Source-Tracking Unification

Venkatesh Choppella and Christopher T. Haynes

Computer Science Department
Indiana University
Bloomington, IN 47405, USA
{choppell,chaynes}@cs.indiana.edu

Abstract. We propose a practical path-based framework for deriving and simplifying source-tracking information for term unification in the empty theory. Such a framework is useful for debugging unification-based systems, including the diagnosis of ill-typed programs and the generation of success and failure proofs in logic programming.
The objects of source-tracking are deductions in the logic of unification. The semantics of deductions are paths over a unification graph whose labels form the language of suffixes of a semi-Dyck set. Based on this framework, two algorithms for generating proofs are presented: the first uses context-free shortest-path algorithms to generate optimal (shortest) proofs in time $O(n^3)$, 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.

1 Introduction

Unification failure indicates type errors in programming languages [3,11,13,28] and unsuccessful queries in logic programs[5,9]. The reporting of this failure can be confusing and insufficient for reconstructing the error. We present a unification source-tracking algorithm to identify proofs of non-unifiability and construct optimal (smallest sized) slices of the original term equations sufficient to prove unification failure.

Our framework for source-tracking rests on a fundamental property relating paths in labeled directed graphs to paths in their quotients under unification closure. The labeled directed graph underlying a unification graph is obtained by orienting and labeling each edge of the unification graph. 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 membership as paths whose labels form the suffix language of a semi-Dyck set (the language of balanced parentheses). This lets us think of unification closure purely in terms of connectivity via these special paths. On the basis of this characterization, we define an efficient *proof-based* unification source-tracking algorithm. To our knowledge, previous work on the diagnosis of

F. Baader (Ed.): CADE-19, LNAI 2741, pp. 458–472, 2003.

unification systems, including those arising in the context of type systems and logic programming, has not used this approach based on semi-Dyck sets.

The contributions of this work are:

1. Characterizing witnesses to membership in unification closure as a formal language path problem. This allows the use of formal language path algorithms to compute shortest proofs [2,19].
2. Defining a logic of unification path expressions for constructing witnesses of unification closure.
3. Inexpensive integration of witness computation with standard unification algorithms.
4. Simplification of witnesses computed by the unification source-tracking algorithm by elementary group rewriting.

The rest of this paper is divided as follows: Section 2 outlines the main ideas with an example. Section 3 defines the machinery of unification in terms of labeled directed graphs. Section 4 identifies source tracking with the problem of tracking the source of paths in the quotient graph in terms of paths in the original graph. Section 5 introduces unification path expressions. Section 6 shows how to inexpensively integrate the construction of unification path expressions into the unification algorithm. Section 7 shows how to simplify the unification expressions. Section 8 discusses related work. Section 9 concludes with suggestions for future work.

2 Motivating Example

The main ideas of this paper are informally introduced using the following set of term equations. (These are type equations whose derivation, with source-tracking as the central concern, is reported elsewhere[7].)

$$\begin{array}{lll} a : t_0 \stackrel{?}{=} t_1 \to t_2 & b : t_2 \stackrel{?}{=} t_4 & c : t_3 \stackrel{?}{=} \texttt{bool} \\ d : t_4 \stackrel{?}{=} t_5 & e : t_3 \stackrel{?}{=} t_1 & f : t_6 \stackrel{?}{=} t_7 \to t_4 \\ g : t_5 \stackrel{?}{=} t_1 & h : t_6 \stackrel{?}{=} \texttt{int} \to \texttt{int} & i : t_7 \stackrel{?}{=} t_1 \end{array}$$

The equations f and h, together with transitivity and symmetry, imply $t_7 \to t_4 \stackrel{?}{=} \texttt{int} \to \texttt{int}$. Equating subterms yields the derived equations $j : \texttt{int} \stackrel{?}{=} t_7$ and $k : \texttt{int} \stackrel{?}{=} t_4$. 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 unification graph of the system of term equations is shown in Figure 1. Variables and function symbol occurrences are represented as vertices. Thick *branch edges* represent the immediate subterm relation. Thin *equational edges* represent equations. Each equational edge is oriented in an arbitrary direction. Solid edges are original constraints and dashed edges are derived equations.

