=} \texttt{int} \to \square \qquad t_6 \stackrel{?}{=} t_7 \to \square \qquad t_7 \stackrel{?}{=} t_1$$
$$t_3 \stackrel{?}{=} t_1 \qquad t_3 \stackrel{?}{=} \texttt{bool}.$$

Similarly, the set of minimally non-unifiable type equations derived from the path $s^{-1}h^{-1}fqdge^{-1}c$ are

$$t_6 \stackrel{?}{=} \square \to \texttt{int} \qquad t_6 \stackrel{?}{=} \square \to t_4 \qquad t_4 \stackrel{?}{=} t_5$$
$$t_5 \stackrel{?}{=} t_1 \qquad t_3 \stackrel{?}{=} t_1 \qquad t_3 \stackrel{?}{=} \texttt{bool}.$$

Thus, a symptom of unification failure may have its origin in multiple program slices. Each program slice is derived from a specially labeled-path in the unification graph. Each such specially labeled-path, therefore, encodes a program slice. The definition, derivation, and simplification of these paths is the focus of this paper.

3. Basic definitions and notation

We begin the main part of this paper by formalizing the basic machinery of unification in terms of labeled directed graphs and noting some elementary properties of labeled-paths. Formal definitions for term graphs, unification graphs, labeled graphs, unification closure and quotient graphs are then presented using labeled directed graphs. The fundamental result of Paterson and Wegman [43] relating unification closure and unifiability is recalled. Finally, the notation used in the rest of the paper is summarized for easy reference.

3.1. Labeled directed graphs

A *labeled directed graph* G is a triple $\langle \Sigma, V, D\rangle$, where Σ is an alphabet, V is a set of vertices, and $D \subseteq V \times V \times (\Sigma \cup \{\epsilon\})$ is the set of *labeled directed edges of* G. The triple $\langle u, v, \eta\rangle \in D$, written $u \xrightarrow{\eta} v$, denotes an edge *from* u *to* v whose *label* is η. The function l projects the label η from an edge $u \xrightarrow{\eta} v$. Edges with labels in Σ are *branch edges* and those with label ϵ are *equational edges*. We use the convention that δ ranges over symbols in Σ, whereas η ranges over $\Sigma \cup \{\epsilon\}$. An equational edge from a vertex to itself is called a *trivial edge*.

We overload the labeling function $l : D \to \Sigma \cup \{\epsilon\}$ to denote its homomorphic extension $l : D^* \to \Sigma^*$. The following observations relating sequence of edges with the decomposition of their labels are natural consequences of this extension. These observations will be used later to prove the completeness of the logic of unification deductions (Lemma 15).

Lemma 1 (Decomposition)**.** *Let* $G = \langle \Sigma, V, D\rangle$ *be a labeled directed graph and let* $s, s_1, s_2 \in D^*$ *be sequences of edges,* b *a branch edge in* D*, and* δ *a symbol in* Σ.

(1) $s = s_1 s_2$ *if and only if* $l(s) = l(s_1)l(s_2)$.
(2) s *contains a branch edge* $b \in D$ *such that* $l(b) = \delta$ *if and only if* $l(s)$ *contains the symbol* δ *as a substring*.

Proof. Follows from the definition of the homomorphic extension of the label function l. □

$$\text{INIT}\quad \frac{}{G \models u \xrightarrow{\eta} v : u \xrightarrow{\eta} v} \qquad u \xrightarrow{\eta} v \in D$$

$$\text{REF}\quad \frac{}{G \models \epsilon : u \xrightarrow{\epsilon} u} \qquad u \in V$$

$$\text{TRANS}\quad \frac{G \models p : u \xrightarrow{l} v' \qquad G \models q : v' \xrightarrow{l'} v}{G \models pq : u \xrightarrow{ll'} v} \qquad \begin{matrix} u, v', v \in V \\ l, l' \in \Sigma^* \end{matrix}$$

Fig. 3. The rules defining labeled-paths over a labeled directed graph $G = \langle \Sigma, V, D \rangle$.

3.2. Labeled-paths

Our analysis of unification source-tracking is formulated in terms of paths over labeled directed graphs and their labels. Let $G = \langle \Sigma, V, D \rangle$ be a labeled directed graph. The set of *labeled-paths in G* are inductively defined using the rules of Fig. 3. The judgement $G \models p : u \xrightarrow{l} v$ denotes that *p is a labeled-path in G from u to v with label l*. The judgement $G \models u \xrightarrow{l} v$ denotes that there is a path from u to v labeled l in G. The label of an empty path is ϵ. Because an edge in a labeled directed graph may be labeled with ϵ, non-empty paths may have empty labels.

The following properties relate the decomposition of paths with the decomposition of their labels. They are straightforward to verify, and are recorded here for later use in proving soundness properties of unification paths, defined and discussed in Section 4.2.

Lemma 2 (Labeled-path decomposition)**.** *Let $G = \langle \Sigma, V, D \rangle$ be a labeled directed graph.*

(1) $G \models qr : u \xrightarrow{l} v$ *if and only if for some vertex* $v' \in V$ *and label strings* $l_q, l_r \in \Sigma^*$, $G \models q : u \xrightarrow{l_q} v'$, $G \models r : v' \xrightarrow{l_r} v$, *and* $l = l_q l_r$.
(2) $G \models p : u \xrightarrow{l_q l_r} v$ *if and only if for some vertex* $v' \in V$ *and some labeled-paths q and r in G*, $G \models q : u \xrightarrow{l_q} v'$, $G \models r : v' \xrightarrow{l_r} v$, *and* $p = qr$.
(3) $G \models p : u \xrightarrow{l_q \delta l_r} v$, *where* $\delta \in \Sigma$ *and* $l_q, l_r \in \Sigma^*$ *if and only if there are vertices u′ and v′ in V, a branch edge* $b \in D$ *from u′ to v′ labeled* δ, *and paths q and r in G such that* $p = qbr$, $G \models q : u \xrightarrow{l_q} u'$, *and* $G \models r : v' \xrightarrow{l_r} v$.

Proof.

(1) Follows immediately from the definition of labeled-paths.
(2) The if part follows from the TRANS rule of labeled-paths. For the only if part, by Lemma 1, $p = qr$ for some edge sequences q and r such that $l(q) = l_q$ and $l(r) = l_r$. Since p is a path, it follows that q and r are paths. In other words, there is a vertex v' such that $G \models q : u \xrightarrow{l_q} v'$, $G \models r : v' \xrightarrow{l_r} v$, which finishes the proof.

(3) The if part follows from one application of the INIT rule and two applications of the TRANS rule of Fig. 3. For the only if part, by Lemma 1, there are vertices u' and v' and a branch edge b from u' to v' labeled δ such that b is a substring of p. Therefore, $p = qbr$ for some edge sequences q and r such that $l(q) = l_q$ and $l(r) = l_r$. Since p is a path, it follows that q and r are paths. Therefore, $G \models q : u \xrightarrow{l_q} u'$ and $G \models r : v' \xrightarrow{l_r} v$, which completes the proof. □

3.3. Unification graphs

Given an alphabet Σ and $\epsilon \notin \Sigma$, let Σ^0, Σ^+, and Σ^* denote, respectively, the set $\{\epsilon\}$ containing the empty word ϵ, the set of finite non-empty words over Σ, and the set of finite words over Σ. A *signature* is an alphabet Σ of *function symbols*, graded by an arity function $\alpha : \Sigma \to \mathbf{N}$, where $\mathbf{N}$ denotes the set of natural numbers. Function symbols of arity 0 are called *constants*.

A family of terms over a signature Σ and a set of variables $\mathbf{V}$ (Σ-*terms* over $\mathbf{V}$) is represented using a Σ-*term graph* $T = \langle W, X, \rho \rangle$, where W is a set of *function vertices* disjoint from $\mathbf{V}$, $X \subset \mathbf{V}$ is a set of *strict vertices*, and $\rho : W \longrightarrow \Sigma(W \cup X)$ is the *subterm function*, where for any set A, $\Sigma(A)$ is the set of terms $f(a_1, \ldots, a_{\alpha(f)})$ such that $f \in \Sigma$, and, for each i such that $1 \leqslant i \leqslant \alpha(f)$, $a_i \in A$. Strict vertices represent variables in a family of terms, whereas function vertices represent non-variable subterms. If w is a function vertex and $\rho(w) = f(u_1, \ldots, u_n)$, then the *label* of w, denoted $L(w)$, is defined to be f.

Pictorially, the graph representation of the Σ-term graph $T = \langle W, X, \rho \rangle$ consists of a set of vertices and edges. The set of vertices is $W \cup X$. For each $w \in W$, if $\rho(w) = f(u_1, \ldots, u_n)$, where $f \in \Sigma$ and $n = \alpha(f)$, there is a *branch* edge from w to u_i, for each i, $1 \leqslant i \leqslant n$.

Thus, the term graph is a formalization of the graph representation of terms. This formalization is closer to the actual implementation of efficient data structures for terms because it makes explicit the sharing assumptions of vertices: variables of a term are always shared, and non-variable subterms *may* be shared.

A *term equation* is a symmetric relation on terms. Sets of term equations are represented using unification graphs. A Σ-*unification graph* G is a pair $\langle T, E \rangle$, where T is a Σ-term graph $\langle W, X, \rho \rangle$ representing the family of terms occurring in the set of term equation, and E is a relation over $W \cup X$. Pictorially, G extends the graph T with a set of *equational* edges such that for each $(u, v) \in E$, there is an undirected edge in G. When Σ is clear from the context, 'Σ-term,' 'Σ-term graph,' and 'Σ-unification graph' are abbreviated 'term,' 'term graph,' and 'unification graph,' respectively.

If Σ is a signature, then $\{f_i \mid f \in \Sigma, 1 \leqslant i \leqslant \alpha(f)\}$, denoted $\Sigma_{\mathbf{N}}$, is called the *projection alphabet of* Σ. To avoid multiple levels of subscripting, the symbol f_i is sometime written $f.i$.

The labeled directed graph *underlying* a Σ-unification graph $G = \langle \langle W, X, \rho \rangle, E \rangle$ is $\langle \Sigma_{\mathbf{N}}, V, D \rangle$, where $V = W \cup X$, and D is the union of the set of branch edges $\{w \xrightarrow{f.i} u_i \mid \rho(w) = f(u_1, \ldots, u_i, \ldots u_n)\}$ and equational edges $\{u \xrightarrow{\epsilon} v \mid (u, v) \in E\}$. When there is no ambiguity, the labeled directed graph underlying a Σ-unification graph G is also denoted G. Note, however, that while the function vertices of a unification graph are labeled, in the underlying labeled directed graph, it is edges, and not vertices, that are labeled.

Example 3. The set of term equations of Fig. 1 is represented by the Σ-unification graph $G = \langle T, E\rangle$, where Σ consists of the binary (infix) function symbol $\rightarrow$ and the constants `bool` and `int`, and $T = \langle W, X, \rho\rangle$ is the term graph representing the family of terms, $W = \{w_1, w_2, w_3, w_4, w_5\}$, $X = \{t_0, \ldots, t_7\}$, and ρ is defined as

$$\begin{aligned}
&\rho(w_1) = t_1 \rightarrow t_2\\
&\rho(w_2) = t_7 \rightarrow t_2\\
&\rho(w_3) = w_4 \rightarrow w_4\,,\\
&\rho(w_4) = \texttt{int}\\
&\rho(w_5) = \texttt{bool}
\end{aligned}$$

where E, the set of equational edges, is

$$\begin{array}{lll}
a : t_0 \stackrel{?}{=} w_1 & b : t_2 \stackrel{?}{=} t_4 & c : t_3 \stackrel{?}{=} w_5\\
d : t_4 \stackrel{?}{=} t_5 & e : t_3 \stackrel{?}{=} t_1 & f : t_6 \stackrel{?}{=} w_2\\
g : t_5 \stackrel{?}{=} t_1 & h : t_6 \stackrel{?}{=} w_3 & i : t_7 \stackrel{?}{=} t_1.
\end{array}$$

The labeled directed graph underlying G is $\langle \Sigma_{\mathbf{N}}, V, D\rangle$, where $\Sigma_{\mathbf{N}}$ is the projection alphabet $\{\rightarrow_1, \rightarrow_2\}$, V is the set of vertices given above, and D is the following set of labeled directed edges:

$$\begin{array}{lll}
a : t_0 \stackrel{\epsilon}{\longrightarrow} w_1 & b : t_2 \stackrel{\epsilon}{\longrightarrow} t_4 & c : t_3 \stackrel{\epsilon}{\longrightarrow} w_5\\
d : t_4 \stackrel{\epsilon}{\longrightarrow} t_5 & e : t_3 \stackrel{\epsilon}{\longrightarrow} t_1 & f : t_6 \stackrel{\epsilon}{\longrightarrow} w_2\\
g : t_5 \stackrel{\epsilon}{\longrightarrow} t_1 & h : t_6 \stackrel{\epsilon}{\longrightarrow} w_3 & i : t_7 \stackrel{\epsilon}{\longrightarrow} t_1\\
m : w_1 \stackrel{\rightarrow_1}{\longrightarrow} t_1 & n : w_1 \stackrel{\rightarrow_2}{\longrightarrow} t_2 & p : w_2 \stackrel{\rightarrow_1}{\longrightarrow} t_7\\
q : w_2 \stackrel{\rightarrow_2}{\longrightarrow} t_4 & r : w_3 \stackrel{\rightarrow_1}{\longrightarrow} w_4 & s : w_3 \stackrel{\rightarrow_2}{\longrightarrow} w_4.
\end{array}$$

Fig. 2 is a "hybrid" picture that represents both a unification graph and the underlying labeled directed graph. In Fig. 2, the unification graph's strict vertices and function vertices are represented by boxes and circles, respectively. The dashed edges are not really part of the unification graph; they are present only to indicate the derived connectivity between subterm vertices. Branch edges are represented by thick edges with closed, solid arrowheads. The thin edges represent equational edges whose directions should be ignored.

To interpret Fig. 2 as the labeled directed graph underlying the unification graph, we consider the orientation of all the edges (including the equational edges). To avoid clutter, the labels on the edges are not shown, but are assumed ϵ for the equational edges and $\rightarrow_1$ (respectively, $\rightarrow_2$) for each branch edge emanating from the left (respectively, right) of a vertex. The labels on the vertices, although shown (because they are part of the definition of a unification graph), are irrelevant to the labeled directed graph.

