

# CAHIERS DE TOPOLOGIE ET GÉOMÉTRIE DIFFÉRENTIELLE CATÉGORIQUES

DANA MAY LATCH

**The connection between the fundamental  
groupoid and a unification algorithm for syntactil  
algebras (extended abstract)**

*Cahiers de topologie et géométrie différentielle catégoriques*, tome  
32, n° 3 (1991), p. 203-242

[http://www.numdam.org/item?id=CTGDC\\_1991\\_\\_32\\_3\\_203\\_0](http://www.numdam.org/item?id=CTGDC_1991__32_3_203_0)

© Andrée C. Ehresmann et les auteurs, 1991, tous droits réservés.

L'accès aux archives de la revue « Cahiers de topologie et géométrie différentielle catégoriques » implique l'accord avec les conditions générales d'utilisation (<http://www.numdam.org/conditions>). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

NUMDAM

Article numérisé dans le cadre du programme  
Numérisation de documents anciens mathématiques  
<http://www.numdam.org/>

# THE CONNECTION BETWEEN THE FUNDAMENTAL GROUPOID AND A UNIFICATION ALGORITHM FOR SYNTACTIL ALGEBRAS

(Extended Abstract)

by Dana May LATCH\*

**RÉSUMÉ:** Commenant par une grammaire 'contexte-libre', nous construisons une algèbre de chemins de l'algèbre syntaxique de cette grammaire. Nous démontrons que (a) décider s'il existe un ensemble fini d'homotopies engendrant le groupoïde fondamental de l'algèbre de chemins et (b) décider s'il existe un algorithme fini d'unification pour l'algèbre de termes de l'algèbre syntaxique, sont équivalents. Ces deux problèmes équivalent au problème d'équivalence de grammaires 'contexte-libres', qui est non-décidable (au sens de Turing). Nous donnons des conditions suffisantes, qui impliquent l'existence d'un ensemble fini d'homotopies engendrant le groupoïde fondamental.

## 1. Introduction

This paper deals with the connection between two very different areas: unification theory from computer science and algebraic topology of algebras / small categories from mathematics. Unfortunately, the first part of this connection deals with a class of *new* Turing undecidable problems (that is, decision problems that are equivalent to deciding the halting of a Turing machine). We add to a class of equivalent Turing undecidable problems for context-free languages, problems dealing with unification theory and homotopy theory. It is well known (for example, see [Ha78], [HoU179], or [DaWe83]) that the following equivalent problems are undecidable. Let:  $G$ ,  $G_1$  and  $G_2$  be context-free grammars (CFG's).

---

\*On leave 1985-1988 at CUNY; Work partially supported by NSF Grants #MCS81-04217; #DCR83-02897 and by CUNY Grants #PSC-CUNY-668293.

- (1) *Ambiguity Problem*: Deciding whether  $G$  is ambiguous (that is, whether there exists more than one way of generating an expression string using  $G$ ) .
- (2) *Intersection Problem*: Determining whether the context-free languages  $L(G_1)$  and  $L(G_2)$  generated by  $G_1$  and  $G_2$  respectively, have any expression strings in common (that is, determining whether  $L(G_1) \cap L(G_2) = \emptyset$ ) .
- (3) *Grammar Equivalence Problem*: Determining whether  $G_1$  and  $G_2$  , are equivalent (that is whether  $L(G_1) = L(G_2)$ ) .

In this paper we show that the following decision problems are equivalent to the above undecidable problems for context-free languages:

- (4) *Unification Problem*: Determining whether there exists a set of most general unifiers (mgu's) for any two terms in  $T(G)$  , the monoidal term algebra generated by  $G$  [Ru 87].
- (5) *Homotopy Generation Problem*: Determining whether, for the path algebra  $P(G)$  generated by  $G$  , the *strong* fundamental groupoid is isomorphic to the *weak* fundamental groupoid (that is, whether there exist *enough* strong "generating homotopies" in  $P(G)$ ) .
- (6) *Homotopy Extension Problem*: Determining whether, for each pair of paths in  $P(G)$  , there exists a minimal set of strong homotopy extensions to elements of the fundamental groupoid

The Unification and Homotopy Extension Problems are equivalent to the Ambiguity and Intersection Problems, while the Homotopy Generation Problem is equivalent to the undecidable Ambiguity Generation Problem which contains the Ambiguity Problem:

- (1') *Ambiguity Generation Problem*: Determining whether there exists a (finite) set of ambiguity pairs that generates the ambiguity/pre-order congruence relation in the path algebra  $P(G)$  .

Fortunately, the second part of the connection between unification theory and homotopy theory, deals with sufficient conditions [La89a] on the path algebra  $P(G)$  of a CFG  $G$  so that a unification algorithm for the term algebra  $T(G)$  exists, and equivalently, so that strong generating homotopies for the fundamental groupoid of the path algebra  $P(G)$  exist. For a path algebra  $P(G)$  that satisfies these sufficient conditions, by examining the geometric "path structure" of  $P(G)$  along with the "punctuation structure" of  $G$  , we are able to describe [La89a] how to generate the unification algorithm for  $T(G)$  (using a "diamond completion" procedure). In [La89a], a nontrivial CFG class  $G_{FP}$  , used to generate

expressions for a class of formal functional programming (FFP) languages [Ba78], is shown to satisfy these conditions. In this paper, for  $G_{FP}$ , we present both generating most general unifiers for the term algebra  $T(G_{FP})$  and generating homotopies for the strong fundamental groupoid of the path algebra  $P(G_{FP})$ .