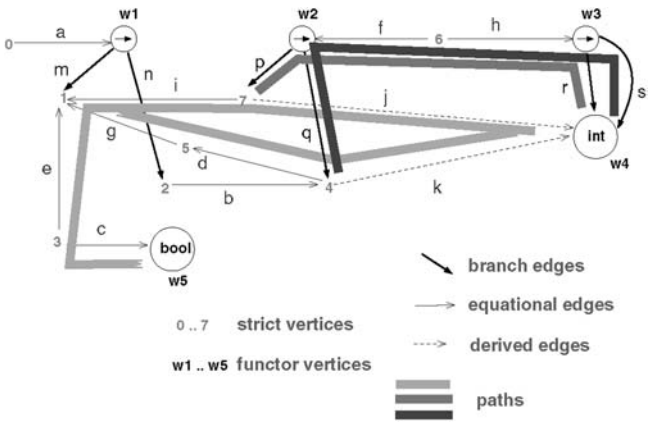

Fig. 1. Unification graph of example program. Each variable t_i is represented by the vertex i.

Unification may be viewed as establishing special connectivity relations between vertices in the graph. In the example, a proof for the equation $\texttt{int} \stackrel{?}{=} \texttt{bool}$ 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 Figure 1 are both witnesses to unsolvability. 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 $q^{-1}f^{-1}hs$.

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 unsolvability 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 Program Slices from Paths

Constructing program slices of the original set of term equations from these paths is straightforward: The set of edges $\{r, h\}$ corresponds to the weakening $t_6 \stackrel{?}{=} \texttt{int} \rightarrow \square$ of the constraint $h : t_6 \stackrel{?}{=} \texttt{int} \rightarrow \texttt{int}$. 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 $t_6 \stackrel{?}{=} t_7 \rightarrow \square$ obtained by replacing t_4 with $\square$ in f. The path $r^{-1}h^{-1}fpie^{-1}c$ consisting of segments $r^{-1}h$ and fp, along with edges i, e and c, therefore correspond to the following set E_1 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 E_2 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. The definition, derivation and simplification of these paths is the focus of the next few sections.

3 Notation and Basic Definitions

Our emphasis is on labeled directed graphs as a canonical representation of terms, term equations and unification graphs.

Given an alphabet Σ and $\epsilon \notin \Sigma$, let Σ^0, Σ^+ and Σ^* denote, respectively, the set $\{\epsilon\}$ containing the empty sentence ϵ, the set of finite non-empty sentences over Σ, and the set of finite sentences over Σ. A *signature* is an alphabet Σ of *functor symbols*, along with an arity function $\alpha : \Sigma \to \mathbf{N}$. 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* $T = \langle W, X, b\rangle$, where W is a set of *functor vertices* disjoint from $\mathbf{V}$, $X \subset \mathbf{V}$ is a set of *strict vertices*, and $b : W \longrightarrow \Sigma(W \cup X)$ is the *subterm function*, where for any set A, $\Sigma(A)$ is the set of terms $f(a_1, \ldots, a_{\alpha(f)})$, where $f \in \Sigma$ and for $1 \leq i \leq \alpha(f)$, $a_i \in A$. The term graph representation is closer to the actual implementation data structures of terms. They explicate 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. The standard way to represent a set of of term equations is by using a unification graph. A Σ-*unification graph* G is a a pair $\langle T, E\rangle$, where T is a Σ-term graph $\langle W, X, b\rangle$ and E is a relation over $W \cup X$. If $w \in W$ and $b(w) = f(u_1, \ldots, u_n)$, for some $u_1, \ldots u_n \in W \cup X$, then the *label* of w, denoted $L(w)$, is f.

To understand source-tracking, we consider the labeled directed graph underlying a unification graph. A *labeled directed graph (LDG)* 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 $\langle u, v, \delta\rangle \in D$, written $u \stackrel{\delta}{\longrightarrow} v$, denotes an edge *from* u *to* v whose *label* is δ. The function l projects the label δ from an edge $u \stackrel{\delta}{\longrightarrow} v$. Edges with labels in Σ are *branch edges* and those labeled ϵ are *equational edges*. An equational edge from a vertex to itself is called a *trivial edge*.