Given a labeled directed graph $G = \langle \Sigma, V, D\rangle$, a relation R on V is *downward-closed* if for each $\delta \in \Sigma$ and uRu', $u \stackrel{\delta}{\longrightarrow} v \in D$ and $u' \stackrel{\delta}{\longrightarrow} v' \in D$ implies vRv'. If C is an equivalence relation on the vertices of G, then the *unification closure of* C is the least downward-closed equivalence on the vertices of G containing R. The *unification closure of* G, denoted $\sim$, is the unification closure of

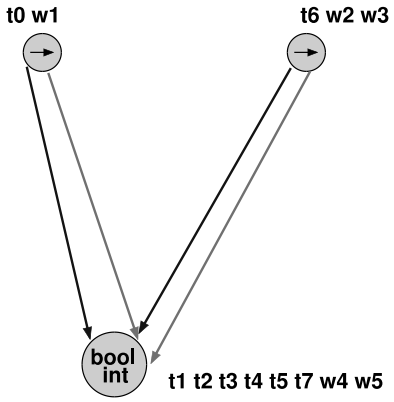

Fig. 4. Quotient graph of the labeled directed graph underlying the unification graph of Fig. 2.

the equational edges of G. $G \models u \sim v$ denotes that (u, v) is an element of $\sim$, the unification closure of G. The *unification quotient of* G is the labeled directed graph $G/\sim = \langle \Sigma, V/\sim, D/\sim \rangle$, where $V/\sim$ is the set of equivalence classes of $\sim$, and for all vertices $u, v \in V$ and $\delta \in \Sigma$, $[u]_\sim \xrightarrow{\delta} [v]_\sim \in D/\sim$ if $u \xrightarrow{\delta} v \in D$. For a unification graph $G = \langle W, X, \rho, E \rangle$, the unification closure of its underlying labeled directed graph is *homogeneous* if for each $w, w' \in W$, $w \sim w'$ implies $L(w) = L(w')$.

Example 4. Fig. 4 shows the quotient graph of the labeled directed graph underlying the unification graph of Fig. 2. Each vertex is an equivalence class of the unification closure of the unification graph of Fig. 2. Although labels on vertices are not part of the labeled directed graph, each vertex is annotated with the set of all the labels of all its constituent vertices from the original unification graph. Because the unification closure contains the pair of function vertices w_4 labeled `bool` and w_5 labeled `int`, the unification graph of Fig. 2 is not homogeneous.

The following fundamental result of Paterson and Wegman [43] relates the unifiability of a system of term equations with the unification closure of the labeled directed graph underlying the unification graph representing the system of term equations.

Theorem 5 (Paterson and Wegman 1978 [43])**.** *The set of term equations represented by a unification graph G is unifiable if and only if the unification closure of its underlying labeled directed graph is homogeneous and the quotient graph $G/\sim$ is cycle-free.*

Example 6. The unification closure computed in Example 4 is non-homogeneous. Applying Paterson and Wegman's result, we conclude that the equations represented unification graph G of Fig. 2 are non-unifiable.

A homogeneous and cycle-free quotient can be used to read off the most general unifier of a system of term equations. However, we will not be concerned with most general unifiers in this paper. Instead, using the basic formalism of this section as the starting point, we develop a more precise and general analysis relating the labeled directed graph and its quotient independent of the unifiability or non-unifiability of the unification graph.

Before we turn to this task, however, we summarize the notation used throughout the rest of the paper.

Notation

Some of the symbols are overloaded with meanings defined later in the paper. For overloaded symbols, usually the meaning will be clear (or will be made clear) from the context.

- Σ is used to denote an alphabet and also a signature.
- f, g, h denote function symbols. x, y, z, t denote variables. Terms are denoted by τ and also sometimes by t.
- T is used to denote a term graph. G denotes a unification graph and also a directed labeled graph.
- E denotes sets of term equations, and also the set of equational edges in a unification graph.
- l denotes the labeling function on edges and also its homomorphic extension on sequence of edges. l is also used to indicate a sequence of labels.
- u, v, r denote vertices of a term graph. x, y denote strict vertices and w denotes a function vertex. u, v denote vertices that may either be strict or function vertices.
- a denotes an equational edge. b denotes a branch edge. c denotes a directed (equational or branch) edge.
- η denotes the label of a directed edge (branch or equational). δ denotes the label of a branch edge.
- s denotes a sequence of edges of a graph.
- p, q, r denote labeled-paths in a graph. They are also used to denote unification expressions, introduced in Section 5.

In addition, subscripted and primed variants of the above meta-variables are also used.

4. Source-tracking

We are now ready to state the first result of this paper, which characterizes connectivity in the quotient graph $G/\sim$ in terms of a special connectivity relation over G. This is the basis for unification source-tracking because we can track paths in the quotient graph in terms of their "source" paths in the original graph. A precise characterization of source-tracking is based on an analysis of the formal language properties of the labels of paths on unification graphs and their quotients to which we turn next.

4.1. Semi-Dyck and Dyck sets

Semi-Dyck and Dyck sets are parentheses languages. These languages have nice cancellation properties. As will be shown in the next section, this cancellation phenomenon is at the heart of unification closure.

Given an alphabet $\Sigma = \{\delta_1, \ldots, \delta_n\}$, Σ^{-1} is the alphabet $\{b_i^{-1} \mid \delta_i \in \Sigma\}$, assumed disjoint from Σ. $\mathbf{S}(\Sigma)$ and $\mathbf{S_2}(\Sigma)$ denote the sets $\{\delta^{-1}\delta \approx \epsilon \mid \delta \in \Sigma\}$ and $\{\delta\delta^{-1} \approx \epsilon, \delta^{-1}\delta \approx \epsilon \mid \delta \in \Sigma\}$ of one-way and two-way cancellation identities, respectively. Let $\approx_{\mathbf{S}(\Sigma)}$ and $\approx_{\mathbf{S_2}(\Sigma)}$ denote the smallest congruences over $(\Sigma \cup \Sigma^{-1})^*$ containing the identities $\mathbf{S}(\Sigma)$ and $\mathbf{S_2}(\Sigma)$, respectively. Intuitively, $u \approx_{\mathbf{S}(\Sigma)} v$ if u

may be obtained from v via applications of the identities in $\mathbf{S}(\Sigma)$ (or vice versa), and likewise for $u \approx_{\mathbf{S_2}(\Sigma)} v$. The equational systems $\mathbf{S}(\Sigma)$ and $\mathbf{S_2}(\Sigma)$ may be turned into rewrite systems by orienting each of the identities $\delta^{-1}\delta \approx \epsilon$ and $\delta\delta^{-1} \approx \epsilon$ as $\delta^{-1}\delta \longrightarrow \epsilon$ and $\delta\delta^{-1} \longrightarrow \epsilon$ from left to right, resulting in strongly normalizing rewrite systems. One step rewriting under $\mathbf{S}(\Sigma)$ and $\mathbf{S_2}(\Sigma)$ is denoted $\longrightarrow_{\mathbf{S}(\Sigma)}$ and $\longrightarrow_{\mathbf{S_2}(\Sigma)}$, respectively. If $l \in (\Sigma \cup \Sigma^{-1})^*$, let $\mu_{\mathbf{S}(\Sigma)}(l)$ and $\mu_{\mathbf{S_2}(\Sigma)}(l)$ denote the unique normal forms under $\mathbf{S}(\Sigma)$ and $\mathbf{S_2}(\Sigma)$ rewriting, respectively. If $L \subseteq (\Sigma \cup \Sigma^{-1})^*$, let

$$S(\Sigma, L) \stackrel{\text{def}}{=} \{l \in (\Sigma \cup \Sigma^{-1})^* \mid \mu_{\mathbf{S}(\Sigma)}(l) \in L\},$$
$$S_2(\Sigma, L) \stackrel{\text{def}}{=} \{l \in (\Sigma \cup \Sigma^{-1})^* \mid \mu_{\mathbf{S_2}(\Sigma)}(l) \in L\}.$$

Intuitively, $S(\Sigma, L)$ (respectively, $S_2(\Sigma, L)$) denotes the parentheses language of words which, after one-way (respectively, two-way) cancellation, reduce to words in L. Obviously, $S(\Sigma, L)$ is a subset of $S_2(\Sigma, L)$. We are primarily interested in the languages $S(\Sigma, L)$ and $S_2(\Sigma, L)$ when L is Σ^0, Σ^+, and Σ^*. These are abbreviated $S^0(\Sigma)$, $S^+(\Sigma)$, and $S^*(\Sigma)$, and $S_2^0(\Sigma)$, $S_2^+(\Sigma)$, and $S_2^*(\Sigma)$, respectively. $S^0(\Sigma)$ is known as the *semi-Dyck set over* Σ. It is the set of balanced parentheses words whose open and closed parenthesis symbols are drawn from Σ^{-1} and Σ, respectively.[1]

Clearly, $S^0(\Sigma)$ and $S^+(\Sigma)$ are disjoint and $S^* = S^0 \cup S^+$. The language S^* is the set of all suffixes of S^0 words, and is suffix-closed. (See [25], p. 314, for the symmetric case of prefixes of S^0 words.) Informally, S^+ is the set of "unbalanced" suffixes of words of balanced parentheses. The languages S^0, S^+, and S^* may be generated using the following context-free grammar:

$$\begin{aligned} S^0 &::= \epsilon \mid S^0\delta^{-1}S^0\delta S^0 && \delta \in \Sigma \\ S^+ &::= S^*\delta S^* && \delta \in \Sigma \\ S^* &::= S^0 \mid S^+ \end{aligned}$$

One-sided reduction will assume special significance in our analysis. If $l \in (\Sigma \cup \Sigma^{-1})^*$, the normal form of l under one sided cancellation, that is $\mu_{\mathbf{S}(\Sigma)}(l)$ is called the *signature of* l.[2]

The next lemma—a direct consequence of the context-free grammar specification of semi-Dyck sets—formalizes the intuition that for every $S^*(\Sigma)$ word l, every open parenthesis occurring in l is matched, and every closed parenthesis is either matched, or unmatched. In the latter case, l is in $S^+(\Sigma)$.

Lemma 7 (Matching)**.** *Let* Σ *be an alphabet and let* $l \in S^*(\Sigma)$.

(1) *If* $l = l_1\delta^{-1}l_2$ *for some* δ *in* Σ *and* l_1, l_2 *in* $(\Sigma \cup \Sigma^{-1})^*$, *then* $l_2 = l_3\delta l_4$ *for some* l_3 *in* $S^0(\Sigma)$ *and* l_4 *in* $(\Sigma \cup \Sigma^{-1})^*$.
(2) *If* $l = l_1\delta l_2$, *for some* δ *in* Σ *and* l_1, l_2 *in* $(\Sigma \cup \Sigma^{-1})^*$, *then*
   (*a*) $l_1 = l_3\delta^{-1}l_4$ *for some* l_3 *in* $(\Sigma \cup \Sigma^{-1})^*$ *and* l_4 *in* $S^0(\Sigma)$, *or*
   (*b*) l_1, l_2 *are both in* $S^*(\Sigma)$, *implying* l *is in* $S^+(\Sigma)$.

[1] The reader will have to wait till Section 4.2 to appreciate the usefulness of this convention, which is counter to the intuition of drawing left parentheses from Σ and the right from Σ^{-1}.

[2] This is not to be confused with the signature consisting of an alphabet and an arity function.

Proof. By induction on the derivation of S^* words. □

Example 8. Consider the signature $\Sigma = \{\mathsf{f} \mapsto 2, \mathsf{g} \mapsto 2, \mathsf{h} \mapsto 1, \mathsf{a} \mapsto 0\}$. The projection alphabet $\Sigma_{\mathbf{N}}$ consists of the closed parenthesis symbols

$$\{\mathsf{f}_1, \mathsf{f}_2, \mathsf{g}_1, \mathsf{g}_2, \mathsf{h}_1\}.$$

The corresponding open parenthesis symbols are

$$\{\mathsf{f}_1^{-1}, \mathsf{f}_2^{-1}, \mathsf{g}_1^{-1}, \mathsf{g}_2^{-1}, \mathsf{h}_1^{-1}\}.$$

The following are examples of words in various subsets of $(\Sigma_{\mathbf{N}} \cup \Sigma_{\mathbf{N}}^{-1})^*$.

ϵ	S^0
f_1	S^+
f_1^{-1}	$(\Sigma_{\mathbf{N}} \cup \Sigma_{\mathbf{N}}^{-1})^* - S_2^*$
$\mathsf{f}_1\mathsf{f}_1^{-1}$	$S_2^0 - S^0$
$\mathsf{f}_1^{-1}\mathsf{f}_1$	S^0
$\mathsf{f}_1^{-1}\mathsf{g}_2^{-1}\mathsf{f}_1$	$(\Sigma_{\mathbf{N}} \cup \Sigma_{\mathbf{N}}^{-1})^* - S_2^*$.

Dyck sets are closely related to groups. The quotient monoid $(\Sigma \cup \Sigma^{-1})^* / \approx_{\mathbf{S_2}(\Sigma)}$ has ${S_2}^0(\Sigma)$ as its identity element. If $l \in (\Sigma \cup \Sigma^{-1})^*$, let $[l]$ denote the equivalence class of the quotient containing the word l. By defining a unary inverse operation $[l]^{-1} = [l']$, where $ll' \approx_{\mathbf{S_2}(\Sigma)} \epsilon$, the quotient monoid can be turned into a group. This group is isomorphic to the free group generated by Σ. The isomorphism is witnessed by the function f mapping each class $[l]$ of $\approx_{\mathbf{S_2}(\Sigma)}$ containing l to the class of the free group containing l with the property that $[l] \subseteq f([l])$. For each word l over $\Sigma \cup \Sigma^{-1}$, the *inverse of* l $inv(l)$ maps l to an element of $[l]^{-1}$ and is defined recursively as follows: $inv(\epsilon) = \epsilon$, $inv(\delta) = \delta^{-1}$ if $\delta \in \Sigma$, $inv(\delta^{-1}) = \delta$, if $\delta^{-1} \in \Sigma$, and $inv(ll') = inv(l')inv(l)$.

The central role of semi-Dyck and Dyck sets in the source-tracking of unification is brought to fore in the next subsection using the idea of unification paths, which is introduced next.