### 1.1 Unification Theory

Unification theory plays an important role in computer science; for example, in conjunction with Robinson's resolution method [Ro65] for Horn clause logic, unification is the best method of evaluation of programs in PROLOG [ShSt87]. The literature dealing with unification is enormous and unification is a basic tool in many areas of computer science, including: computational complexity, data structures and algorithms, automated theorem proving, logic programming, higher order logic, functional programming languages and polymorphism, feature structures, natural language processing, equational theories for term rewriting systems, lattice theory, type inference, machine learning, etc. An introductory survey to unification theory is given in [Kn89].

J.Siekmann presents a formal theory for and an extensive survey of the theory of equational unification in [Si89]. D.Rydeheard explores applying category theory to unification algorithms with respect to the empty equational theory [BuRy-86] and extends this categorical approach to combining unification algorithms in [RySt87]. In [Baa89], F.Baader shows that unification for commutative theories can be characterised categorically as those which have semi-additive categories of "substitution morphisms". In [BHS89], H.-J.Bürchert, A.Herold, and M.Schmidt-Schauss explore decidability questions of unification for various classes of equational theories, by exploiting properties of their associated monads.

However, probably because of the undecidability of the Unification Problem for context-free term algebras, there has been little research on the development of unification algorithms for these algebras. In [La89a] sufficient conditions are given for the existence of unification algorithms for context-free algebras, and applications of these algorithms are presented in [La89b], [LaSi88] and [LSR90]. In [La89a] and [La89b], these algorithms are shown to exist for the class  $G_{FP}$ , of ambiguous CFG's for FFP languages [Ba78].

An important reason for developing unification algorithms for syntactic algebras based on ambiguous CFG's for FFP languages is that the collection of these unification algorithms acts as a basic tool in the development of a pro-

gram verification and analysis system for functional programs (see [LaSi88] and [LSR90]). The input to the verification and analysis system is a nonconstructive denotational semantics for the primitives (that is, the primitive functions and functional/combining forms) of the FFP language and an operational semantics in the form of a (possible nondeterministic) collection of rewrite rules. For a language primitive, the operational semantics can contain head recursive rules, tail recursive rules, or rules which employ combinations of head and tail recursion.

The (nonconstructive) semantics describes the transformational effect of a primitive on its operand(s) (possibly resulting in recursive applications of the language primitives). The performance of this transformation might be considered a "macro-operation", or machine instruction, of the interpreter, occurring in a single "instruction cycle". The operational rewrite rules describe the "micro-operations" that implement a language primitive, where a basic "machine-cycle" is occupied by a single rewrite operation.

Various defined rewrite rule collections can reflect the operational nature of diverse target architectures, while the nonconstructive semantics hides this level of detail from the language user. Typically, an operational semantics will assign a fixed recursive structure to compound objects. But, at the operational level, it may be appropriate to consider lists (a basic data structure in functional programming languages) variously as *Lisp*-like lists, tail-constructed lists, a mixture of the two, or concatenated sublists, depending on the context. For example, the *last* function that selects the rightmost element of a list has a minimal cost with a right-constructed list  $\langle l_1 x_1 \rangle$  (where  $l_1$  represents an expression list and  $x_1$  denotes a single expression component), and a maximal cost with a left-constructed list  $\langle x_2 l_2 \rangle$ . For the functional form *apply-to all* (which is similar to the LISP higher-order primitive *map*) that applies a function  $f$  to each component of an operand list  $\langle x_1 \dots x_n \rangle$  returning a list of applications  $\langle (f : x_1) \dots (f : x_n) \rangle$ , the format of the constructed operand list does not matter. For *apply-to-all*, the interpreter must construct a list with  $n$  top-level applications independent of the construction of the operand list  $\langle x_1 \dots x_n \rangle$ ; hence, the number  $n$  of the top-level components of the operand list is the input parameter to the cost measure.

Different structural interpretations, reflected in distinct operational semantics, could require separate validations. The verification and analysis system, presented in

[LaSi88] and [LSR90], permits ambiguity in the operational "deconstruction" of compound structures. Thus, for example, lists can be treated variously by different rewrite rules as left-constructed, right-constructed or a mixture of the two. Validation of an operational semantics that permits this kind of ambiguity may be considered the certification of a class of deterministic interpreter models, or of one or more non-deterministic models, one for each subset of the rewrite rules adequate to implement the denotational semantic specification.