If Σ is a signature, then $\{f.i \mid f \in \Sigma, 1 \leq i \leq \alpha(f)\}$, denoted $\Sigma_{\mathbf{N}}$, is called the *projection alphabet of* Σ. The LDG *underlying* a Σ-unification graph $G =$

$\langle\langle W, X, b\rangle, E\rangle$ is $\langle \Sigma_{\mathbf{N}}, V, D\rangle$, where $V = W \cup X$, and D is the union of the set of branch edges $\{w \xrightarrow{f.i} u_i \mid b(w) = f(u_1, \ldots, u_i, \ldots u_n)\}$ and equational edges $\{u \xrightarrow{\epsilon} v \mid (u, v) \in E\}$. When there is no ambiguity, the LDG underlying a Σ-unification graph G is also denoted G.

We are interested in paths in LDG's and their labels. The label $l(p)$ of a path p is the concatenation of the labels of each of its edges. The label of an empty path is ϵ. The judgement $G \models u \xrightarrow{l} v$ denotes that there is a path from u to v labeled l in G.

Given an LDG $G = \langle \Sigma, V, D\rangle$, a relation R on V is *downward-closed* if for each uRu', $u \xrightarrow{\delta} v \in D$ and $u' \xrightarrow{\delta} v' \in D$ implies vRv'. The *unification closure* of an LDG of G, denoted $\sim$, is the least downward-closed equivalence on the vertices of G containing the equational edges of G. The *quotient graph* $G/\!\sim$ is the LDG $\langle \Sigma, V/\!\sim, D/\!\sim\rangle$ where $V/\!\sim$ is the set of equivalence classes of $\sim$, and for all vertices $u, v \in V$ and $\delta \in D$, $[u]_\sim \xrightarrow{\delta} [v]_\sim \in D/\!\sim$ if $u \xrightarrow{\delta} v \in D$. If $G = \langle W, X, b, E\rangle$ is a unification graph, the unification closure of the LDG underlying G is *homogeneous* if for each $w, w' \in W$, $w \sim w'$ implies $L(w) = L(w')$. It is well-known [21] that a unification graph G is unifiable if and only if the unification closure of its underlying LDG is homogeneous and the graph $G/\!\sim$ has no cycles whose labels are non-empty.

If $\Sigma = \{b_1, \ldots, b_n\}$, then Σ^{-1} is the alphabet $\{b_i^{-1} \mid b_i \in \Sigma\}$, assumed disjoint from Σ. The *inverse* G^{-1} of an LDG $G = \langle \Sigma, V, D\rangle$ is the LDG $\langle \Sigma^{-1}, V, D^{-1}\rangle$, where $D^{-1} = \{v \xrightarrow{\delta^{-1}} u \mid u \xrightarrow{\delta} v \in D\}$, and $\epsilon^{-1} = \epsilon$. The operation $inv : (\Sigma \cup \Sigma^{-1})^* \longrightarrow (\Sigma \cup \Sigma^{-1})^*$ is defined as $inv(\epsilon) = \epsilon$, $inv(c) = c^{-1}$ and $inv(c^{-1}) = c$ for $c \in \Sigma^{-1}$, and for $p, q \in (\Sigma \cup \Sigma^{-1})^*$, $inv(pq) = inv(q)inv(p)$.

3.1 Dyck and Semi-Dyck Sets

We will see that unification proofs can be labeled using semi-Dyck sets, which are languages with nice cancellative properties. As will be shown in the next section, this cancellation phenomenon is at the heart of unification closure.

Given an alphabet Σ, let $\mathbf{D}(\Sigma)$ and $\mathbf{D}'(\Sigma)$ denote the sets $\{b^{-1}b \approx \epsilon \mid b \in \Sigma\}$ and $\{bb^{-1} \approx \epsilon, b^{-1}b \approx \epsilon \mid b \in \Sigma\}$ of one-way and two-way cancellative identities, respectively. These identities, when oriented left to right, yield strongly normalizing rewrite systems. One step rewriting under $\mathbf{D}(\Sigma)$ and $\mathbf{D}'(\Sigma)$ is denoted $\longrightarrow_{\mathbf{D}(\Sigma)}$ and $\longrightarrow_{\mathbf{D}'(\Sigma)}$, respectively. If $x \in (\Sigma \cup \Sigma^{-1})^*$, let $\mu_{\mathbf{D}(\Sigma)}(x)$ and $\mu_{\mathbf{D}'(\Sigma)}(x)$ denote the unique normal forms under $\mathbf{D}(\Sigma)$ and $\mathbf{D}'(\Sigma)$ rewriting, respectively. If $L \subseteq (\Sigma \cup \Sigma^{-1})^*$, let

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

We are primarily interested in the languages $D(\Sigma, L)$ when L is Σ^0, Σ^+, and Σ^*. These are abbreviated $D^0(\Sigma)$, $D^+(\Sigma)$, and $D^*(\Sigma)$, respectively. $D^0(\Sigma)$ is known as the *semi-Dyck set over* Σ. It is the set of balanced parentheses sentences whose left and right parenthesis symbols are drawn from Σ^{-1} and Σ, respectively.