4.2. Unification paths

To understand unification in terms of path connectivity, it is necessary to consider the traversal of edges in the unification graph in both forward and reverse directions. This is formalized by defining the inverse of a labeled directed graph. The *inverse* G^{-1} of a labeled directed graph $G = \langle \Sigma, V, D \rangle$ is the labeled directed graph $\langle \Sigma^{-1}, V, D^{-1} \rangle$, where $D^{-1} = \{v \xrightarrow{\delta^{-1}} u \mid u \xrightarrow{\delta} v \in D\}$, and $\epsilon^{-1} = \epsilon$. D^{-1} is the set of *inverse edges*. It is natural to consider extending the operation of inverting edges to edge sequences using the *inv* operation defined as before but operating on $(D \cup D^{-1})^*$. The labels of a path and its inverse are related in the following way, which is again straightforward to show:

Lemma 9 (Inverses of labeled-paths)**.** *Let $G = \langle \Sigma, V, D \rangle$ be a labeled directed graph.*

(1) *($S^0(\Sigma)$ labels) If $l \in S^0(\Sigma)$, then $inv(l) = l$.*

(2) *(Inverse)* $G \cup G^{-1} \models p : u \xrightarrow{l} v$ *if and only if* $G \cup G^{-1} \models inv(p) : v \xrightarrow{l'} u$, *where* $l' = inv(l)$.

Proof. Straightforward consequence of the definitions of *inv* and labeled-paths. □

We now define a special class of a labeled-paths called *unification paths*. These are paths whose labels are elements of semi-Dyck sets.

Definition 10 (*Unification paths*)**.** Let $G = \langle \Sigma, V, D \rangle$ be a labeled directed graph and let $u, v \in V$. A *unification path from u to v over G* is a path p in $G \cup G^{-1}$ such that $\mu_{\mathbf{S}(\Sigma)}(l(p)) \in \Sigma^*$.

If $G = \langle \Sigma, V, D \rangle$ is a labeled directed graph and p is a labeled-path in $G \cup G^{-1}$, then the *signature of p* is defined to be the signature $\mu_{\mathbf{S}(\Sigma)}(l(p))$ of the label of p.

The structure of unification paths follows the nesting of parentheses. The label of a unification path over a graph $G = \langle \Sigma, V, D \rangle$ is an element of the suffix language $S^*(\Sigma)$. Intuitively, a unification path is obtained by traversing any of the edges of a labeled directed graph, or by concatenating two unification paths, or by an upstream traversal on a branch edge labeled δ followed by a unification path whose label is balanced, followed by a downstream traversal on an identically labeled branch edge. The signature of the upward traversal of the branch edge is δ^{-1}, which is akin to an opening left parenthesis, whereas the corresponding traversal downward earns the signature δ which corresponds to a closing right parenthesis.

Fig. 5 shows instances and non-instances of unification paths over the unification graph of Fig. 2. The labels of labeled directed graph underlying the graph of Fig. 2 are drawn from the projection alphabet $\{\rightarrow_1, \rightarrow_2\}$. Note that the signature of unification paths over this labeled directed graph is always a word over the projection alphabet.

path	label	signature	unification path?
p	$l(p)$	$\mu_{\mathbf{S}(\Sigma)}(l(p))$	yes/no
a	ϵ	ϵ	yes
a^{-1}	ϵ	ϵ	yes
fqb^{-1}	$\rightarrow_2$	$\rightarrow_2$	yes
$p^{-1}fhr$	$\rightarrow_1^{-1}\rightarrow_1$	ϵ	yes
$amg^{-1}d^{-1}q^{-1}f^{-1}hs$	$\rightarrow_1\rightarrow_2^{-1}\rightarrow_2$	$\rightarrow_1$	yes
$m^{-1}m$	$\rightarrow_1^{-1}\rightarrow_1$	ϵ	yes
m^{-1}	$\rightarrow_1^{-1}$	$\rightarrow_1^{-1}$	no
mm^{-1}	$\rightarrow_1\rightarrow_1^{-1}$	$\rightarrow_1\rightarrow_1^{-1}$	no
$p^{-1}q$	$\rightarrow_1^{-1}\rightarrow_2$	$\rightarrow_1^{-1}\rightarrow_2$	no
nbq^{-1}	$\rightarrow_2^{-1}\rightarrow_2$	$\rightarrow_2^{-1}\rightarrow_2$	no

Fig. 5. Instances and non-instances of unification paths over the graph of Fig. 2.

The name "unification paths" is justified because connectivity in the unification quotient graph is witnessed by connectivity via unification paths. The rest of this section makes this notion precise.

The soundness property (Theorem 11) ensures that unification paths capture connectivity in the quotient graph. The completeness property (Theorem 14) verifies that connectivity in the quotient graph is witnessed by an appropriate unification path in the source graph.

Theorem 11 (Soundness of unification paths)**.** *Let G be a labeled directed graph $\langle \Sigma, V, D\rangle$ whose unification closure is $\sim$. If $G \cup G^{-1} \models u \xrightarrow{l} v$ and $l \in S^*(\Sigma)$, then $G/\sim \models [u] \xrightarrow{\mu_{\mathbf{S}(\Sigma)}(l)} [v]$.*

Proof. Let $G \cup G^{-1} \models p : u \xrightarrow{l} v$. The proof is by induction on the derivation of l in the grammar $S^*(\Sigma)$. For each case, we construct a path p' such that $G/\sim \models p' : [u] \xrightarrow{\mu_{\mathbf{S}(\Sigma)}(l)} [v]$.

(1) $l = \epsilon$: Every edge of p is labeled ϵ. Therefore, $G \models u \sim v$, and $G/\sim \models \epsilon : [u] \xrightarrow{\epsilon} [v]$. The result follows.

(2) $l = l_1\delta^{-1}l_2\delta l_3$, where $\delta \in \Sigma$, and $l_1, l_2, l_3 \in S^0(\Sigma)$. By Lemma 2, there are vertices u_1, u_2, u_3, u_4, branch edges b_1, b_2, and paths p_1, p_2, and p_3, such that $p = p_1 b_1^{-1} p_2 b_2 p_3$, and

$$G \cup G^{-1} \models p_1 : u \xrightarrow{l_1} u_1$$
$$b_1 : u_2 \xrightarrow{\delta} u_1 \in D$$
$$G \cup G^{-1} \models p_3 : u_2 \xrightarrow{l_2} u_3$$
$$b_2 : u_3 \xrightarrow{\delta} u_4 \in D$$
$$G \cup G^{-1} \models p_3 : u_4 \xrightarrow{l_3} v$$

The normal forms of l_1, l_2, and l_3 under $\mathbf{S}(\Sigma)$ are all equal to ϵ. Applying the induction hypothesis, $G/\sim \models [u] \xrightarrow{\epsilon} [u_1]$. Thus, $[u] = [u_1]$. Similarly, $[u_2] = [u_3]$, and $[u_4] = [v]$. Due to the downward closure of $\sim$, $[u_1] = [u_4]$, implying $[u] = [v]$. It follows that $G/\sim \models [u] \xrightarrow{\epsilon} [v]$.

(3) $l = l_q \delta l_r$, where $\delta \in \Sigma$, and $l_q, l_r \in S^*(\Sigma)$. By Lemma 2, part (2), there are vertices u' and v' in G, a branch edge b from u' to v' in $G \cup G^{-1}$ and paths q and r in $G \cup G^{-1}$ such that $G \cup G^{-1} \models q : u \xrightarrow{l_q} v'$, $G \cup G^{-1} \models r : v' \xrightarrow{l_r} v$. By the induction hypothesis, $G/\sim \models [u] \xrightarrow{\mu_{\mathbf{S}(\Sigma)}(l_q)} [v']$ and $G/\sim \models [v'] \xrightarrow{\mu_{\mathbf{S}(\Sigma)}(l_r)} [v]$. Furthermore, $G/\sim \models [u'] \xrightarrow{\delta} [v']$, because of the branch edge b from u' to v'. By transitivity of labeled-paths, it follows that $G/\sim \models [u] \xrightarrow{\mu_{\mathbf{S}(\Sigma)}(l_q)\delta\mu_{\mathbf{S}(\Sigma)}(l_r)} [v]$. Since $\mu_{\mathbf{S}(\Sigma)}(l_q)\delta\mu_{\mathbf{S}(\Sigma)}(l_r)$ has no $\mathbf{S}(\Sigma)$ redexes, it is equal to $\mu_{\mathbf{S}(\Sigma)}(l_q\delta l_r)$. The result follows. □

By specializing Theorem 11 we conclude that S^0-labeled connectivity is included in the unification closure:

Corollary 12 (Soundness of S^0 unification paths)**.** *Let G be a labeled directed graph $\langle \Sigma, V, D\rangle$ whose unification closure is $\sim$. If $G \cup G^{-1} \models u \xrightarrow{l} v$ and $l \in S^0(\Sigma)$, then $G \models u \sim v$.*

Proof. If $l \in S^0(\Sigma)$, then $\mu_{\mathbf{S}(\Sigma)}(l) = \epsilon$. Also, $G/\sim \models [u] \xrightarrow{\epsilon} [v]$ implies $G \models u \sim v$. □

The next lemma shows that membership in the unification closure $\sim$ of a labeled directed graph G is witnessed by a unification path labeled by a semi-Dyck word over the edge labels of G.

Lemma 13 (Completeness of S^0-labeled unification paths)**.** *Let $G = \langle \Sigma, V, D\rangle$ be a labeled directed graph with unification closure $\sim$. If $G \models u \sim v$, then for some path p in $G \cup G^{-1}$ with label l, $G \cup G^{-1} \models p : u \xrightarrow{l} v$ and $l \in S^0(\Sigma)$.*

Proof. By induction on $\sim$. There are five cases. In each case, we show that $l \in S^0(\Sigma)$.

(1) REF$_\sim$: $u = v$, therefore $G \cup G^{-1} \models \epsilon : u \xrightarrow{\epsilon} u$ and $\epsilon \in S^0(\Sigma)$.
(2) INIT$_\sim$: Let $u \xrightarrow{\epsilon} v \in D$, then $G \cup G^{-1} \models u \xrightarrow{\epsilon} v : u \xrightarrow{\epsilon} v$.
(3) SYM$_\sim$: Let $G \models v \sim u$. By the induction hypothesis, $G \cup G^{-1} \models q : v \xrightarrow{l'} u$ for some path q labeled l' such that $l' \in S^0(\Sigma)$. By Lemma 9 (1), $inv(l') = l'$. By Lemma 9 (2), $G \cup G^{-1} \models inv(q) : u \xrightarrow{l'} v$.
(4) TRANS$_\sim$: Let $u \sim v'$, $v' \sim v$, for some $v' \in V$. By the induction hypothesis, $G \cup G^{-1} \models q : u \xrightarrow{l'} v'$ and $G \cup G^{-1} \models r : v' \xrightarrow{l''} v$ for some paths q, r in G such that $l', l'' \in S^0(\Sigma)$. Then, by the TRANS rule for path construction, $G \cup G^{-1} \models qr : u \xrightarrow{l'l''} v$. Clearly, $l'l'' \in S^0(\Sigma)$.
(5) DN$_\sim$: $u \sim v$ because there are vertices u', v' in V, and edges $u' \xrightarrow{\delta} u$ (abbreviated b) and $v' \xrightarrow{\delta} v$ (abbreviated b') in D such that $G \models u' \sim v'$. By the induction hypothesis, $G \cup G^{-1} \models q : u' \xrightarrow{l'} v'$ for some path q with label l' such that $l' \in S^0(\Sigma)$. Therefore, $G \cup G^{-1} \models b^{-1}qb' : u \xrightarrow{\delta^{-1}l'\delta} v$. It is easy to see that $\delta^{-1}l'\delta \in S^0(\Sigma)$. □

Corollary 12 and Lemma 13 together provide an alternative characterization of membership in the unification closure of a labeled directed graph in terms of connectivity via labeled-paths. Objects witnessing membership in the unification closure are paths labeled with semi-Dyck sets. Obviously, this opens up the possibility of applying graph algorithms for labeled-path connectivity problems to problems related to membership in unification closure. For example, finding all witnesses, or finding the smallest-sized witness (see Section 4.3) can now be reduced to a semi-Dyck language path reachability problem.

Lemma 13 is used to prove the completeness of unification paths over a graph with respect to connectivity in the unification quotient of that graph.

Theorem 14 (Unification path completeness)**.** *Let $G = \langle \Sigma, V, D\rangle$ be a labeled directed graph with unification closure $\sim$. If $G/\sim \models [u] \xrightarrow{l'} [v]$, where $l' \in \Sigma^*$, then $G \cup G^{-1} \models u \xrightarrow{l} v$ for some $l \in S^*(\Sigma)$ such that $\mu_{\mathbf{S}(\Sigma)}(l) = l'$.*

Proof. Let $G/\sim \models p : [u] \xrightarrow{l'} [v]$. The proof is by induction on the construction of p.

(1) REF: If p is ϵ, the empty path, then $l' = \epsilon$, $[u] = [v]$, and $G \models u \sim v$. From Lemma 13, $G \cup G^{-1} \models q : u \xrightarrow{l} v$ for some $l \in S^0(\Sigma)$. The normal form of l is ϵ, which is equal to l'.

(2) INIT: Let $[u] \xrightarrow{\delta} [v]$ be an edge in $G/\sim$. Then $l' = \delta$ and $\delta \in \Sigma$. By the definition of a quotient graph,
(a) there is an edge $b : u' \xrightarrow{\delta} v' \in D$ for some $u' \in [u]$ and $v' \in [v]$,
(b) $G \models u \sim u'$, and
(c) $G \models v' \sim v$
By Lemma 13, $G \cup G^{-1} \models q : u \xrightarrow{l_1} u'$, where $l_1 \in S^0(\Sigma)$, and $G \cup G^{-1} \models r : v' \xrightarrow{l_2} v$, where $l_2 \in S^0(\Sigma)$. Using the TRANS rule for path construction, $G \cup G^{-1} \models qbr : u \xrightarrow{l_1 \delta l_2} v$. Clearly, $\mu_{\mathbf{S}(\Sigma)}(l_1 \delta l_2) = \delta$, which is equal to l'.

(3) TRANS: Let $G/\sim \models [u] \xrightarrow{l'_1} [v']$, and $G/\sim \models [v'] \xrightarrow{l'_2} [v]$. By the induction hypothesis, $G \cup G^{-1} \models q : u \xrightarrow{l_1} v'$, where $\mu_{\mathbf{S}(\Sigma)}(l_1) = l'_1$. Similarly, $G \cup G^{-1} \models r : v' \xrightarrow{l_2} v$, where $\mu_{\mathbf{S}(\Sigma)}(l_2) = l'_2$. Then, by the TRANS rule for path construction over $G \cup G^{-1}$, $G \cup G^{-1} \models qr : u \xrightarrow{l_1 l_2} v$. l'_1, l'_2, and $l'_1 l'_2$ are already in normal form with respect to $\mathbf{S}(\Sigma)$ reductions. Furthermore, $l' = l'_1 l'_2$. This implies that $\mu_{\mathbf{S}(\Sigma)}(l_1 l_2)$ simplifies to l', which completes the proof. □

Theorems 11 and 14 together show how unification source-tracking information may be encoded as unification paths.