The user of the FFP verification and analysis system must provide a set of rewrite rules based on a monoidal context-free term algebra. The system must be able to determine a finite set of mgu's [Si89] for any pair of left-hand-sides, because it must be possible to recognize when two or more rewrite rules (for the same primitive) can be used to evaluate the same applicative expression. Moreover, each left-hand-side corresponds to a sentential form derivation in the grammar, and represents the set of all expressions that match it. Describing all expressions that match more than one left-hand-side is equivalent both to giving a unification algorithm for the term algebra (that is, finding a set of mgu's) and to finding a minimal partition of the intersection of the left-hand-side expression sets or context-free sublanguages [La89a]. Unfortunately, since we are not doing simple syntactic unification (where unifiable terms must have the same structure), two left-hand-sides may have more than one, indeed an infinite number of, mgu's. Each mgu or equivalent intersection component represents a separate case that must be examined by the verifier. More specifically, then, each pair of terms must have only a *finite number* of mgu's (that is, there exist an equational unification algorithm of finite type [Si89]). The equational unification theory identifies two differently constructed expression lists whenever their components are identified and is the pre-order/ambiguity congruence in the syntactic path algebra of the FFP grammar. In [La89a], sufficient conditions for the existence of such unification algorithms are given. However, there is no hope of determining necessary conditions, because, in Section 2.3, the Unification Problem for context-free term algebras is shown to be equivalent to the undecidable Grammar Equivalence Problem.

## 1.2 Homotopy Theory

Let  $A$  be a small category and  $Ab$  be the category of abelian groups. In [La79] it is shown that derived functors of col-

mit

$$\text{colim} : Ab^A \longrightarrow Ab$$

form *the* homology theory for  $\mathcal{E}$ , the category of small categories, and that derived functors of limit

$$\text{lim} : Ab^A \longrightarrow Ab$$

form *the* cohomology theory for  $\mathcal{E}$ . The *homological dimension* of  $A$ , denoted  $\text{hd}(A)$ , is the highest non-vanishing dimension of the homology (i.e.  $\text{hd}(A) = \max\{k \mid \text{colim}_k \neq 0\}$ ) and the *cohomological dimension* of  $A$ , denoted  $\text{cd}(A)$ , is the highest non-vanishing dimension of the cohomology (that is,  $\text{cd}(A) = \max\{k \mid \text{lim}^k \neq 0\}$ ). In [La73] and [LaMi74], it is shown that, if the *cardinality* of  $A$  is  $\aleph_n$ , then

$$\text{cd}(A) - \text{hd}(A) \leq n + 1.$$

In addition, if  $A$  is a directed poset with smallest cardinality of a cofinal subset being  $\aleph_n$ , then  $\text{cd}(A) = n + 1$ .

A search for why the cardinality of a small category implies such a "topological" theorem resulted in:

- A weak homotopy equivalence between  $\mathcal{E}$  and the category  $\mathcal{W}$  of CW complexes that was induced by the composition of the nerve functor  $N : \mathcal{E} \rightarrow \mathcal{K}$  from  $\mathcal{E}$  to the category  $\mathcal{K}$  of simplicial sets, with the geometric realisation functor  $|-| : \mathcal{K} \rightarrow \mathcal{W}$  (see, for example, [La79] and [LTW79]);
- A characterisation of a class of weak homotopy inverses [FriLa81] for the nerve functor  $N : \mathcal{E} \rightarrow \mathcal{K}$ .

Because  $\mathcal{E}$  does not have enough strong homotopies (that is, natural transformations), there is no hope of giving a "constructive" definition for the fundamental groupoid  $\Pi(A)$  of  $A$ . In Section 3, if  $A$  is the monoidal path algebra  $P(G)$  of a CFG  $G$ , we show that:

- Homotopy Generation Problem is equivalent to the undecidable Ambiguity Generation Problem;
- Homotopy Extension Problem is equivalent to the undecidable Unification Problem.

However, if the path algebra  $P(G)$  generated from the CFG  $G$  satisfies the sufficient conditions given in [La89a], then the fundamental groupoid  $\Pi(PG)$  is constructible, since it is generated by a finite set of strong homotopies (see Section 3.4 below).

**Organisation of the paper:** Section 2 develops the definitions of various syntactic algebras for CFG's and defines the concepts of the generation of the ambiguity congruence relation for the path algebra and the generation of mgu's for the context-free term algebra. Also in Section 2 sufficient conditions for the existence of ambiguity generators and the existence of unification algorithms for these context-free algebras are presented. In Section 3, the homotopy definitions for the category  $\mathcal{E}$  of small categories are developed, along with the connection between giving generating homotopies for the fundamental groupoid of the path algebra  $P(G)$  of a CFG  $G$  and giving generator pairs for the ambiguity congruence on  $P(G)$ . Also in Section 3, we present sufficient conditions on  $P(G)$  for constructing the (strong) fundamental groupoid of  $P(G)$ .

**Acknowledgments:** This paper is in memory of Evelyn Nelson who showed me that undecidable problems need not be hopeless problems. The author would like to thank John Cherniavsky, Geoffrey Frank, Quan Nguyen, Ron Sigal and Don Stanat for fundamental contributions to the unification algorithm. She also extends her appreciation to Ellis Cooper, Alex Heller, Michael Singer, and Jim Stasheff for many useful conversations about the "homotopy connection".