Clearly, $D^0(\Sigma)$ and $D^+(\Sigma)$ are disjoint and $D^* = D^0 \cup D^+$. The language D^* is the set of all suffixes of D^0 sentences, and is suffix-closed [12]. Informally, D^+ is the set of "unbalanced" suffixes of sentences of balanced parentheses. The languages D^0, D^+, and D^* may be generated using context-free grammars:

$$\begin{array}{lll} D^0 & ::= \epsilon \mid D^0 b^{-1} D^0 b D^0 & b \in \Sigma \\ D^+ & ::= D^* b D^* & b \in \Sigma \\ D^* & ::= D^0 \mid D^+ & \end{array}$$

4 Source-Tracking

Let $G = \langle \Sigma, V, D \rangle$ be an LDG 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 $l(p) \in D^*(\Sigma)$. We are now ready to state the first part of the main result of this paper, which characterizes connectivity in the quotient graph $G/\sim$ in terms of unification paths 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.

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

Proof. By induction on the derivation of l in the grammar $D^*(\Sigma)$.

Unification paths are complete with respect to connectivity in the $\sim$-quotient of an LDG.

Theorem 2. *(Unification path completeness)*
Let $G = \langle \Sigma, V, D \rangle$ be an LDG with unification closure $\sim$. If $G/\sim \models [u]_\sim \xrightarrow{l'} [v]_\sim$, then $G \cup G^{-1} \models u \xrightarrow{l} v$ for some $l \in D^*(\Sigma)$ such that $\mu_{\mathbf{D}(\Sigma)}(l) = l'$.

Proof. By induction on path construction in $G/\sim$.

4.1 Computation of Shortest Unification Paths

Theorems 1 and 2 show how unification source-tracking information may be encoded as unification paths. One measure of the succinctness of this information is its length. Since unification paths are paths over a graph whose labels are constrained by the context-free grammar (CFG) $D^*(\Sigma)$ for an alphabet Σ, computation of the shortest unification path is a special case of the context-free path problem. If G is a directed graph whose edges are labeled from an alphabet Σ, and $\mathcal{L}$ is a context-free language over Σ, the CFG 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 sentence in $\mathcal{L}$.

Shortest unification paths may therefore be computed using the dynamic-programming-based CFG shortest-path algorithms of Barrett et al. [2][1] (See also Melski and Reps [19]). If $\mathcal{L}$ 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 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 $D^*(\Sigma)$ is of size $O(\Sigma)$ and may be transformed into a grammar in CNF 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 extension to the unification algorithm that efficiently computes a non-optimal path which, in practice, may be adequate for the purpose of diagnosis. Our extension to the unification algorithm computes *unification path expressions*, which are unification paths extended with an inverse operation. These expressions may be informally thought of as the execution "trace" of a particular inference made by the unification algorithm.

5 Unification Path Expressions

The construction of unification path expressions over an LDG $G = \langle \Sigma, V, D \rangle$ is defined inductively using the system P^U of rules shown in Figure 2. The rules may be thought of as a type system whose untyped terms are drawn from the term algebra $\mathcal{T}(\Sigma_{\mathbf{Gr}}, D)$ generated by D, where $\Sigma_{\mathbf{Gr}} = \{\epsilon \mapsto 0, (.)^{-1} \mapsto 1, \circ \mapsto 2\}$ 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 Figure 2. The triple $u \xrightarrow{l} v$ is the "type" of the path expression p. The interesting rule here is DN, which given a path p connecting u' to v', connects the child u of u' with child v of v'. This connection involves traversing in reverse direction the edge connecting u' to u, with the path c^{-1} labeled δ^{-1}. This is concatenated with the path p labeled ϵ and the path c', labeled δ which consists of an edge from v' to v. The label of the resultant path $c^{-1}pc'$ is $\delta^{-1}\epsilon\delta$. This simplifies to ϵ, which is the "net" label associated with the path from u to v.