4.3. Computation of shortest unification paths

One measure of the succinctness of source-tracking information is its length. Since unification paths are paths over a graph whose labels are constrained by the context-free grammar $S^*(\Sigma)$ for an alphabet Σ, computation of the shortest unification path is a special case of the context-free labeled-path problem. If G is a directed graph whose edges are labeled from an alphabet Σ, and $\mathbf{L}$ is a context-free language over Σ, the context-free labeled shortest-path problem consists of finding the shortest-path from the set of all paths p in G between a given source and destination vertex such that the label of p is a word in $\mathbf{L}$.

Shortest unification paths may therefore be computed using the dynamic-programming-based context-free labeled shortest-path algorithms proposed by Barrett et al. [3].[3] (See also [40].) If $\mathbf{L}$ is specified by a context-free grammar in Chomsky Normal Form, then the algorithm of Barrett et al. has time complexity $O(|V|^5|N|^2|R|)$, where V is the set of vertices in G, N the set of non-terminals, and R the set of productions of the grammar. The efficiency may be improved to $O(|V|^3|N||R|)$ using Fibonacci heaps.

For unification paths over a directed labeled graph $G = \langle \Sigma, V, D \rangle$, the grammar $S^*(\Sigma)$ is of size $O(\Sigma)$ and may be transformed into a grammar in Chomsky Normal Form whose set of non-terminals and productions are each of size $O(\Sigma)$. Thus, shortest unification paths can be computed in $O(|V|^3|\Sigma|^2)$ time. Assuming a fixed alphabet, this means that shortest unification paths can be computed in $O(|V|^3)$ time, where V is the vertex set of the unification graph.

This worst-case complexity can make a direct implementation of computing the optimal unification path expensive in practice. In the next few sections of the paper, we present a simple

[3] The algorithm is unaffected if labels on edges are drawn from $\Sigma \cup \{\epsilon\}$, where ϵ is the empty word [36].

extension to the unification algorithm that efficiently computes a non-optimal path which, in practice, may be adequate for the purpose of reasoning about membership in the unification closure.

5. The logic of unification path expressions

The goal of a logic is to formalize the principles of reasoning about objects of a model. The rules of the logic are used to explain and derive the consequences of the basic assumptions in the model. In the case of unification, we want to be able to formally reason about unifiability and non-unifiability of a system of equations. Since these properties of a system of equations are subsumed by the connectivity relation in the unification quotient of the unification graph representing the system of equations, it is natural to consider a logic of vertices and paths on the labeled directed graph unification graph.

We construct a simple deduction system P^U to compute proofs of connectivity relations in the unification quotient of a unification graph. These deductions are encoded using *unification path expressions*, generated from the edges of the graph using concatenation, the empty path expression ϵ, and an inverse operation.

The construction of unification path expressions over a labeled directed graph $G = \langle \Sigma, V, D \rangle$ is defined inductively using the system P^U of rules shown in Fig. 6. The rules may be thought of as a type system defining the well-typed unification path expressions from the set of untyped terms drawn from the term algebra $\mathbf{T}\ (\Sigma_{\mathbf{Gr}}, D)$ generated by D. Here, $\Sigma_{\mathbf{Gr}} = \{\epsilon \mapsto 0, (.)^{-1} \mapsto 1, \circ \mapsto 2\}$ is the group signature. Judgements are of the form $G \vdash p : u \xrightarrow{l} v$, where $p \in T^*(\Sigma_{\mathbf{Gr}}, D)$, the set of $\Sigma_{\mathbf{Gr}}$-terms over D, and $l \in \Sigma^*$.

We let $G \vdash_{P^U} p : u \xrightarrow{l} v$ denote judgements derived from the rules of Fig. 6. The triple $u \xrightarrow{l} v$ is the "type" of the expression p and consists of *endpoint vertices* u, v and a *signature* l. We say that *p is a unification path expression from u to v with signature l*. The rules of P^U mirror the definition of an equivalence relation and are easily interpretable as connectivity relations. Downward closure, however, is also captured as a connectivity relation by P^U via the DN rule. Given a path p connecting u' to v', the DN rule connects the child u of u' with child v of v' provided the branch edges b from u' to u and b' from v' to v have the same label δ. This connection involves traversing edge b^{-1} from u to u' labeled δ^{-1}, followed by the path expression p from u' to v' with signature ϵ, followed by the edge b' from v' to v labeled δ. The signature of the resultant path expression $b^{-1}pb'$ is $\mu_{\mathbf{S}(\Sigma)}(\delta^{-1}\epsilon\delta)$, which is ϵ.

Since the free monoid $(D \cup D^{-1})^*$ extended with the inverse function *inv* is a $\Sigma_{\mathbf{Gr}}$-algebra, the function *flatten* from D to $(D \cup D^{-1})^*$ defined by

$$\mathit{flatten}(c) = c$$

uniquely extends to the homomorphism $\mathit{flatten} : T^*(\Sigma_{\mathbf{Gr}}, D) \longrightarrow (D \cup D^{-1})^*$. We abbreviate $\mathit{flatten}(p)$ by $\overline{p}$. Thus, $\overline{\epsilon} = \epsilon$, $\overline{p^{-1}} = \mathit{inv}(\overline{p})$, and $\overline{pq} = \overline{p}\,\overline{q}$.

$$\text{INIT}\quad \frac{}{G \vdash c : u \xrightarrow{\eta} v} \qquad c : u \xrightarrow{\eta} v \in G$$

$$\text{REF}\quad \frac{}{G \vdash \epsilon : u \xrightarrow{\epsilon} u} \qquad u \in G$$

$$\text{SYM}\quad \frac{G \vdash p : v \xrightarrow{\epsilon} u}{G \vdash p^{-1} : u \xrightarrow{\epsilon} v}$$

$$\text{TRANS}\quad \frac{G \vdash p : u \xrightarrow{l} v' \qquad G \vdash q : v' \xrightarrow{l'} v}{G \vdash pq : u \xrightarrow{ll'} v}$$

$$\text{DN}\quad \frac{G \vdash p : u' \xrightarrow{\epsilon} v'}{G \vdash b^{-1}pb' : u \xrightarrow{\epsilon} v} \qquad \begin{array}{l}\delta \in \Sigma \\ b : u' \xrightarrow{\delta} u \in D \\ b' : v' \xrightarrow{\delta} v \in D\end{array}$$

Fig. 6. The logic $P^U(G)$ of unification path expressions over a labeled directed graph $G = \langle \Sigma, V, D \rangle$.

The main point of introducing unification path expressions is to show that for each deduction $G \vdash_{P^U} p : u \xrightarrow{l} v, \overline{p}$ is a unification path witnessing the membership of u, v in the unification closure of G. This is formalized in the next lemma:

Lemma 15 (Soundness and completeness of P^U)**.** *Let $G = \langle \Sigma, V, D \rangle$ be a labeled directed graph.*

(1) *(Soundness) If $G \vdash_{P^U} p : u \xrightarrow{l'} v$, then $\overline{p}$ is a unification path from u to v over G such that $\mu_{\mathbf{S}(\Sigma)}(l(\overline{p})) = l'$.*
(2) *(Completeness) If p is a unification path from u to v over G and $\mu_{\mathbf{S}(\Sigma)}(l(p)) = l'$, then $G \vdash_{P^U} p : u \xrightarrow{l'} v$.*

Proof. Soundness: By induction on P^U deductions.

- INIT: Suppose $G \vdash_{P^U} c : u \xrightarrow{\eta} v$. Then, $c : u \xrightarrow{\eta} v \in G$. The result follows immediately since $\overline{c} = c$.
- REF: Obvious.
- SYM: Suppose p is equal to q^{-1} and $\dfrac{G \vdash_{P^U} q : v \xrightarrow{\epsilon} u}{G \vdash_{P^U} q^{-1} : u \xrightarrow{\epsilon} v}$.
  By the induction hypothesis, $\overline{q}$ is a unification path from v to u over G. By the definition of *flatten*, $\overline{q^{-1}} = inv(\overline{q})$. By Lemma 9, it follows that $inv(\overline{q})$ is a unification path from u to v over G whose label reduces to ϵ under $\mathbf{S}(\Sigma)$ reduction.
- TRANS: Suppose p is equal to qr and

$$\frac{G \vdash q : u \xrightarrow{l'_q} v' \qquad G \vdash r : v' \xrightarrow{l'_r} v}{G \vdash qr : u \xrightarrow{l'_q l'_r} v}.$$

Thus, $l' = l'_q l' r$. By the induction hypothesis, $\overline{q}$ is a unification path from u to v' such that $l(\overline{q})$, abbreviated l_q, reduces to l'_q. By the induction hypothesis again, $\overline{r}$ is a unification path from u to v' such that $l(\overline{r})$, abbreviated l_r, reduces to l'_r. Now, $\overline{qr} = \overline{q}\ \overline{r}$. Also, from the definition of labeled-paths, it follows that $\overline{q}\ \overline{r}$ is a path from u to v labeled $l_q l_r$. Furthermore, l_q and l_r are both in $S^*(\Sigma)$, therefore so is $l_q l_r$. Next, we need to show that $\mu_{\mathbf{S}(\Sigma)}(l_q l_r)$ is equal to $l'_q l'_r$. We first observe that $\mu_{\mathbf{S}(\Sigma)}(l_q l_r)$ is equal to $\mu_{\mathbf{S}(\Sigma)}(\mu_{\mathbf{S}(\Sigma)}(l_q)\mu_{\mathbf{S}(\Sigma)}(l_r))$, which is $\mu_{\mathbf{S}(\Sigma)}(l'_q l'_r)$. Since both l'_q and l'_r are in Σ^*, there are no $\mathbf{S}(\Sigma)$ redexes in $l'_r l'_r$ and hence, $\mu_{\mathbf{S}(\Sigma)}(l'_q l'_r) = l'_q l'_r$.

- DN: Suppose p equals $b^{-1}qb'$, u', and v' are vertices in G, and $b : u' \xrightarrow{\delta} u, b' : v' \xrightarrow{\delta} v$ are branch edges in G, and $\dfrac{G \vdash q : u' \xrightarrow{\epsilon} v'}{G \vdash b^{-1}qb' : u \xrightarrow{\epsilon} v}$.
  By the induction hypothesis, $\overline{q}$ is a unification path over G from u' to v', such that $l(\overline{q})$, abbreviated l_q reduces to ϵ. Now, $\overline{p} = b^{-1}\overline{q}b'$ is clearly a path in $G \cup G^{-1}$ from u to v. To verify that $\overline{p}$ is a unification path over G from u to v, we note that $l(\overline{p})$, equal to $\delta l_q \delta^{-1}$, reduces to ϵ under $\mathbf{S}(\Sigma)$ rewriting.

Completeness: By a double induction on the derivation of unification path labels and paths in $G \cup G^{-1}$. We are given that p is a unification path from u to v with label $l(p)$. Since p is a unification path, $l_p \in S^*(\Sigma)$, where l_p abbreviates $l(p)$. Let $\mu_{\mathbf{S}(\Sigma)}(l_p) = l'_p$.

- $l_p = \epsilon$: Then, $l'_p = \epsilon$. there are four cases:
  - REF: $p = \epsilon$. Then, $u = v$ and $G \vdash_{P^U} \epsilon : u \xrightarrow{\epsilon} u$ by the REF rule.
  - INIT: $p = a$, where a is an equational edge from u to v. Then $G \vdash_{P^U} a : u \xrightarrow{\epsilon} v$ by the INIT rule.
  - SYM: $p = a^{-1}$, where a is an equational edge from v to u. Then, $G \vdash_{P^U} p : u \xrightarrow{\epsilon} v$ follows from one application of the INIT rule and one application of the SYM rule:

$$\frac{\dfrac{}{G \vdash a : v \xrightarrow{\epsilon} v}\ \text{INIT}}{G \vdash a^{-1} : u \xrightarrow{\epsilon} v}\ \text{SYM}.$$

  - TRANS: $p = qr$, where $G \cup G^{-1} \models q : u \longrightarrow v'$ and $G \cup G^{-1} \models r : v' \longrightarrow v$ are paths in $G \cup G^{-1}$. Since $l_p = \epsilon$, it follows that $l(q)$, abbreviated l_q and $l(r)$, abbreviated l_r are both ϵ. Therefore, both q and r are unification paths. By the inductive hypothesis, $G \vdash_{P^U} q : u \xrightarrow{\epsilon} v'$ and $G \vdash_{P^U} q : v \xrightarrow{\epsilon} v$. Using the TRANS rule,

$$\frac{\dfrac{\ldots}{G \vdash q : u \xrightarrow{\epsilon} v'} \quad \dfrac{\ldots}{G \vdash r : v' \xrightarrow{\epsilon} v}}{G \vdash qr : u \xrightarrow{\epsilon} v}\ \text{TRANS}$$

- $l_p = l_q\delta^{-1}l_r\delta l_s$, where $l_q, l_r, l_s \in S^0(\Sigma)$ and $\delta \in \Sigma$. By Lemma 1 and Lemma 2, there are vertices u', u'', v'', v' in G and paths q, r, s and branch edges b and b' in G such that $p = qb^{-1}rb's$, $G \cup G^{-1} \models q : u \xrightarrow{l_q} u'$, $b : u'' \xrightarrow{\delta} u' \in D$, $G \cup G^{-1} \models r : u'' \xrightarrow{l_r} v''$, $b' : v'' \xrightarrow{\delta} v' \in D$, $G \cup G^{-1} \models s : v' \xrightarrow{l_s} v$. By the induction hypothesis, $G \vdash_{P^U} q : u \xrightarrow{\epsilon} u'$, $G \vdash_{P^U} r : u'' \xrightarrow{\epsilon} v''$, and $G \vdash_{P^U} s : v' \xrightarrow{\epsilon} v$. Thus, we have the P^U deduction

$$\dfrac{\dfrac{\dfrac{\dots}{G \vdash_{P^U} q : u \xrightarrow{\epsilon} u'} \quad \dfrac{\dfrac{\dots}{G \vdash r : u'' \xrightarrow{\epsilon} v''}}{G \vdash b^{-1}rb : u' \xrightarrow{\epsilon} v'}\ \text{DN}}{G \vdash qb^{-1}rb : u \xrightarrow{\epsilon} v'}\ \text{TRANS} \quad \dfrac{\dots}{G \vdash s : v' \xrightarrow{\epsilon} v}}{G \vdash qb^{-1}rbs : u \xrightarrow{\epsilon} v}\ \text{TRANS}$$