## 2. Context-free Syntactic Algebras

This section includes:

- (1) A review, similar to the development in [D<sup>2</sup>Qu78], of a context-free grammar (CFG) as a derivation system for expressions, along with an example class  $G_{FP}$  of CFG's for expressions in FFP languages [Ba78];
- (2) A universal algebraic definition of the typed derivation/path algebra of a CFG [Ru87], along with monoidal homomorphisms between the path algebra and the typed syntactic term algebra;
- (3) The equivalence between finite ambiguity presentations [La89a] in the path algebra and unification algorithms in the syntactic term algebra, along with a discussion of (Turing) undecidability [HoU179] of the existence of such presentations and algorithms.

### 2.1 A Traditional Presentation of Context-free Languages

Let  $T$  be a finite set of symbols, and let  $T^m$  denote the set of all finite strings of  $m$  symbols from  $T$ .  $T^0$  represents the singleton set containing only the empty string  $\epsilon$ .



The set  $T^*$  of all strings of symbols over  $T$  is the disjoint union  $\sqcup T^m$ ,  $m \geq 0$ , where ' $\sqcup$ ' denotes the disjoint union. The set  $T^*$  together with the associative operation of concatenation is a free monoid with empty string  $\epsilon$  as the identity.

**Definition 2.1.1:** Let  $T$  be a finite set of symbols. A *language*  $L$  over  $T$  is any subset of  $T^*$ ; that is,  $L \subseteq T^*$ . An element  $e \in L$  is called an *expression* (or a *sentence*) of  $L$ .  $\square$

**Definition 2.1.2:** A *context-free grammar* is a four-tuple  $(T, N, P, \sigma)$  where:

- (a)  $T$  is a finite set of *terminal symbols*;
- (b)  $N$  is a finite set of *nonterminal symbols*;
- (c)  $T$  and  $N$  are disjoint; that is  $T \cap N = \emptyset$ ;
- (d)  $P$  is the finite set of *productions* or syntactic rewrite rules;  $P \subset N \times (T \sqcup N)^*$ ;
- (e)  $\sigma \in N$  is the *starting nonterminal*.  $\square$

**Notation 2.1.3:** If  $p \in P$  and  $p = (A, S(p))$ , then we write  $A \rightarrow_p S(p)$ . The left-hand-side of  $p$  is  $A$  and the right-hand-side is  $S(p)$ . The right-hand-side  $S(p)$  can assume the following "shape":

$$S(p) = s_1 A_1 s_2 \dots s_k A_k s_{k+1}.$$

with  $k \geq 0$ ,  $A_i \in N$ , and  $s_j \in T^*$ .  $\square$   
The process of generating an expression according to a CFG  $G$  is the successive rewriting of *sentential forms* (that is, elements of  $(T \sqcup N)^*$ ) through the use of productions of the grammar, starting with the start nonterminal  $\sigma$ . The sequence of sentential forms and productions required to generate an expression constitutes a *derivation* of the expression according to the grammar  $G$ .

**Definition 2.1.4:** Let  $G = (T, N, P, \sigma)$  be a CFG.

- (a) If  $A \rightarrow_p S(p)$  is a production in  $G$ , and  $w_0 = u A v$  and  $w_1 = u S(p) v$  are sentential forms (with  $u$  and  $v$  possibly empty), we say that  $w_1$  is *immediately derived* from  $w_0$  in  $G$ , and we indicate this relation by writing  $w_0 \xrightarrow{*}_p w_1$ . The collection  $F_1(G)$  of all immediate derivations is called the *free (monoidal) graph* on  $G$ .
- (b) If  $(w_0, w_1, \dots, w_n)$  is a sequence of sentential forms such that

$$w_0 \xrightarrow{*}_{p_1} w_1 \xrightarrow{*}_{p_2} \dots \xrightarrow{*}_{p_n} w_n,$$

we say that  $w_n$  is *derivable* from  $w_0$  and indicate this relation by writing  $w_0 \xrightarrow{*}_d w_n$ , where

$$d = (w_0, p_1, w_1, p_2, \dots, p_n, w_n).$$

The sequence  $d$  is called a *derivation* of  $w_n$  from  $w_0$  according to  $G$ . The beginning sentential form  $w_0$  is called the *domain* of  $d$ , while the last sentential form  $w_n$ , denoted by  $S(d)$ , is called the *codomain* of  $d$ .

- (c) For each sentential form  $w \in (T \cup N)^*$ , let  $w \xrightarrow{*}_w w$  (or more simply  $w \xrightarrow{*} w$ ) denote the *identity derivation* on  $w$ . The *free category*  $F(G)$  of the free monoidal graph,  $F_1(G)$  is the collection all derivations in  $G$ , closed with respect to identity derivations and composition of derivations [Mac71].
- (d) A derivation  $w_0 \xrightarrow{*}_d w_n$  is *leftmost*, whenever, for each  $j$ ,  $1 \leq j \leq n$ , the nonterminal rewritten at the  $j^{\text{th}}$  step is the leftmost nonterminal in the sentential form  $w_{j-1}$ .
- (e) The grammar  $G$  is said to be *ambiguous* if there exist a terminal string  $e \in T^*$  and distinct leftmost derivations  $\sigma \xrightarrow{*}_{d_1} e$  and  $\sigma \xrightarrow{*}_{d_2} e$ . □