Each term $p \in T^*(\Sigma_{\mathbf{Gr}}, D)$ can be "flattened" to a unique sentence in $(D \cup D^{-1})^*$: *flatten*(p), abbreviated $\overline{p}$, is defined by $\overline{\epsilon} = \epsilon$, $\overline{p^{-1}} = inv(\overline{p})$, and $\overline{pq} = \overline{p}\,\overline{q}$.

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

[1] The algorithm is unaffected if labels on edges are drawn from $\Sigma \cup \{\epsilon\}$, where ϵ is the empty sentence [17].

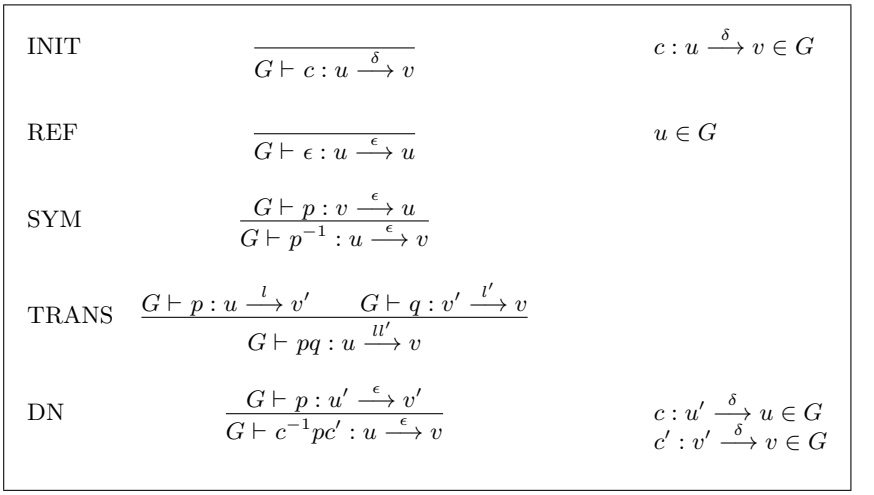

Fig. 2. The logic $P^U(G)$ of unification path expressions over an LDG $G = \langle \Sigma, V, D \rangle$.

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

Proof. Soundness is by induction on P^U deductions. Completeness is by induction on the derivation of unification path labels.

From this and theorems 1 and 2, the soundness and completeness of P^U deductions with respect to connectivity in the $\sim$-quotient graph follows.

Theorem 3. *(Soundness and completeness of P^U deductions with respect to unification closure)*
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 $G/\!\sim \models u \xrightarrow{l} v$.*
2. *(Completeness) If $G/\!\sim \models u \xrightarrow{l} v$, then $G \vdash_{P^U} p : u \xrightarrow{l} v$ where p is some $\Sigma_{\mathbf{Gr}}$-term over D.*

6 Unification Algorithm with Source-Tracking

Construction of unification path expression deductions is easily integrated into the standard unification algorithm to yield an algorithm that computes proof

of membership in the unification closure of a unification graph. To illustrate the integration of computation of P^U-deductions with unification, we pick the quadratic-time unification algorithm of Corbin and Bidoit[8] as presented in (Baader and Siekmann [1]) although other unification algorithms may be used as well. The resulting algorithm is shown in Figure 3. The binding field of each variable and the return value of *find* is a tuple containing two objects: a pointer to the root of an equivalence class and a unification path expression denoting the path expression to root. The procedures *unify* and *union* also carry an extra parameter that is a unification path expression. 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 in the unification graph containing the vertices u and v.

Theorem 4. *(Invariants for Unification algorithm with source-tracking)*
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 be unify(v_1, v_2, m). Then, the the following invariants are maintained:

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. *If find$(u) = \langle v, p \rangle$, then $G \vdash_{P^U} p : u \xrightarrow{\epsilon} v$.*
4. *If occurs?$(u, v) =$ yes(x, p), then $x \in \{0, +\}$, $G \vdash_{P^U} p : u \xrightarrow{l} v$ and $l \in \Sigma^x_{\mathbf{N}}$, where $\Sigma_{\mathbf{N}}$ is the set of projection labels in G.*

Proof. By inspection of the path expressions constructed at each stage of the algorithm.