- $l_p = l_q\delta l_r$ where $l_q, l_r \in S^*(\Sigma)$ and $\delta \in \Sigma$. Again, by Lemma 1 and Lemma 2, there are vertices u', v' in V and paths q, r and edge b in D such that $p = qbr$, $G \cup G^{-1} \models q : u \xrightarrow{l_q} u'$, $b : u' \xrightarrow{\delta} v' \in D$, $G \cup G^{-1} \models r : v' \xrightarrow{l_r} v$. Let l'_q and l'_r are equal to $\mu_{\mathbf{S}(\Sigma)}(l_q)$ and $\mu_{\mathbf{S}(\Sigma)}(l_r)$ respectively. By the induction hypothesis, $G \vdash_{P^U} q : u \xrightarrow{l'_q} u'$ and $G \vdash_{P^U} r : v' \xrightarrow{l'_r} v$. Now, $\mu_{\mathbf{S}(\Sigma)}(l_q\delta l_r)$ is equal to $\mu_{\mathbf{S}(\Sigma)}(l'_q\delta l'_r)$. This in turn is equal to $l'_q\delta l'_r$, since l'_r, δ, and l'_q are in Σ^*. Thus, we have the P^U deduction

$$\dfrac{\dfrac{G \vdash q : u \xrightarrow{l'_q} u' \quad \dfrac{}{G \vdash b : u' \xrightarrow{\delta} v'}\ \text{INIT}}{G \vdash qb : u \xrightarrow{l'_q\delta} v'}\ \text{TRANS} \quad \dfrac{\dots}{G \vdash r : v' \xrightarrow{l'_r} v}}{G \vdash qbr : u \xrightarrow{l'_q\delta l'_r} v}\ \text{TRANS}$$

This completes the proof. □

The above result, and those of Theorem 11 and Theorem 14 imply the soundness and completeness of P^U deductions with respect to connectivity in the $\sim$-quotient graph:

Theorem 16 (Soundness and completeness of P^U deductions)**.** *Let $G = \langle \Sigma, V, D\rangle$ be a labeled directed graph whose unification closure is $\sim$.*

(1) *(Soundness) If $G \vdash_{P^U} p : u \xrightarrow{l} v$, then $G/{\sim} \models u \xrightarrow{l} v$.*
(2) *(Completeness) If $G/{\sim} \models u \xrightarrow{l} v$, then $G \vdash_{P^U} p : u \xrightarrow{l} v$ where p is a $\Sigma_{\mathbf{Gr}}$-term over D.*

The results of this section show how P^U is adequate as a logic of unification. It seems reasonable to ask, then, how structure-sharing unification algorithms implement this logic. The answer to this question leads us to the design of an algorithm that integrates the proof construction in P^U with unification. We call this a *unification algorithm with source-tracking* and turn our attention to it in the next section.

6. Unification algorithm with source-tracking

The goal of this section is to show how to adapt standard structure-sharing unification algorithms to construct proofs of unification in the logic P^U. Since P^U relates vertices of the unification graph, it is more natural to integrate the derivation of P^U deductions with graph-based unification algorithms than transformation based algorithms like that of Martelli and Montanari [37].

6.1. The Robinson unification algorithm

Fig. 7 illustrates a "text-book" variety unification algorithm. This is the recursive version of Robinson's linear space, exponential time algorithm operating on term graphs with structure sharing [47]. (See, for example, its presentation in [2] and also [1], p. 85). There are of course, many other unification algorithms whose theoretical worst case behavior is much better than Robinson's, like the quadratic one due to Corbin and Bidoit [16] obtained by a simple modification to Robinson's, the almost linear time algorithms of Huet [28] and Ružička and Prívara [49] and Baxter [4], and the linear time algorithms due to Paterson and Wegman [43] and Martelli and Montanari [37], to mention a few. Despite its exponential worst-case behavior, the Robinson algorithm is quite efficient in practice and is commonly used in many practical implementations of unification. We have therefore chosen to use it to illustrate the integration of source-tracking information with unification.

The algorithm operates on a term graph structure that is acyclic. Each vertex v of the unification graph has the following fields: a *type* field which identifies whether v is strict or is a function vertex. A *binding* field which is either empty (in which case v is *unbound*) or a pointer to a variable v' related to v by the unification closure. The procedure *unify* takes a pair of vertices and either terminates successfully indicating that the terms represented by the vertices have unified, or fails, indicating either a clash or a cycle. Equivalence classes of vertices are maintained as a tree via the *binding* pointer. The procedure *find* takes a vertex v and returns the root of the tree representing the equivalence class $[v]$. The procedure *union* takes a pair of roots of equivalence classes and merges them by making the binding pointer of one of the roots point to the other root. The procedure *occurs?* (v_1, v_2) determines whether the term represented by vertex $[v_2]$ is reachable from $[v_1]$ in the quotient graph. More precisely, *occurs?*(v_1, v_2) returns either no, or yes(x), where $x \in \{0, +\}$. If it returns no, then $[v_1]$ and $[v_2]$ are unconnected in the quotient graph. If *occurs?*(v_1, v_2) returns yes(0), then $[v_1] = [v_2]$, that is $v_1 \sim v_2$. Otherwise, $[v_1]$ and $[v_2]$ are distinct and $[v_2]$ is reachable from $[v_1]$ in the quotient graph.

The proofs of termination, correctness, and the exponential time complexity (and linear space-complexity) of the Robinson unification algorithm are standard and can be found in several places. (See [1] and also [2], for example.)

6.2. Source-tracking the Robinson algorithm

We now consider how the computation of source-tracking information can be built into the Robinson algorithm. Only a simple addition of proof information is needed to accommodate the construction of P^U deductions by the unification algorithm of Fig. 7. The construction requires adding an additional "proof" parameter—which turns out to be a unification path expression—to the algorithm. The extra parameter may be informally interpreted as containing the execution "trace" of P^U inferences made by the unification algorithm.

```
01 procedure unify(v1, v2) =
02     let r1 = find(v1) and r2 = find(v2)
03         in if r1 = r2 then return
04             else case r1.type, r2.type
05                 strict, strict: union(r1, r2)
06                 function, strict: unify(v2, v1)
07                 strict, function: let ans = occurs?(r2, r1)
08                                       in case ans
09                                           no: union(r1, r2)
10                                           yes: fail(CYCLE)
11                 function,function:
12                     if L(r1) ≠ L(r2)
13                     then fail(CLASH)
14                     else
15                         for i = 1 to α(L(r1)) do
16                             unify(r1.child(i), r2.child(i))
17                                 where b1 = edge(r1, r1.child(i))
18                                     and  b2 = edge(r2, r2.child(i))

19 procedure union(r1, r2) = r1.binding := r2

20 procedure find(v) =
21     if unbound?(v) then return v
22     else let v' = v.binding
23             in let r = find(v') in return r

24 procedure occurs?(v1, v2) =
25     let r1 = find(v1) and r2 = find(v2)
26         in if r1 = r2 then return yes
27           else case r1.type
28               strict: return no
29               function:
30                   for i = 1 to α(L(r1)) do
31                       let ans = occurs?(r1.child(i), r2)
32                           in case ans
33                               no: continue
34                               yes: return yes(+, p1bqp2^-1)
35                                       where b = edge(r1, r1.child(i))
36                   return no
```

Fig. 7. Computing the most general unifier, based on the recursive, structure-sharing term graph version of the Robinson unification algorithm [47]. This presentation has been adapted from [2] and also from [1], p. 85.

Fig. 8 shows the unification algorithm of Fig. 7 with source-tracking information attached is shown in . In the extended unification algorithm, the *binding field* contains an additional attribute, which is a unification path expression denoting the path expression from v to v'. The procedure *find* takes a variable v, follows a chain of *binding* pointers starting from v and returns, in addition to the root of the equivalence class under unification closure containing v, a path expression from v to

```
01 procedure unify(v1, v2, m) =
02     let ⟨r1, p1⟩ = find(v1) and ⟨r2, p2⟩ = find(v2)
03         in if r1 = r2 then return
04             else case r1.type, r2.type
05                     strict, strict: union(r1, r2, p1^-1 m p2)
06                     function, strict: unify(v2, v1, m^-1)
07                     strict, function: let ans = occurs?(r2, r1)
08                                           in case ans
09                                                 no: union(r1, r2, p1^-1 m p2)
10                                                 yes(_ , q): fail(CYCLE, r1, p1^-1 m p2 q)
11                     function,function:
12                         if L(r1) ≠ L(r2)
13                         then fail(CLASH, r1, r2, p1^-1 m p2)
14                         else
15                               for i = 1 to α(L(r1)) do
16                                   unify(r1.child(i), r2.child(i), b1^-1 p1^-1 m p2 b2)
17                                       where b1 = edge(r1, r1.child(i))
18                                           and  b2 = edge(r2, r2.child(i))

19 procedure union(r1, r2, p) = r1.binding := ⟨r2, p⟩

20 procedure find(v) =
21     if unbound?(v) then return ⟨v, ϵ⟩
22     else let ⟨v', p⟩ = v.binding
23             in let ⟨r, q⟩ = find(v') in return ⟨r, pq⟩

24 procedure occurs?(v1, v2) =
25     let ⟨r1, p1⟩ = find(v1) and ⟨r2, p2⟩ = find(v2)
26         in if r1 = r2 then return yes(0, p1 p2^-1)
27           else case r1.type
28               strict: return no
29               function:
30                   for i = 1 to α(L(r1)) do
31                       let ans = occurs?(r1.child(i), r2)
32                           in case ans
33                               no: continue
34                               yes(_, q): return yes(+, p1 b q p2^-1)
35                                       where b = edge(r1, r1.child(i))
36                   return no
```

Fig. 8. The unification algorithm of Fig. 7 extended with source-tracking.

the root. The procedures *unify* and *union* also carry an extra parameter which is a unification path expression. The *union* procedure sets the *binding* field. The procedure *occurs?*(u, v) returns either no or yes(x, p), where x is either 0 or $+$, and p is a unification path expression from u to v.

Some of the unification paths constructed by the algorithm are schematically illustrated in Figs. 9–11.

If $m : \tau_1 \stackrel{?}{=} \tau_2$ is a term equation, its unification graph G contains term trees rooted at vertices v_1 and v_2, representing the terms τ_1 and τ_2 respectively, and an equational edge from v_1 to v_2 labeled m. The first call to unify is $unify(v_1, v_2, m)$. The unification algorithm maintains the invariant that for every call $unify(v_1, v_2, p)$, $G \vdash_{PU} p : u \xrightarrow{\epsilon} v$. Also, if $find(v_1) = \langle r_1, p_1 \rangle$, then $G \vdash_{PU} p_1 : v_1 \xrightarrow{\epsilon} r_1$. Similarly, if $find(v_2) = \langle r_2, p_2 \rangle$, then $G \vdash_{PU} p_2 : v_2 \xrightarrow{\epsilon} r_2$. The call $union(r_1, r_2, p)$ maintains the invariant that $G \vdash_{PU} p : r_1 \xrightarrow{\epsilon} r_2$. If the call $unify(v_1, v_2, m)$ results in the call $union(r_1, r_2, p_1^{-1} m p_2)$, binding r_1 to r_2, where $find(v_1) = \langle r_1, p_1 \rangle$ and $find(v_2) = \langle r_2, p_2 \rangle$, then the vertex r_2 and the path expression $p_1^{-1} m p_2$ are stored as the binding information $r_1.binding$ (see Fig. 9A). The path $p_1^{-1} m p_2$ implicitly carries the information that v_1 is connected to v_2 with the path m. The algebra of path

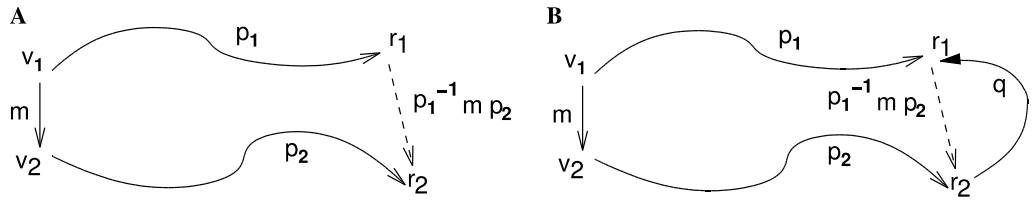

Fig. 9. Schematic view of path expression construction in procedure *unify* of Fig. 8. (A) The unification path expression constructed at lines 05, 09, and 13. (B) The path constructed at line 10.

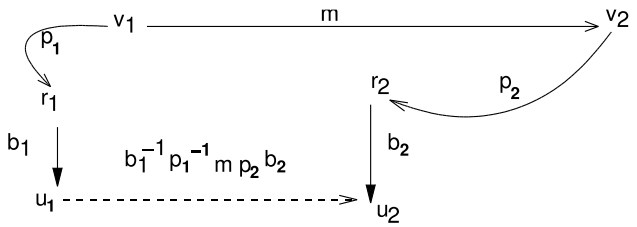

Fig. 10. Schematic view of path expression construction in procedure *unify* of Fig. 8 at line 16. u_1 and u_2 abbreviate $r_1.child(i)$ and $r_2.child(i)$, respectively, in line 16 of *unify*.

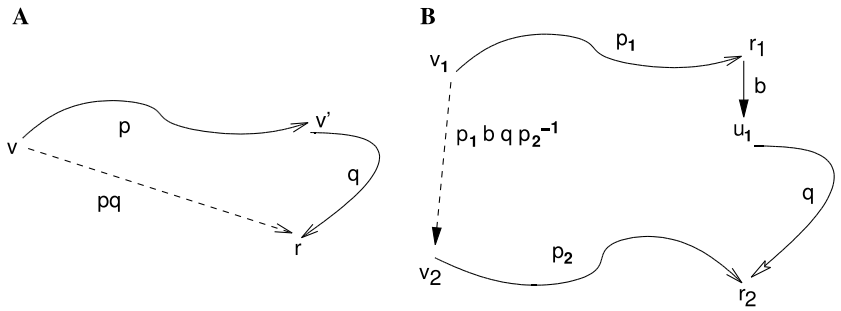

Fig. 11. Schematic view of path expression construction in procedures *find* and *occurs* of the unification source-tracking algorithm of Fig. 8. (A) The unification path expression constructed at line 23 of procedure *find*. (B) The unification path expression constructed at line 34 of procedure *occurs?*

expressions allows us to recover the information that v_1 is connected to v_2 by m. We simply compute the path expression following the *binding* pointers from v_1 to v_2, which is done in three steps:

(1) compute the path expression from v_1 to r_1 using *find*(v_1), which is p_1.
(2) compute the path from r_1 to r_2 using *find*(r_1), which is $p_1^{-1}mp_2$.
(3) compute the path from r_2 to v_2, which is p_2^{-1}.