**Definition 2.1.5:** Let  $G = (T, N, P, \sigma)$  be a CFG and  $u \xrightarrow{*}_d w$ .

- (a) The set  $|d| = \{u \xrightarrow{*}_d w \xrightarrow{*}_{d_1} S(d_1)\}$  consists of all *derivation extensions* of  $d$ . (Within the comma category of  $F(G)$  [Mac71],  $|d|$  is the set of all morphisms with domain  $d$ .)
- (b)  $|d|^T = \{u \xrightarrow{*}_d w \xrightarrow{*}_{d_1} e \mid e \in T^*\}$  is the set of all *terminal derivation extensions* of  $d$ . Similarly,  $|d|^T = \{u \xrightarrow{*}_d w \xrightarrow{*}_{d_1} S(d_1) \mid d_1 \text{ is leftmost}\}$  is the set of all *leftmost derivation extensions* of  $d$ .
- (c) Let  $|w|$  denote the set of all derivations that begin with  $w \in (T \cup N)^*$ ; that is, the set of all derivation extensions of the identity derivation  $w \xrightarrow{*} w$ . If  $A \in N$ ,  $|A|$  represents the set of all derivations with domain  $A$ .
- (d) If  $A \in N$ , let  $L_G(A)$  denote the *language* of all terminal strings derivable from  $A$ ; that is, the set of all derivation codomains of derivations in  $|A|^T$ .
- (e) The language  $L_G(\sigma)$  is called the *language generated by*  $G$

and is denoted by  $L(G)$ . If  $e \in L(G)$ , we say that  $e$  is a *string*, an *expression* or a *word* generated by  $G$ .  $\square$

An FFP language  $\mathcal{L}$  [Ba78] contains three basic kinds of expressions: atoms, applications, and sequences. *Atomic expressions* include numbers and boolean constants, as well as a collection of "names" for the language primitives. An expression  $(f : e)$  is called an *application*. It represents the application of the function (or program component)  $f$  to the operand (or input) component  $e$ ; both  $f$  and  $e$  can be any expression in  $\mathcal{L}$ . *Sequences* are defined recursively as lists of FFP expressions bracketed by ' $<$ ' and ' $>$ '. No unbracketed list is a well-formed expression in  $\mathcal{L}$ . The following example develops a context-free description of the syntactic structure of a class of FFP languages.

**Example 2.1.6:** The following table describes a class  $G_{FP}$  of CFG's for expressions in FFP languages, by listing the principal productions appearing in any CFG in  $G_{FP}$ . The grammars in this class are partially specified; the remaining productions appearing in any CFG in  $G_{FP}$  which generate atomic expression collections are not given. The set of atoms is the language  $L(AT)$  (generated by  $G_{FP}$ ) and is denoted by  $|AT|$ . (When no confusion arises, we refer to this class of CFG's as "the grammar  $G_{FP}$ ".)

Because the unification algorithm takes as input a pair of terms (in a typed syntactic algebra) that contain variables for expressions, atomic expressions and expression lists, the grammar  $G_{FP}$  contains an arbitrary number of (terminal) expression variables  $V(E) = \{x_i\}$ , atomic variables  $V(AT) = \{a_i\}$  and expression list variables  $V(L) = \{l_j\}$ . A list variable  $l_j$  represents a (possibly empty) sublist of a list. For example, if  $\theta$  is a substitution such that the instantiation of  $l_j$ , denoted by  $\theta l_j$ , is the empty string  $\epsilon$ , then the instantiated sequence expression  $\theta \langle x_i l_j \rangle = \langle \theta x_i \rangle$ ; similarly for  $\langle l_j x_i \rangle$ . If  $\theta l_j = e_1 \dots e_n$  then

$$\theta \langle x_i l_j \rangle = \langle \theta x_i e_1 \dots e_n \rangle.$$

The grammar  $G_{FP}$  is ambiguous, because lists can be constructed by gluing a single component onto the left-end or the right-end of an already constructed list. For example, the distinct leftmost derivations:

$$d_1 = E \xrightarrow{*}_{br} \langle L \rangle \xrightarrow{*}_{gl} \langle E L \rangle \xrightarrow{*}_{gr} \langle E L E \rangle$$

$$d_r = E \xrightarrow{*}_{br} \langle L \rangle \xrightarrow{*}_{gr} \langle L E \rangle \xrightarrow{*}_{gl} \langle E L E \rangle$$

derive the same sentential form,  $S(d_1) = \langle E L E \rangle = S(d_r)$ . In fact,  $|\langle E L E \rangle|^T$  is the set of all derivations to sequence expressions having at least two components.  $\square$