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 1. *(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$(CYCLE, q)$, then $G \vdash_{P^U} q : u \xrightarrow{l} u$ for some vertex u in G and label $l \in \Sigma^+_{\mathbf{N}}$.*
2. *If unify $=$ fail$(CLASH, q)$, then $G \vdash_{P^U} q : w \xrightarrow{\epsilon} w'$ for some functor vertices w, w' in G such that $L(w_1) \neq L(w_2)$.*

```
procedure unify(v1, v2, m) =
  let ⟨r1, p1⟩ = find(v1) and ⟨r2, p2⟩ = find(v2)
    in if r1 = r2 then return
      else case r1.type, r2.type
        strict, strict: union(r1, r2, p1^-1 m p2)
        functor, strict: unify(v2, v1, m^-1)
        strict, functor: let ans = occurs?(r2, r1)
                         in case ans
                            no: union(r1, r2, p1^-1 m p2)
                            yes(_ , q): fail(CYCLE, p1^-1 m p2 q)
        functor,functor:
          if r1.L ≠ r2.L then fail(CLASH, p1^-1 m p2)
          else
            union(r1, r2, p1^-1 m p2);
            for i = 1 to α(r1.L) do
              unify(r1.child(i), r2.child(i), b1^-1 p1^-1 m p2 b2)
                where b1 = edge(r1, r1.child(i))
                  and b2 = edge(r2, r2.child(i))

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

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

procedure occurs?(v1, v2) =
  let ⟨r1, p1⟩ = find(v1) and ⟨r2, p2⟩ = find(v2)
    in if r1 = r2 then return yes(0, p1 p2^-1)
      else case r1.type
        strict: return no
        functor:
          for i = 1 to α(r1.L) do
            let ans = occurs?(r1.child(i), r2)
              in case ans
                 no: continue
                 yes(_, q): return yes(+, p1 b q p2^-1)
                       where b = edge(r1, r1.child(i))
          return no
```

Fig. 3. Unification algorithm with source-tracking.

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 [22].

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 2. *(Decomposition of normal forms)*
If D is any set and $p \in T^*(\Sigma_{\mathbf{Gr}}, D)$, then $\mu_{\mathbf{Gr}}(p) = \mu_{\mathbf{D}'(D)}(\overline{p})$.

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 $\mathbf{D}'(D)$ are the same.

Unification path expressions are not closed under one-step R/A rewriting (thus P^U lacks subject reduction), but types "lost" after one step of rewriting are recovered at normalization.

Example 1. (One-step rewriting in R/A does not "preserve types," but normalization recovers them.)
Let G be an LDG consisting of the edges

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

Let $p = (b_1^{-1}(ab_2))^{-1}$. Clearly $G \vdash_{P^U} p : v \xrightarrow{\epsilon} u$. If $q = (ab_2)^{-1}(b_1^{-1})^{-1}$, then $p \longrightarrow_{R/A} q$, but $G \not\vdash_{P^U} q : v \xrightarrow{\epsilon} u$. However, $G \vdash_{P^U} b_2^{-1}a^{-1}b_1 : v \xrightarrow{\epsilon} u$, 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. In order to show this, we rely on the following property, easily proved about Dyck languages, which states that reduction by two-sided cancellations on $D^*(\Sigma)$ sentences can be simulated by one-sided cancellation:

Lemma 3. *(Closure of $D^*(\Sigma)$ sentences under $\mathbf{D}'(\Sigma)$ reduction)*
If $x \in D^*(\Sigma)$ and $x \longrightarrow_{\mathbf{D}'(\Sigma)} y$, then $x \approx_{\mathbf{D}(\Sigma)} y$ and $\mu_{\mathbf{D}(\Sigma)}(x) = \mu_{\mathbf{D}(\Sigma)}(y)$.

Proof. By induction on the derivation of $D^*(\Sigma)$ sentences.