The concatenation of these three path expressions, $p_1p_1^{-1}mp_2p_2^{-1}$, simplifies to m using the rewrite rules discussed later in Section 7. All paths expressions manipulated by the algorithm are unification path expressions.

Theorem 17 (Invariants of the unification source-tracking algorithm)**.** *Let G be a Σ-unification graph of a term equation $\tau_1 \stackrel{?}{=} \tau_2$ between the Σ-terms τ_1 and τ_2 represented by the term graphs rooted at vertices v_1 and v_2 and connected by an equational edge m. Let the top-level invocation of the unification algorithm of Fig. 8 be unify(v_1, v_2, m). Then, the following invariants are maintained for each subsequent call to unify, union, find, occurs?, and fail.*

(1) *For each call unify(u, v, p), $G \vdash_{P^U} p : u \xrightarrow{\epsilon} v$.*
(2) *For each call union(u, v, p), $G \vdash_{P^U} p : u \xrightarrow{\epsilon} v$.*
(3) *For each call fail$(CYCLE, u, s)$, $G \vdash_{P^U} s : u \xrightarrow{l} u$, where $l \in \Sigma_{\mathbf{N}}^+$.*
(4) *For each call fail$(CLASH, u, v, s)$, u and v are function vertices such that $L(u) \neq L(v)$, and $G \vdash_{P^U} s : u \xrightarrow{\epsilon} v$.*
(5) *If find$(u) = \langle v, p\rangle$, then $G \vdash_{P^U} p : u \xrightarrow{\epsilon} v$.*
(6) *If occurs?$(u, v) = \mathsf{yes}(x, p)$, then $x \in \{0, +\}$, $G \vdash_{P^U} p : u \xrightarrow{l} v$ and $l \in \Sigma_{\mathbf{N}}^x$.*

Proof. Without loss of generality, we assume that the system of term equations represented by the unification graph consists of a single equation. Thus, there is a single "top-level" call *unify*(v_1, v_2, m), where m represents an equational edge. Clearly, the above algorithm terminates since the unification algorithm terminates. For the calls to *unify*, we show that the invariant is true at each call within the body of the *unify* procedure, assuming the invariant is true at the entry to the procedure and assuming the invariants for *find* and *occurs?* are true.

For the base case corresponding to the top-level call *unify*(v_1, v_2, m), m is just equal to the equational edge labeled ϵ connecting vertices v_1 and v_2 in G. Therefore, using the INIT rule, we have $G \vdash_{P^U} p : v_1 \xrightarrow{\epsilon} v_2$.

For the inductive case, we construct P^U deductions for calls at each of the following locations:

(1) Line 2: By the inductive hypothesis, $G \vdash_{P^U} p_1 : v_1 \xrightarrow{\epsilon} r_1$ and $G \vdash_{P^U} p_2 : v_2 \xrightarrow{\epsilon} r_2$. Also, $G \vdash_{P^U} m : v_1 \xrightarrow{\epsilon} v_2$ using the INIT rule. Using one application of the SYM rule and two applications of the TRANS rules, we have $G \vdash_{P^U} p_1^{-1}mp_2 : r_1 \xrightarrow{\epsilon} r_2$. This proves the invariant for the calls to *union* on lines 5, 9, and the clash on line 13.
(2) Line 6: Using one application of the SYM rule, we get $G \vdash_{P^U} m^{-1} : v_2 \xrightarrow{\epsilon} v_1$.

(3) Line 10: By the induction hypothesis, and the fact that r_1 and r_2 are roots of different equivalence classes, it follows that $G \vdash_{PU} q : r_2 \xrightarrow{l} r_1$, where $l \in \Sigma_N^+$. Again, using the SYM and TRANS rules, we obtain the following deduction for a cycle involving r_1: $G \vdash_{PU} p_1^{-1} m p_2 q : r_1 \xrightarrow{l} r_1$, where $l \in \Sigma_{\mathbf{N}}^+$.

(4) Line 16: From the invariant at line 2 (see (1)), we have $G \vdash_{PU} p_1^{-1} m p_2 : r_1 \xrightarrow{\epsilon} r_2$. From this, and lines 17 and 18, using the DN rule, we can construct a deduction for $G \vdash_{PU} b_1^{-1} p_1^{-1} m p_2 b_2 : r_1.child(i) \xrightarrow{\epsilon} r_2.child(i)$.

From the invariant (2) just proved about *union*, and from line 19, we conclude that whenever $u.binding = \langle v, p \rangle$, it is the case that $G \vdash_{PU} p : u \xrightarrow{\epsilon} v$.

We prove the invariants for *find* and *occurs?* by induction on the depth of their calls. For the base case of invariant (5) of *find*, if v is unbound, then $find(v) = \langle v, \epsilon \rangle$. Clearly, using the REF rule, $G \vdash_{PU} \epsilon : v \xrightarrow{\epsilon} v$. Otherwise, from line 22, we have $G \vdash_{PU} p : v \xrightarrow{\epsilon} v'$ and from the inductive hypothesis and line 23, we have, $G \vdash_{PU} q : v' \xrightarrow{\epsilon} r$. Using the TRANS rule, we have $G \vdash_{PU} pq : v \xrightarrow{\epsilon} r$.

To show invariant (6) of *occurs?*, we examine the return values at lines 26 and 34. By the inductive hypothesis, the invariants for *find* hold. Therefore, from line 25, $G \vdash_{PU} p_1 : v_1 \xrightarrow{\epsilon} r_1$ and $G \vdash_{PU} p_2 : v_2 \xrightarrow{\epsilon} r_2$. On line 26, using one application of the SYM rule and one application of the TRANS rule (since $r_1 = r_2$), we get $G \vdash_{PU} p_1 p_2^{-1} : v_1 \xrightarrow{\epsilon} v_2$.

From the induction hypothesis, on line 34, $G \vdash_{PU} q : r_1.child(i) \xrightarrow{l} r_2$, where $l \in \Sigma_{\mathbf{N}}^*$. From line 35, using the INIT rule, we have $G \vdash_{PU} b : r_1 \xrightarrow{f.i} r_1.child(i)$, where f abbreviates $L(r_1)$. From line 25, we have $G \vdash_{PU} p_1 : v_1 \xrightarrow{\epsilon} r_1$ and $G \vdash_{PU} p_2 : v_2 \xrightarrow{\epsilon} r_2$. From these, using one application of the SYM rule and three applications of the TRANS rule, we get $G \vdash_{PU} p_1 b q p_2^{-1} : v_1 \xrightarrow{f.i\ l} v_2$, where Since $l \in \Sigma_{\mathbf{N}}^*$, it follows that $f.i\ l \in \Sigma_{\mathbf{N}}^+$. □

The invariants show that at each stage, the unification algorithm with source tracking not only constructs the unification closure $\sim$ of a unification graph G, but also computes witnesses of membership in the relation $\sim$. These witnesses are unification path expressions. When *unify* fails, the algorithm presents a witness of non-unifiability:

Corollary 18 (Witnesses to non-unifiability)**.** *Let G be a Σ-unification graph of term equation $\tau_1 \stackrel{?}{=} \tau_2$ between Σ-terms τ_1 and τ_2 represented by the term graphs rooted at vertices v_1 and v_2 and connected by an equational edge m. Let $unify(v_1, v_2, m)$ be the top-level call of the unification algorithm. Then*

(1) *If $unify = fail(CYCLE, u, q)$, then $G \vdash_{PU} q : u \xrightarrow{l} u$ and $l \in \Sigma_{\mathbf{N}}^+$.*

(2) *If $unify = fail(CLASH, w, w', q)$, then $G \vdash_{PU} q : w \xrightarrow{\epsilon} w'$ for function vertices w, w' in G such that $L(w) \neq L(w')$.*

The source-tracking algorithm of Fig. 8 can be seen as a constructive extension of the unification algorithm of Fig. 7. It returns a proof object of every inference it makes. The algorithm of Fig. 7 can be recovered from that of Fig. 8 by suppressing the path expression parameters and return values.

To see how the source-tracking algorithm of Fig. 8 works, consider the following example:

Example 19. Let E be the system of equations

$$\{a_1 : x \stackrel{?}{=} \mathsf{int}, a_2 : y \stackrel{?}{=} z, a_3 : y \stackrel{?}{=} \mathsf{int}\}.$$

Fig. 12 shows the unification graph of these three equations consisting of three variable vertices x, y, and z and one function vertex w labeled int. Assuming the three equations are chosen for unification in the order a_1, a_2, a_3, the resulting sequence of calls to *unify* using the source-tracking algorithm of Fig. 8 is

(1) $unify(x, w, a_1)$,
(2) $unify(y, z, a_2)$,
(3) $unify(y, w, a_3)$.

The result of these calls generates the graph shown in Fig. 12B. Note that in this graph, each edge denotes a binding pointer and not an edge of the original graph in Fig. 12A. The annotations on each binding pointer are computed using the source-tracking algorithm. The annotations are such that navigating between any two vertices will generate the unification path expression witnessing the connection between the two vertices. For example, to check how y is connected to the function vertex w labeled int, we invoke the query $occurs?(y, w)$, which returns $\mathsf{yes}(0, a_2 a_2^{-1} a_3)$, indicating $y \sim w$ and the unification path witnessing this is the unification path expression $a_2 a_2^{-1} a_3$.

This expression can be simplified (using rules discussed in the next section), to a_3, eliminating the irrelevant deductions involving equations a_2.

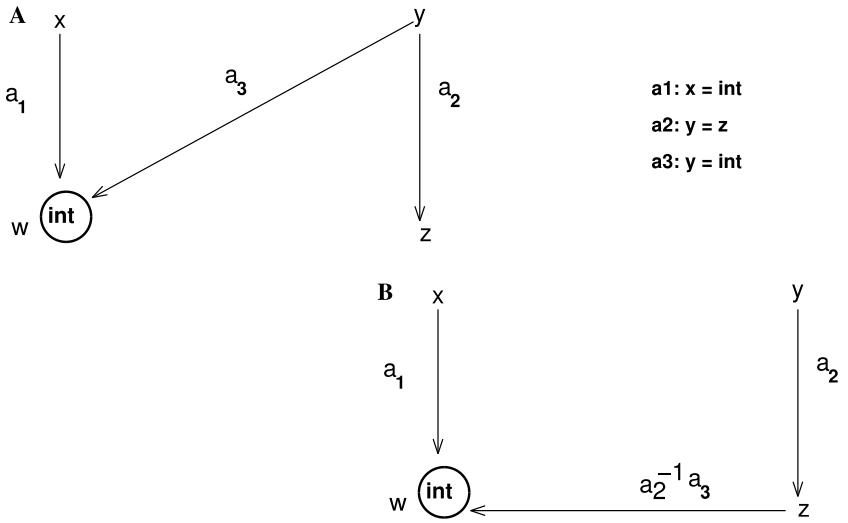

Fig. 12. The representation of equations of Example 19. (A) The graph of the original equations. (B) The binding pointers annotated with unification path expressions constructed by the unification source-tracking algorithm of Fig. 8. Running the ordinary unification algorithm of Fig. 7 will result in the graph with the same binding pointer structure as shown in (B), but without the annotations.

6.3. Accommodating optimizations

We discuss how to accommodate source-tracking with two common optimizations of the unification algorithm of Fig. 8. The first optimization, due to Corbin and Bidoit [16], includes a small but significant change to the unification algorithm of Fig. 7 which reduces the time complexity from exponential to quadratic. The modification involves inserting a call to $union(r_1, r_2)$ between lines 14 and 15 of the Robinson algorithm of Fig. 7. Accommodating this optimization with source-tracking involves inserting the call $union(r_1, r_2, p_1^{-1}mp_2)$ between lines 14 and 15 of the Robinson algorithm with source-tracking of Fig. 8.

Path compression is a common optimization employed for speeding up the *find* procedure [51]. Path compression can easily be added to the algorithm shown in Fig. 8. The *find* procedure of Fig. 8 needs to be replaced by the procedure *findcompress*. In this procedure, before returning the pair consisting of the root and the path to the root of a variable, the variable's binding is assigned that pair. The procedure *findcompress* is shown in Fig. 13.

Continuing Example 19, a $findcompress(y)$ query on the unification graph of Fig. 12B results in the graph shown in Fig. 14. Note the binding of variable y and its annotation $a_2a_2^{-1}a_3$, which when simplified using rules discussed in the next section, yields a_3. Both the graphs of Figs. 12B and 14 correctly encode the path information of the original graph of Fig. 12A.

We have illustrated how the construction of unification path expressions encoding P^U deductions can be easily incorporated into standard unification algorithms based on structure-sharing unification graphs. These path expressions, however, could encode redundant edge traversals some of which could cancel out, like the path expression $a_2a_2^{-1}a_3$ annotating the edge from y to w in Fig. 14. This expression can be simplified to a_3 by reducing $a_2a_2^{-1}$ to ϵ using cancellation rules. Simplification using a formal rewrite system is the subject of the next section.

```
01 procedure findcompress(v) =
02     if unbound?(v) then return ⟨v, ϵ⟩
03     else let ⟨v′, p⟩ = v.binding
04             in let ⟨r, q⟩ = findcompress(v′) in
05                 v.binding := ⟨r, pq⟩;
06                 return ⟨r, pq⟩
```

Fig. 13. The procedure *findcompress* for implementing source-tracking in the presence of path compression.

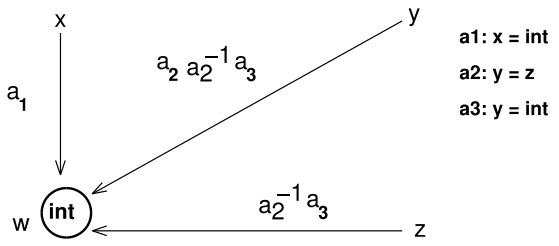

Fig. 14. The result of path compression after the query $findcompress(y)$ is applied on the unification graph of Fig. 12B. The figure also illustrates the sequence of calls to *unify* preceding the query *findcompress*.