Table 1: A Grammar for FFP Languages

**Terminals:**  $T = |AT| \cup V(E) \cup V(AT) \cup V(L) \cup \{ "<", ">", "(", ")", ":", ";" \}$

**Nonterminals:**  $N = \{ E, AT, L \}$

**Start Nonterminal:**  $E$

**Productions:**

| Production Type                                                                                                                             | Labeled Production                                                                                                            | Production Type                                                                                                                       | Labeled Production                                                                                              |
|---------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>EXPRESSION</b> <ul style="list-style-type: none"> <li>• atomic</li> <li>• application</li> <li>• sequence</li> <li>• variable</li> </ul> | $E \xrightarrow{at} AT$<br>$E \xrightarrow{ap} (E : E)$<br>$E \xrightarrow{br} \langle L \rangle$<br>$E \xrightarrow{ve} x_i$ | <b>LISTS</b> <ul style="list-style-type: none"> <li>• empty</li> <li>• glue left</li> <li>• glue right</li> <li>• variable</li> </ul> | $L \xrightarrow{em} \epsilon$<br>$L \xrightarrow{gl} EL$<br>$L \xrightarrow{gr} LE$<br>$L \xrightarrow{vl} l_j$ |
| <b>ATOMIC</b> <ul style="list-style-type: none"> <li>• atom</li> <li>• variable</li> </ul>                                                  | $AT \xrightarrow{a} a$<br>$AT \xrightarrow{va} a_i$                                                                           |                                                                                                                                       |                                                                                                                 |

## 2.2 Typed Syntactic Algebras

Each derivation  $A \xrightarrow{*}_d S(d)$  in a CFG  $G$  can be viewed both as a path  $d = p(d)$  in a free monoidal universal algebra [Ru87] and as a term  $t(d)$  in the free typed term algebra (once variables have been "derived" for the nonterminals in  $S(d)$ ). Furthermore, each term  $t$  corresponds to a "retracted" derivation  $r(t)$  with each variable replaced by its corresponding nonterminal/type so that the sentential form  $S(r(t))$  is "variable-free."

For the remainder of the paper, we assume that  $G = (T, N, P, \sigma)$  is a CFG and that  $G$  contains terminal variable productions  $A \xrightarrow{v} v(A)_n$  for each nonterminal  $A \in N$ . The variable  $v(A)_n$  is said to have *nonterminal type*  $A$ ; that is, the nonterminal collection  $N$  represents the types. When the context is clear, we use  $v_n$  to denote the  $A$ -variable  $v(A)_n$ .

**Definition 2.2.1:** For each CFG  $G$ , there is a syntactic congruence relation, the *interchange congruence*  $\mathcal{I}(G)$ , for the free derivation category  $F(G)$ . Two interchange congruent derivations differ only in the order that productions are applied.  $\mathcal{I}(G)$  is generated by the interchange derivation pairs; that is, if  $w_j \in (T \cup N)^*$ ,  $j = 1, 2, 3$ ,  $A_i \in N$ ,  $i = 1, 2$ , and  $A_i \rightarrow_{p_i} S(p_i)$ ,  $i = 1, 2$ , then the following derivation pair is a generating pair in  $\mathcal{I}(G)$ :

$$\begin{aligned} (w_1 A_1 w_2 A_2 w_3 \xrightarrow{*}_{p_1} w_1 S(p_1) w_2 A_2 w_3 \xrightarrow{*}_{p_2} w_1 S(p_1) w_2 S(p_2) w_3, \\ w_1 A_1 w_2 A_2 w_3 \xrightarrow{*}_{p_2} w_1 A_1 w_2 S(p_2) w_3 \xrightarrow{*}_{p_1} w_1 S(p_1) w_2 S(p_2) w_3). \end{aligned}$$

Each congruence class has a unique leftmost derivation which corresponds to a unique *parse tree* [HoU179]; hence the quotient category  $F(G)/\mathcal{I}(G)$  is called the "algebra" of parse trees generated by  $G$ . Under the interchange congruence, concatenation of derivations is *monoidal*; that is,  $\mathcal{I}(G)$  is the smallest congruence so that  $F(G)/\mathcal{I}(G)$  is monoidal (see, for example, [Be75] or [Ne80]).  $\square$

**Definition 2.2.2:** The *free path algebra*  $P(G)$  generated by the CFG  $G$  [Ru87] is the monoidal comma category [Mac71] of  $F(G)/\mathcal{I}(G)$ .

- (a) The *carrier sets* (that is, the sets used to define the algebraic operations) correspond to the nonterminal derivation sets (modulo  $\mathcal{I}(G)$ ); that is, if  $A \in N$ , then

$$|A| = \{A \xrightarrow{*}_d S(d)\} \simeq |A|^L.$$

- (b) The *operations* correspond to the productions as follows:  
 (b1) Each terminal production  $A \rightarrow_p e_p$ ,  $e_p \in T^*$  corresponds to a *nullary operation*

$$\hat{p}: \{*\} \rightarrow |A|.$$

- (b2) If  $A \rightarrow_p s_1 A_1 s_2 A_2 \dots A_{k-1} s_k A_k s_{k+1}$ ,  $k \geq 1$ , is a production, then

$$\hat{p} : |A_1| \times |A_2| \times \dots \times |A_k| \rightarrow |A|$$

is the operation, which we call a *p-concatenation*, defined by:

$$\hat{p}(A_1 \xrightarrow{*}_{d_1} S(d_1), \dots, A_k \xrightarrow{*}_{d_k} S(d_k)) =$$

$$\begin{aligned}
 A & \xrightarrow{p} s_1 A_1 s_2 A_2 \dots A_{k-1} s_k A_k s_{k+1} \\
 & \xrightarrow{d_1} s_1 S(d_1) s_2 A_2 \dots A_{k-1} s_k A_k s_{k+1} \\
 & \xrightarrow{d_2} s_1 S(d_1) s_2 S(d_2) \dots A_{k-1} s_k A_k s_{k+1} \\
 & \vdots \\
 & \xrightarrow{d_{k-1}} s_1 S(d_1) s_2 S(d_2) \dots S(d_{k-1}) s_k A_k s_{k+1} \\
 & \xrightarrow{d_k} s_1 S(d_1) s_2 S(d_2) \dots S(d_{k-1}) s_k S(d_k) s_{k+1}.
 \end{aligned}$$