Theorem 5. *(P^U weak subject reduction)*
Let G be an LDG 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 1, $G \vdash_{P^U} p : u \xrightarrow{l} v$ implies $G \vdash_{P^U} \overline{p} : u \xrightarrow{l} v$. By Lemma 2, $p' = \mu_{\mathbf{D}'(D)}(\overline{p})$. Since $\overline{p} \xrightarrow{*}_{\mathbf{D}'(D)} p'$, it follows that $l(\overline{p}) \xrightarrow{*}_{\mathbf{D}'(\Sigma)} l(p')$. Since $l(\overline{p}) \in D^*(\Sigma)$, by Lemma 3, $\mu_{\mathbf{D}(\Sigma)}(l(p'))$ is equal to $\mu_{\mathbf{D}(\Sigma)}(l(\overline{p}))$, which is l. Then $G \vdash_{P^U} p' : u \xrightarrow{l} v$ follows from Lemma 1 (Completeness).

7.1 Efficiency Considerations

The cost of constructing unification path expressions at each point in the algorithm 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 increases 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 an LDG 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

A more detailed survey is reported elsewhere [7].

8.1 Diagnosis of Type Inference

Wand [28] modified the unification algorithm to accumulate reasons when traversing the the unification graph. Unfortunately, Wand did not show how the reasons together simulate the error. More importantly, Wand's algorithm sometimes fails to report reasons critical to the reconstruction of the unification failure, and at other times returns much redundant information. Eliminating redundant inferences from reason lists in a systematic way can be done by abandoning the algorithm's set-based approach, and instead using the path-based approach suggested by our algorithm.

Johnson and Walz [13,27] introduce "error-tolerant" unification, in which a multi-set of type constraints is solved by using a disjunction of constraints rather than a conjunction. Their 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.

The attribute grammar approach of Johnson and Walz and others confuses the generation of type constraints, which is directed by the syntax of the program, with the solution of these constraints, which is directed by the geometry of the constraints themselves. In any unification-based type system, types are more accurately viewed as flowing along special paths of the unification graph,

not through the source code. Other approaches include interactive explanation-based systems[11], but these lack automation because the user is expected to navigate the unification graph.

Polymorphic type inference algorithms designed specifically for better type error reporting include the M algorithm of Lee and Yi [16], which is a top-down version of Milner's original W algorithm [20], and the incremental type inference algorithm of Yang et al.[29]. Because these algorithms are built on *top* of unification, they are unable to remove redundant inferences introduced by the underlying unification process.

The output of our diagnostic unification algorithm may be considered as a slice of the unification graph. "Origin tracking" [4,25,26] 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 [10]. Term unification may also be approached as a rewriting system in which equations are transformed to a solved form if unifiable [14,18], in contrast to the relational approach of solving unification [21], which is closer to implementation. Results from origin-tracking are likely to be applicable to the transformational view of unification, but we feel that this approach is much harder than the approach outlined in this paper. Other relevant work is the path-based program slicing of Reps and others (see for example [19,24]), which connects program slicing, context-free reachability and set-based analysis.

8.2 Logic Programming and Unification-Based Systems

Port [23] carried out unification failure analysis by identifying minimally unifiable subsets. Port's algorithm attempts to construct regular path expressions over the unification graph. Our work shows that the path expressions of relevance are context-free, not regular.

Cox [9] and Chen et al. [5] 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. On the other hand, the simplification framework proposed in this paper does not address minimality.

Le Chenadec [15] introduces a framework of logical systems to characterize unification. The logic 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 Le Chenadec's logics involves terms and contexts, whereas P^U is a logic of labeled paths on graph vertices. This connection with context-free languages makes it possible for us to extract practical algorithms for proof construction.

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. Our semantic char-

acterization of unification proofs shows how unification proofs may be computed independent of any unification algorithm. This characterization allows us to employ different implementations to compute these unification proofs. A simple extension to the unification algorithm allows unification proofs to be inexpensively constructed and simplified. This extension of the unification algorithm has been implemented in Scheme [6]. On the other hand, computation of minimum length unification proofs can be implemented by using shortest-path CFG algorithms.

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. Also, it is worthwhile to examine how the path-based framework proposed here can be extended to diagnosis of unification failure in higher-order, equational, 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.