7. Simplification of unification path expressions

Simplification of unification path expressions is achieved using the rewrite system R/A for free groups of Peterson and Stickel [44]. The rewrite system is shown in Fig. 15. (For a general introduction to term rewriting systems, see, for example, Baader and Nipkow's text [1].)

Theorem 20 ([44]). *The equational system R/A is convergent and complete with respect to the equational theory of free groups.*

The normal form of a term p under R/A rewriting is denoted $\mu_{\mathbf{Gr}}(p)$. It may be computed by first flattening p to $\overline{p}$ and then reducing $\overline{p}$ to its normal form by applying the two-sided cancellation rules:

Lemma 21 (Decomposition of normal forms). *If D is any set and $p \in T^*(\Sigma_{\mathbf{Gr}}, D)$, then $\mu_{\mathbf{Gr}}(p) = \mu_{\mathbf{S}_2\ (D)}(\overline{p})$.*

Proof. It is easy to see that $p \xrightarrow{*}_{R/A} \overline{p}$ and therefore p and $\overline{p}$ have the same normal form under R/A. Since $\overline{p}$ is flattened, every symbol in p is either c^{-1} or c, where $c \in D$. Hence, the only redexes in $\overline{p}$ are of the form $cc^{-1} \longrightarrow_{R/A} \epsilon$ and $c^{-1}c \longrightarrow_{R/A} \epsilon$. The redexes using $\mathbf{S}_2(D)$ are the same.

7.1. Weak subject reduction

Since unification path expressions are "typed," and the rewrite system operates on "untyped" terms, it is important to examine the effect of rewriting on the types of expressions. Unification path expressions are not closed under one-step R/A rewriting. Thus, P^U lacks the subject reduction property with respect to R/A rewriting. The types "lost" after one step of rewriting are, however, recovered at normalization. We refer to this phenomenon as *weak subject reduction*. We illustrate this idea through an example.

Associativity	$(pq)r$	$=$ $p(qr)$	$= pqr$
1	$(pq)^{-1}$	$\longrightarrow q^{-1}p^{-1}$	
2	$(p^{-1})^{-1}$	$\longrightarrow p$	
3	ϵ^{-1}	$\longrightarrow \epsilon$	
4	$p\epsilon$	$\longrightarrow p$	
5	ϵp	$\longrightarrow p$	
6	pp^{-1}	$\longrightarrow \epsilon$	
7	$p^{-1}p$	$\longrightarrow \epsilon$	

Fig. 15. Equational rewrite system R/A for free groups, where A consists of the Equational rule of Associativity and R consists of the remaining rules [44].

Example 22 (*Types are not "preserved" by one-step rewriting in R/A, but are recovered at normalization.*)**.**
Let G be a labeled directed graph consisting of the edges

$$\{a : w \xrightarrow{\epsilon} w', b_1 : w \xrightarrow{\text{f.1}} u, b_2 : w' \xrightarrow{\text{f.1}} v\}.$$

Let $p = (b_1^{-1}(ab_2))^{-1}$. Clearly, $G \vdash_{PU} p : v \xrightarrow{\epsilon} u$. If $q = (ab_2)^{-1}(b_1^{-1})^{-1}$, then $p \longrightarrow_{R/A} q$, but $G \not\vdash_{PU} q : v \xrightarrow{\epsilon} u$. However, $G \vdash_{PU} b_2^{-1}a^{-1}b_1 : v \xrightarrow{\epsilon} u$, where $b_2^{-1}a^{-1}b_1$ is the normal form of p and q under R/A rewriting.

Simplification using R/A rewriting is justified because unification path expressions normalize to unification paths of the same type. To show this, we rely on the following easily proved property about Dyck languages: reduction by two-sided cancellations on $S^*(\Sigma)$ words can be simulated by one-sided cancellation:

Lemma 23 (Closure of $S^*(\Sigma)$ words under $\mathbf{S_2}(\Sigma)$ reduction)**.** *If $x \in S^*(\Sigma)$ and $x \longrightarrow_{\mathbf{S_2}(\Sigma)} y$, then $x \approx_{\mathbf{S}(\Sigma)} y$ and $\mu_{\mathbf{S}(\Sigma)}(x) = \mu_{\mathbf{S}(\Sigma)}(y)$.*

Proof. By a case analysis on the rules of $\mathbf{S_2}$. Let $\delta \in \Sigma$.

(1) $\delta^{-1}\delta \longrightarrow \epsilon$: $l = r\delta^{-1}\delta s$ and $l' = rs$ for some $r, s \in (\Sigma \cup \Sigma^{-1})^*$. Since the redex $\delta^{-1}\delta$ is being reduced to ϵ, and $\delta^{-1}\delta \approx \epsilon \in \mathbf{S}(\Sigma)$, it follows that $l \approx_{\mathbf{S}(\Sigma)} l'$.

(2) $\delta\delta^{-1} \longrightarrow \epsilon$: $l = r\delta\delta^{-1}s$ and $l' = rs$, for some $\delta \in \Sigma$ and $r, s \in (\Sigma \cup \Sigma^{-1})^*$. By Lemma 7, $s = s_1\delta s_2$, where $s_1 \in S^0$. Again, since δ occurs in l, there are two cases:

(a) $r = r_1\delta^{-1}r_2$, where $r_2 \in S^0$: Therefore,

l	$=$	$r\delta\delta^{-1}s$	given
	$=$	$r_1\delta^{-1}r_2\delta\delta^{-1}s_1\delta s_2$	by Lemma 7
	$\xrightarrow{*}_{\mathbf{S}}$	$r_1\epsilon\epsilon s_2$	since $r_2, s_1 \in S^0$
	$\xrightarrow{*}_{\mathbf{S}}$	$r_1 s_2$	and
l'	$=$	rs	given
	$=$	$r_1\delta^{-1}r_2 s_1\delta s_2$	since $r = r_1\delta^{-1}r_2$, $s = s_1\delta s_2$
	$\xrightarrow{*}_{\mathbf{S}}$	$r_1\epsilon s_2$	since $r_2, s_1 \in S^0$
	$\xrightarrow{*}_{\mathbf{S}}$	$r_1 s_2$	

Hence, we have, $l \approx_{\mathbf{S}(\Sigma)} r_1 s_2 \approx_{\mathbf{S}(\Sigma)} l'$,

(b) $r \in S^*(\Sigma)$ and $\delta s \in S^*$. Therefore, $l = r\delta\delta^{-1}s_1\delta s_2$. Since $s_1 \in S^0$, $l \xrightarrow{*}_{\mathbf{S}} r\delta s_2$. Also, $l' = rs_1\delta s_2$. Since $s_1 \in S^0$, $l' \xrightarrow{*}_{\mathbf{S}} r\delta s_2$. Hence both l and l' reduce to $r\delta s_2$, implying $l \approx_{\mathbf{S}(\Sigma)} r\delta s_2 \approx_{\mathbf{S}} l'$. □

Theorem 24 (P^U weak subject reduction)**.** *Let G be a labeled directed graph and $G \vdash_{P^U} p : u \xrightarrow{l} v$. If $p' = \mu_{\mathbf{Gr}}(p)$, then $G \vdash_{P^U} p' : u \xrightarrow{l} v$.*

Proof. By Lemma 15, $G \vdash_{P^U} p : u \xrightarrow{l} v$ implies $G \vdash_{P^U} \overline{p} : u \xrightarrow{l} v$. By Lemma 21, $p' = \mu_{\mathbf{S}_2(D)}(\overline{p})$. Since $\overline{p} \xrightarrow{*}_{\mathbf{S}_2(D)} p'$, it follows that $l(\overline{p}) \xrightarrow{*}_{\mathbf{S}_2(\Sigma)} l(p')$. Since $l(\overline{p}) \in S^*(\Sigma)$, by Lemma 23, $\mu_{\mathbf{S}(\Sigma)}(l(p'))$ is equal to $\mu_{\mathbf{S}(\Sigma)}(l(\overline{p}))$, which is l. Then $G \vdash_{P^U} p' : u \xrightarrow{l} v$ follows from Lemma 15 (Completeness). □

7.2. Efficiency considerations

The cost of constructing unification path expressions at each point in the algorithm of Fig. 8 is constant time per call to *unify*, *find*, *union*, and *occurs?*, assuming paths are represented as terms sharing structure. Thus, the addition of source-tracking to the unification algorithm of Fig. 7 increases its runtime by only a constant factor.

Since normalization is orthogonal to the building of path expressions, it may be performed once the unification path expression has been computed. It is easily seen that normalization using R/A takes time proportional to the size of the term being normalized. Although the normal form does not always correspond to the shortest unification path, its computation is considerably less expensive than that of the shortest unification path.

A unification path p of type $u \xrightarrow{l} v$ in a labeled directed graph G is *minimal* if there is no unification path q of type $u \xrightarrow{l} v$ in G such that the edge set of q is a proper subset of the edge set of p. The normal forms obtained by R/A rewriting do not yield minimal unification paths. Consider the example set of equations $\{a : x \stackrel{?}{=} y, a' : y \stackrel{?}{=} x\}$. The path $a'a^{-1} : y \xrightarrow{\epsilon} y$ does not reduce to the minimal path $\epsilon : y \xrightarrow{\epsilon} y$ under R/A rewriting without the presence of a "type aware" rule that rewrites p to ϵ if $p : u \xrightarrow{\epsilon} u$.

8. Related research

We briefly survey the various approaches to the error diagnosis in unification and unification-based systems, including type inference and logic programming.

8.1. Logic programming and unification-based systems

Port [45] carried out unification failure analysis by identifying minimally unifiable subsets. Port's algorithm constructs regular path expressions over the unification graph. Our work shows that the path expressions of relevance are context-free, not regular.

Cox [17] and Chen et al. [10] propose an algorithm to derive maximally unifiable subsets and minimally non-unifiable subsets of term equations employed as the basis for developing search strategies for breadth-first resolution of logic programs. Our extension to the unification algorithm keeps track of information that is more precise than subsets. Furthermore,

the information maintained statically in the auxiliary graph constructed by the algorithm of Chen et al. can be generated dynamically in our extension to the unification algorithm. On the other hand, the simplification framework proposed in this paper does not address minimality.

Two approaches closely related to ours are the unification failure analysis of Cox [18] and the Logics of Unification by Le Chenadec [34]. Cox treats the unification graph as a push-down automaton in which the vertices and edges of the unification graph form the states and transitions of the automaton respectively. However, because the formalization is in terms of a transition relation on configuration states, the connection with labeled-paths is not made in the paper. On the other hand, our framework is based on labeled semi-Dyck paths and emphasizes unification as a path connectivity problem. This results in a clearer and more general formalism. For example, we are able to precisely formulate the relation between arbitrary paths in the quotient graph (not just cycles, as in Cox's paper [18]) with the path structure in the original graph.

Le Chenadec [34] introduces a framework of logical systems to characterize unification and congruence closure. P^U is equivalent to Le Chenadec's logic LE_0: both LE_0 and P^U are sound and complete with respect to paths in the quotient unification graph. The syntactic machinery of Le Chenadec's logics, however, involves terms and contexts, whereas P^U is a logic of labeled-paths on graph vertices that is explicitly aware of the sharing relationship between subterms. The connection with context-free languages makes it possible for us to extract simple and practical algorithms for proof construction, something that is harder to do from the work of Cox and Le Chenadec.

8.2. Diagnosis of type inference

To provide explanations of type inference, Wand [55] and later Beaven and Stansifer [5] and Duggan and Bent [21] modified the unification algorithm to accumulate sets of reasons (called "reason lists" in [55]) when traversing the unification graph. Each reason in the reason list is a fragment of the original source code of the program whose type is being inferred. However, as noted in [21], Wand's algorithm sometimes fails to report reasons critical to the reconstruction of the unification failure, and at other times returns redundant information. Wand's paper leaves open the question of formulating a soundness and completeness criterion for type explanation.

While [55] and [5] rely on explicitly carrying type substitutions, Duggan and Bent's algorithm relies on the graph version of the Robinson unification algorithm in which substitutions are maintained by updating binding pointers of variables. Duggan and Bent also propose a path-based approach. In this respect, their approach is similar to ours. Unlike Wand, they do provide some formalization of the paths computed by their unification algorithm. However, this formalization seems to be tied to type inference (the main concern of their paper) rather than unification closure. Their paper also contains an elaborate discussion of the implications of using other unification algorithms for deriving type explanations. Because Duggan and Bent's algorithm also relies on manipulation of paths, it is worth comparing the unification algorithm for type explanation in (Duggan and Bent [21, Appendix C]) with our algorithm in some detail.

Our source-tracking algorithm works by computing unification path expressions and annotating binding pointers with them. But for the annotations, it is essentially the standard unification

algorithm. Duggan and Bent's algorithm, in contrast, relies on a somewhat more radical modification of the unification algorithm: unification now involves reversing the binding pointers of all variables in the paths from these variables to their respective roots. Unification of the three equations of Example 19 using Duggan and Bent's algorithm results in the unification graph of Fig. 16. The details of how the changes are done to the pointers are described in their algorithm and are skipped here. Since Duggan's and Bent's algorithm does not have the notion of a positive and negative traversal of edges, it relies on "accumulation" rather than algebraic composition of path information. Redundant equations are removed using a separate pass outside their unification algorithm. Since these paths are not treated as proof objects, elimination of duplicates seems to work, but their paper does not formalize this aspect.

Duggan and Bent also report problems with their algorithm in the presence of path compression, whereas our algorithm handles path compression easily. Path compression is handled by a much more complex algorithm (in Appendix D of their paper). Much of the complexity is due to addressing the related but separate problem of polymorphism. Another advantage of our algorithm is that it easily generates correct source-tracking information during subterm unification, as shown in Fig. 10. Duggan and Bent's algorithm relies on a different mechanism [21, Appendix C], which keeps track of unification histories during subterm unification.

Soosaipillai, in her Master's thesis [50] constructs a type explanation system that requires interactive navigation. Gomard [23] introduces partial type inference to isolate untypable parts of a program. Bernstein and Stark [6] consider an extended ML type system to locate untypable parts of a program.