Also we use the following composition / concatenation notation for  $p$ -concatenations:

$$\hat{p}(A_1 \xrightarrow{d_1} S(d_1), \dots, A_k \xrightarrow{d_k} S(d_k)) = (d_1 \dots d_k) \circ p.$$

□

Each path  $d$  in  $\mathbf{P}(G)$  corresponds to a unique leftmost derivation from a nonterminal  $A$  to the sentential form  $S(d)$ .

**Example 2.2.3:** In the free monoidal algebra  $\mathbf{P}(G_{FP})$ , the leftmost E-derivation (a derivation with domain  $E$ ):

$$\begin{aligned}
 d = E & \xrightarrow{br} \langle L \rangle \xrightarrow{gl} \langle EL \rangle \xrightarrow{gr} \langle ELE \rangle \\
 & \xrightarrow{vl} \langle El_j E \rangle \xrightarrow{ve} \langle El_j x_i \rangle
 \end{aligned}$$

corresponds to the *algebraically constructed element*

$$p(d) = \hat{d} = \hat{br} \hat{gl} \hat{l}_E \hat{gr} \hat{vl} \hat{l}_j \hat{ve} x_i.$$

When no confusion arises, we use the derivation format for paths in  $\mathbf{P}(G)$  and suppress the "hat" notion.

The path algebra  $\mathbf{P}(G_{FP})$  has the  $p$ -concatenations that are given below in Table 2. □

Table 2: FFP  $p$ -concatenation

| Production Type                                                                                                                             | Labeled Production                                                                                                                  | Production Type                                                                                                                       | Labeled Production                                                                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>EXPRESSION</b> <ul style="list-style-type: none"> <li>• atomic</li> <li>• application</li> <li>• sequence</li> <li>• variable</li> </ul> | $ A  \rightarrow_{at}  E $<br>$ E  \times  E  \rightarrow_{ap}  E $<br>$ L  \rightarrow_{br}  E $<br>$\{x_i\} \rightarrow_{vs}  E $ | <b>LISTS</b> <ul style="list-style-type: none"> <li>• empty</li> <li>• glue left</li> <li>• glue right</li> <li>• variable</li> </ul> | $\{\epsilon\} \rightarrow_{em}  L $<br>$ E  \times  L  \rightarrow_{gl}  L $<br>$ L  \times  E  \rightarrow_{gr}  L $<br>$\{l_j\} \rightarrow_{vl}  L $ |
| <b>ATOMIC</b> <ul style="list-style-type: none"> <li>• atom</li> <li>• variable</li> </ul>                                                  | $\{a\} \rightarrow_a  A $<br>$\{a_i\} \rightarrow_{va}  A $                                                                         |                                                                                                                                       |                                                                                                                                                         |

**Definition 2.2.4:** The *free term algebra* generated by the CFG  $G$  [Ru87] is the typed syntactic subalgebra  $T(G)$  of  $P(G)$ .

(a) The *carrier sets* correspond to the terminal derivation sets

$$|A|^T = \{ A \xrightarrow{*}_d S(d) \mid S(d) \in T^* \},$$

where  $A \in N$ .

(b) The *operations* correspond to the productions as follows:

(b1) Each non-variable terminal production  $A \rightarrow_p a_p$ ,  $a_p \in T^*$  corresponds to a *nullary operation*

$$\hat{p} : \{*\} \rightarrow |A|.$$

The collection  $AT(G)$  of all non-variable nullary operations is the set of *atoms* of  $T(G)$ .

(b2) Each variable production  $A \rightarrow_v v_i$  corresponds to a nullary operation

$$\hat{v} : \{*\} \rightarrow |A|.$$

The collection  $V(G)$  of all variable nullary operations is the set of *typed variables* of  $T(G)$ .

(b3) If  $A \rightarrow_p s_1 A_1 s_2 A_2 \dots A_{k-1} s_k A_k s_{k+1}$ ,  $k \geq 1$ , is a production, then

$$\hat{p} : |A_1|^T \times |A_2|^T \times \dots \times |A_k|^T \rightarrow |A|^T$$

is called *p-concatenation* and is the restriction of the p-concatenation in  $P(G)$  to  $|A|^T \times \dots \times |A|^T$ .