References

1. Baader, F., and Siekmann, J. Unification theory. In *Handbook of Logic in Artificial Intelligence and Logic Programmaning*, D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Eds. Oxford University Press, 1993.
2. Barrett, C., Jakob, R., and Marathe, M. Formal language constraint path problems. *SIAM Journal of Computing 30* (2000), 809–837.
3. Beaven, M., and Stansifer, R. Explaining type errors in polymorphic languages. *ACM Letters on Programming Languages* (1994).
4. Bertot, Y. Origin Functions in λ-calculus and Term Rewriting Systems. In *CAAP'92* (1992). Springer Verlag LNCS 581.
5. Chen, T. Y., Lassez, J.-L., and Port, G. S. Maximal unifiable subsets and minimal non-unifiable subsets. *New Generation Computing* (1986), 133–152.
6. Choppella, V. Implementation of unification source-tracking. `http://www.cs.indiana.edu/hyplan/chaynes/unif.tar.gz`, July 2002.
7. Choppella, V. *Unification Source-tracking with Application to Diagnosis of Type Inference*. PhD thesis, Indiana University, August 2002. IUCS Tech Report TR566.
8. Corbin, J., and Bidoit, M. A rehabilitation of Robinson's unification algorithm. In *Information Processing* (1983), R. E. A. Mason, Ed., Elsevier Science Publishers (North Holland), pp. 909–914.
9. Cox, P. T. Finding backtrack points for intelligent backtracking. In *Prolog Implementation*, J. Campbell, Ed. 1984, pp. 216–233.
10. Dinesh, T., and Tip, F. A case-study of slicing-based approach for locating type errors. In *Proc. 2nd International Conference on the Theory and Practice of Algebraic Specifications (ASF+SDF'97)* (September 1997).
11. Duggan, D., and Bent, F. Explaining type inference. *Science of Computer Programming 27*, 1 (July 1996), 37–83.
12. Harrison, M. A. *Introduction to Formal Language Theory.* Addison-Wesley, 1978.
13. Johnson, G. F., and Walz, J. A. A maximum-flow approach to anomaly isolation in unification-based incremental type inference. In *Proceedings of the 13th ACM Symposium on Programming Languages* (1986), pp. 44–57.

14. Lassez, J., Maher, M. J., and Marriot, K. Unification revisited. In *Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, 1988, ch. 15, pp. 587–625.
15. Le Chenadec, P. On the logic of unification. *Journal of Symbolic computation 8*, 1 (July 1989), 141–199.
16. Lee, O., and Yi, K. Proofs about a folklore let-polymorphic type inference algorithm. *ACM Transactions on Programming Languages 20*, 4 (July 1998), 707–723.
17. Marathe, M. personal communication, May 2002.
18. Martelli, A., and Montanari, U. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst. 4*, 2 (April 1982), 258–282.
19. Melski, D., and Reps, T. Interconvertibility of a class of set constraints and context-free-language reachability. *Theoretical Computer Science 248*, 1-2 (Nov 2000), 29–98.
20. Milner, R. A theory of type polymorphism in programming. *Journal of Computer and System Sciences 17* (1978), 348–375.
21. Paterson, M., and Wegman, M. Linear unification. *J. Comput. Syst. Sci. 16*, 2 (1978), 158–167.
22. Peterson, G. E., and Stickel, M. E. Complete sets of reductions for some equational theories. *Journal for the ACM 28*, 2 (April 1981), 233–264.
23. Port, G. S. A simple approach to finding the cause of non-unifiability. In *Logic Programming: Proceedings of the Fifth International Conference and Symposium* (1988), R. A. Kowalski and K. A. Bowen, Eds., MIT Press, pp. 651–665.
24. Reps, T. Program analysis via graph reachability. In *International Symposium on Logic Programming* (1997), J. Maluszynski, Ed., MIT Press, pp. 5–19.
25. Tip, F. *Generation of Program Analysis Tools.* PhD thesis, Institute for Logic, Language and Computation, CWI, Amsterdam, 1995.
26. van Deursen, A., Klint, P., and Tip, F. Origin Tracking. *Journal of Symbolic Computation 15* (1993), 523–545. Special issue on automatic programming.
27. Walz, J. A. *Extending Attribute Grammars and Type Inference Algorithms.* PhD thesis, Cornell University, February 1989. TR 89-968.
28. Wand, M. Finding the source of type errors. In *13th Annual ACM Symp. on Principles of Prog. Languages.* (January 1986), pp. 38–43.
29. Yang, J., Trinder, P., Michaelson, G., and Wells, J. Improved type error reporting. In *Proceeding of Implementation of Functional Languages, 12th International Workshop* (September 2000), pp. 71–86.