Johnson and Walz [29,54] introduce "error-tolerant" unification, in which a multi-set of type constraints is solved by using a disjunction of constraints rather than a conjunction. For example, error-tolerant unification of the constraint multi-set

$$\{t \stackrel{?}{=} \texttt{bool}, t \stackrel{?}{=} \texttt{bool}, t \stackrel{?}{=} \texttt{int}\}$$

yields the solution

$$t = \texttt{int}[1]|\texttt{bool}[2].$$

This indicates that t has the 'options' `int` and `bool` with indicated strengths.

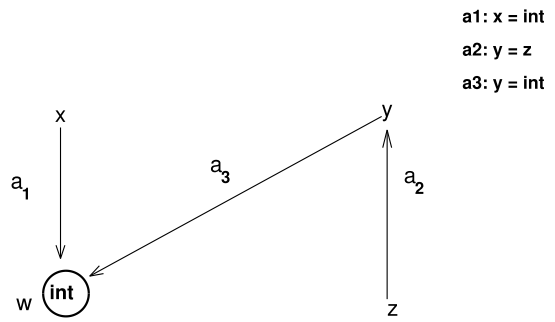

Fig. 16. Schematic view of result of path reversal using Duggan and Bent's unification algorithm [21].

Johnson and Walz's scheme rests on a complicated algorithm that derives implied type constraints obtained from the original type constraints by applying the rules of substitution and transitivity. Unfortunately, their work presents no correctness and completeness criterion against which their algorithm can be judged. Their approach, however, may be useful in conjunction with "soft typing" [9].

There has been considerable work in the last few years in the area of type error diagnosis using explanation-based and graphical front-ends, specially in the presence of polymorphic type inference. Yang's system [58] provides a visual front end for type explanation. Gandhe et al. [22] use Cox's pushdown automaton [18] to repair type errors in the simply typed λ-calculus. McAdam's thesis work also builds a framework for type repair in addition to locating type errors [39]. The M algorithm of Lee and Yi [35], uses a top-down version of Milner's original W algorithm [41] and shows that type errors are flagged "earlier" than the W algorithm. Yang et al. [59] propose an incremental type inference algorithm. Chitil [11] focuses on compositional type explanations. Haack and Wells [24] focus on the generation of minimal program slices which combines the use of a novel unification algorithm with a constraint collecting algorithm due to Damas [19]. Their analysis of type diagnosis is inspired by intersection types, while the recent work of Neubauer and Thiemann [42] employs disjoint unions. Many of these systems are based on the display and navigation of the unification graph in conjunction with the program graph and offer some form of automated generation of explanations.

Trace-based approaches for type error diagnoses have also been suggested. Early work here is Maruyama et al. [38]. The problem with tracing is the large output, often with redundancies. Recently, Heeren et al. [26] have proposed the use of type inference directives and specialized type rules to control the order of unification and type inference.

The unification path obtained by normalizing the output of our source-tracking unification algorithm may be considered as a slice of the unification graph. "Origin tracking" [7,52,53] has focused on a formal approach to program slicing based on term rewrite systems. This work has been applied to locating errors in statically typed languages with type checking but without type inference [20]. Term unification may also be approached as a rewriting system in which equations are transformed to a solved form if unifiable [33,37], in contrast to the relational approach of computing closures over a unification graph, which is closer to practical implementations. Results from origin-tracking are likely to be applicable to the transformational view of unification, but it appears that this approach is much harder than the one outlined in this paper.

We have related unification with a form of context-free reachability. Context-free language reachability, however, has had many applications to compiling and program slicing of functional and imperative programs. The main work here is Reps and others (see for example Melski and Reps [40] ans also Reps [46]), which connects program slicing, context-free reachability and set-based analysis.

9. Conclusions and future work

We have connected unification proofs with labeled-path problems over an important class of context-free languages, the semi-Dyck sets. The deduction system presented here can be used as a formalism to evaluate approaches for computing unification errors. The characterization of unification proofs in terms of labeled-path expressions allows us to employ context-free path algorithms

to compute minimum length unification proofs using context-free labeled shortest-path algorithms. On the other hand, a simple extension to the structure-sharing based unification algorithms allows unification proofs to be inexpensively constructed and simplified. This extension of the unification algorithm has been implemented in Scheme [12].

The design of fast algorithms for semi-Dyck labeled-path problems, which will help the efficient construction of optimized unification proofs, remains to be investigated. Some progress in this front has been made recently using a novel scheme for reduction from Dyck reachability to set constraints [32]. Also, it is worthwhile examining how the path-based framework proposed here can be extended to diagnosis of unification failure in higher-order unification, equational unification, and semi-unification. Finally, from a practical viewpoint, it will be valuable to integrate the unification source-tracking algorithm with logic programming systems for diagnosis of the success and failure of queries and with static type reconstruction systems for diagnosis of type errors. Preliminary work in the direction of applying the above framework to type error reconstruction has been reported in [13].

Acknowledgment

Critical feedback from the anonymous referees also helped considerably in improving the paper.

References

[1] F. Baader, T. Nipkow, Term Rewriting and All That, Cambridge University Press, Cambridge, 1998.
[2] F. Baader, J. Siekmann, Unification theory, in: D.M. Gabbay, C.J. Hogger, J.A. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, Oxford University Press, 1994, pp. 41–125 (chapter 2).
[3] C. Barrett, R. Jakob, M. Marathe, Formal language constraint path problems, SIAM J. Comput. 30 (2000) 809–837.
[4] L.D. Baxter. An efficient unification algorithm. Technical Report, University of Waterloo, 1973.
[5] M. Beaven, R. Stansifer, Explaining type errors in polymorphic languages, ACM Lett. Program. Lang. 2 (1–4) (1993) 17–30.
[6] K. Bernstein, E.W. Stark. Debugging type errors (full version). Nov. 1995. Unpublished Technical Report, Computer Science Dept. State University of New York at Stony Brook.
[7] Y. Bertot. Origin Functions in λ-calculus and Term Rewriting Systems, in: CAAP'92, Lecture Notes in Computer Science, vol. 581, Springer Verlag, 1992, pp. 49–65.
[8] R. Boyer, J.S. Moore, The sharing of structure in theorem proving programs, Mach. Intell. 7 (1972) 101–116.
[9] R. Cartwright, M. Fagan. Soft typing, in: Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation, 1991, pp. 278–292.
[10] T.Y. Chen, J. Lassez, G.S. Port, Maximal unifiable subsets and minimal nonunifiable subsets, New Generat. Comput. 4 (2) (1986) 133–152.
[11] O. Chitil. Compositional explanation of types and debugging of type errors, in: Proceedings of the 6th ACM SIGPLAN International Conference on Functional Programming (ICFP'01), ACM Press, September 2001, pp. 193–204.
[12] V. Choppella. Implementation of Unificationsource-tracking, July 2002. Available from: <http://www.cs.indiana.edu/hyplan/chaynes/unif.tar.gz>.
[13] V. Choppella. Polymorphic type reconstruction usingtype equations, in: P. Trinder, G. Michaelson, R. Peña (Eds.), Implementation of Functional Languages: 15th International Workshop, IFL 2003, Edinburgh, UK, Springer Verlag, December 2004, pp. 53–68.
[14] V. Choppella. Unification Source-tracking with Application to Diagnosis of Type Inference. PhD thesis, Indiana University, August 2002. IUCS Tech Report TR566.

[15] V. Choppella, C.T. Haynes. Source-tracking Unification, in: F. Baader (Ed.), Proceedings of 19th International Conference on Automated Deduction, CADE-19, Miami Beach, USA, Springer, 2003, pp. 458–472.
[16] J. Corbin, M. Bidoit, A rehabilitation of Robinson's unification algorithm, in: R.E.A. Mason (Ed.), Information Processing, Elsevier Science Publishers, North Holland, 1983, pp. 909–914.
[17] P.T. Cox, Finding backtrack points for intelligent backtracking, in: J.A. Campbell (Ed.), Implementations of Prolog, Prentice Hall, Englewood Cliffs, NJ, 1984, pp. 216–233.
[18] P.T. Cox, On determining the causes of non-unifiability, J. Logic Program. 4 (1) (1987) 33–58.
[19] L. Damas. Type assignment in Programming Languages. PhD thesis, University of Edinburgh, April 1985.
[20] T. Dinesh, F. Tip. A case-study of slicing-based approach for locating type errors, in: Second International Conference on the Theory and Practice of Algebraic Specifications, Amsterdam, Netherlands, British Computer Society, Springer, September 1997.
[21] D. Duggan, F. Bent, Explaining type inference, Sci. Comput. Program. 27 (1) (1996) 37–83.
[22] M. Gandhe, G. Venkatesh, A. Sanyal. Correcting type errors in the Curry system, in: V. Chandru, V. Vinay (Eds.), Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science, 1996, pp. 347–358.
[23] C.K. Gomard. Partial type inference for untyped functional programs, in: Proceedings of the 1990 ACM conference on LISP and functional programming, Nice, France, 1990, pp. 282–287.
[24] C. Haack, J. Wells. Type error slicing in higher order polymorphic languages, in: Programming Language and Systems, Proceedings of the 12th European Symposium on Programming, Springer, 2003, pp. 284–301.
[25] M.A. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.
[26] B. Heeren, J. Hage, S.D. Swierstra. Scripting the type inference process, in: Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ACM Press, Uppsala, Sweden, August 2003, pp.3–13.
[27] J. Herbrand. Recherches sur la The'orie de la Demonstration. PhD thesis. Reprinted in W.D. Goldfarb (Ed.), Logical Writings. Reidel, 1971.
[28] G. Huet. Resolution d'equations dans les languages d'ordre $1, 2, \ldots, \omega$. PhD thesis, University de' Paris, 1976. In French.
[29] G.F. Johnson, J.A. Walz. A maximum-flow approach to anomaly isolation in unification-based incremental type inference, in: Proceedings of the 13th ACM Symposium on Principles of Programming Languages (POPL'86), 1986, pp. 44–57.
[30] J.P. Jouannaud, C. Kirchner, Solving equations in abstract algebras: a rule based survey of unification, in: J. Lassez, G. Plotkin (Eds.), Computational Logic: Essays in honor of Alan Robinson, MIT Press, Cambridge, MA, 1991, pp. 257–321.
[31] K. Knight, Unification: a multidisciplinary survey, Comput. Surveys 21 (1) (1989) 93–124.
[32] J. Kodumal, A. Aiken. The set/CFL reachability connection in practice, in: Proceedings of the ACM SIGPLAN 2004 Conference on Programming Languauges Design and Implementation, Washington DC, USA, ACM Press, 2004, pp. 207–218.
[33] J. Lassez, M.J. Maher, K. Marriot, Unification revisited, in: J. Minker (Ed.), Deductive Databases and Logic Programming, Morgan Kaufmann, 1988, pp. 587–625 (chapter 15).
[34] P. Le Chenadec, On the logic of unification, J. Symbolic Comput. 8 (1) (1989) 141–199.
[35] O. Lee, K. Yi, Proofs about a folklore let-polymorphic type inference algorithm, ACM Trans. Program. Lang. 20 (4) (1998) 707–723.
[36] M. Marathe. Personal communication. May 2002.
[37] A. Martelli, U. Montanari, An efficient unification algorithm, ACM Trans. Program. Lang. Syst. 4 (2) (1982) 258–282.
[38] H. Maruyama, M. Matsuyama, K. Araki. Support tool and strategy for type error correction with polymorphic types, in: Proceedings of the Sixteenth Annual International Computer Software and Applications Conference, Chicago, IEEE, September 1992, pp. 287–293.
[39] B.J. McAdam, Repairing Errors in Functional Programs. PhD thesis, University of Edinburgh, 2002.
[40] D. Melski, T. Reps, Interconvertibility of a class of set constraints and context-free language reachability, Theor. Comput. Sci. 248 (1–2) (2000) 29–98.
[41] R. Milner, A theory of type polymorphism in programming, J. Comput. Syst. Sci. 17 (1978) 348–375.

[42] M. Neubauer, P. Thiemann. Discriminative sum types locate the source of type errors, in: Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ACM Press, Uppsala Sweden, 2003, pp. 15–26.
[43] M. Paterson, M. Wegman, Linear unification, J. Comput. Syst. Sci. 16 (2) (1978) 158–167.
[44] G.E. Peterson, M.E. Stickel, Complete sets of reductions for some equational theories, J. ACM 28 (2) (1981) 233–264.
[45] G.S. Port, A simple approach to finding the cause of non-unifiability, in: R.A. Kowalski, K.A. Bowen (Eds.), Logic Programming: Proceedings of the Fifth International Conference and Symposium, MIT Press, Cambridge, MA, 1988, pp. 651–665.
[46] T. Reps, Program analysis via graph reachability, in: J. Maluszynski (Ed.), International Symposium on Logic Programming, MIT Press, Cambridge, MA, 1997, pp. 5–19.
[47] J.A. Robinson, Computational Logic: the unification computation, Mach. Intell. 5 (1971) 63–72.
[48] J.A. Robinson, A machine-oriented logic based on the resolution principle, J. ACM 12 (1965) 23–41.
[49] P. Ružička, I. Prívara, An almost linear Robinson unification algorithm, Acta Informat. 27 (1) (1989) 61–71.
[50] H. Soosaipillai, An Explanation based polymorphic type checker for Standard ML. Master's thesis, Heriot-Watt University, 1990.
[51] R.E. Tarjan, Efficiency of a good but not linear set union algorithm, J. ACM 22 (2) (1975) 215–225.
[52] F. Tip, Generation of Program Analysis Tools. PhD thesis, Institute for Logic, Language and Computation, CWI, Amsterdam, 1995.
[53] A. van Deursen, P. Klint, F. Tip, Origin tracking, J. Symbolic Comput. 15 (1993) 523–545 (special issue on Automatic Programming).
[54] J.A. Walz, Extending Attribute Grammars and Type Inference Algorithms. PhD thesis, Cornell University, February 1989. 89-968.
[55] M. Wand, Finding the source of type errors, in: Proceedings of the 13th Annual ACM Symposium on Principles of Programming Languages (POPL'86), January 1986, pp. 38–43.
[56] M. Weiser, Program Slices: Formal, Psychological, and practical investigations of an automatic program abstraction method. PhD thesis, University of Michigan, Ann Arborpages, 1979.
[57] M. Weiser, Program slicing, IEEE Trans. Software Eng. 10 (4) (1984) 352–357.
[58] J. Yang, G. Michaelson, A visualisation of polymorphic type checking, J. Funct. Programm. 10 (1) (2000) 57–75.
[59] J. Yang, P. Trinder, G. Michaelson, J. Wells. Improved type error reporting, in: Proceeding of Implementation of Functional Languages, 12th International Workshop, September 2000, pp. 71–86.