If  $(t_1, \dots, t_k) \in |A_1|^T \times \dots \times |A_k|^T$ , then

$$(t_1 \dots t_k) \circ p = \hat{p}(t_1, \dots, t_k) \in |A|^T. \quad \square$$

**Definition 2.2.5:**

(a) The *extension homomorphism*  $t : P(G) \rightarrow T(G)$  is defined as follows:

- If  $A \xrightarrow{*}_d S(d)$  is a path in  $|A|$ , then  $t(d)$  is the term in  $|A|^T$

$$t(d) = A \xrightarrow{*}_d S(d) \xrightarrow{*}_v S(t(d)),$$

where the leftmost terminal derivation extension  $S(d) \xrightarrow{*}_v S(t(d))$  is obtained from  $S(d)$  by "replacing" each nonterminal  $A_i$  occurring in  $S(d)$  with the next indexed  $A_i$ -variable  $A_i \xrightarrow{*}_v v_{n+1}$  not occurring in  $S(d)$ .

(b) The *retraction homomorphism*  $r : T(G) \rightarrow P(G)$  is defined as follows:

- If  $t = A \xrightarrow{*}_t S(t)$  is a term in  $|A|^T$ , then  $r(t)$  is the "retract" derivation in  $|A|$

$$A \xrightarrow{*}_{r(t)} S(r(t))$$

obtained from  $t$  by "replacing" each variable  $A_i \xrightarrow{*}_v v_n$  occurring in  $S(t)$  with its nonterminal type  $A_i$  (that is, its identity derivation  $A_i \xrightarrow{*}_{A_i} A_i$ ).  $\square$

**Example 2.2.6:** For example, for the grammar  $G_{FP}$ , the path  $d_1$

$$d_1 = E \xrightarrow{*}_{br} < L > \xrightarrow{*}_{gl} < EL > \xrightarrow{*}_{gr} < ELE >$$

has derivation extension term  $t(d_1)$

$$\begin{aligned} t(d_1) = E \xrightarrow{*}_{br} < L > \xrightarrow{*}_{gl} < EL > \xrightarrow{*}_{gr} < ELE > \\ \xrightarrow{*}_{ve} < x_1 LE > \xrightarrow{*}_{vl} < x_1 l_1 E > \xrightarrow{*}_{ve} < x_1 l_1 x_2 > . \end{aligned}$$



While the term  $t_1 \in T(G_{FP})$

$$\begin{aligned} t_1 = & E \xrightarrow{br} \langle L \rangle \xrightarrow{gl} \langle EL \rangle \xrightarrow{ve} \langle x_1 L \rangle \\ & \xrightarrow{gr} \langle x_1 LE \rangle \xrightarrow{vl} \langle x_1 l_1 E \rangle \xrightarrow{ve} \langle x_1 l_1 x_2 \rangle \end{aligned}$$

has retract  $r(t_1) = d_1$  . □

The retract homomorphism  $r : T(G) \rightarrow P(G)$  is used to make precise the concept of substitution in the term algebra  $T(G)$  .

**Definition 2.2.7:** A *substitution* in the free term algebra  $T(G)$  is a set  $\theta$  of variable-terms pairs:

$$\theta = \{(v_n \Rightarrow t_n)\}$$

where, for each variable  $v_n$  of type  $A_i$  ,  $A_i \xrightarrow{\lambda_n} S(t_n)$  is an  $A_i$ -term. For any A-term  $t = A \xrightarrow{\lambda} S(t)$  in  $T(G)$  and substitution  $\theta$  , the *instantiated term*  $\theta t$  is the derivation extension

$$\theta t = A \xrightarrow{r(t)} S(r(t)) \xrightarrow{d(\theta)} S(\theta t),$$

where  $d(\theta)$  is the  $r(t)$ -concatenation in  $|S(r(t))|^T$  formed using the derivation collection  $\{A_i \xrightarrow{\lambda_n} S(t_n)\}$  . □

**Example 2.2.8:** For example, in  $T(G_{FP})$  , if the term  $t_1$

$$\begin{aligned} t_1 = & E \xrightarrow{br} \langle L \rangle \xrightarrow{gl} \langle EL \rangle \xrightarrow{ve} \langle x_1 L \rangle \\ & \xrightarrow{gr} \langle x_1 LE \rangle \xrightarrow{vl} \langle x_1 l_1 E \rangle \xrightarrow{ve} \langle x_1 l_1 x_2 \rangle \end{aligned}$$

and if  $\theta$  is the substitution

$$\begin{aligned} \theta = & \{(x_1 \Rightarrow (E \xrightarrow{d_1} \text{true})), (l_1 \Rightarrow (L \xrightarrow{d_2} x_3 \langle \rangle)) , \\ & (x_2 \Rightarrow (E \xrightarrow{d_3} \langle \text{false } l_2 \rangle))\} , \end{aligned}$$

then the instantiated E-term  $\theta t_1$  is the derivation concatenation  $(d_1 d_2 d_3) \circ r(t_1)$  :

$$\begin{aligned} \theta t_1 = & E \xrightarrow{r(t_1)} \langle ELE \rangle \\ & \xrightarrow{d_1} \langle \text{true } LE \rangle \end{aligned}$$

$$\begin{aligned} * \rightarrow_{d_2} < \text{true } x_3 < > E > \\ * \rightarrow_{d_2} < \text{true } x_3 < > < \text{false } l_2 >> . \quad \square \end{aligned}$$

**Definition 2.2.9:** *Composition of substitutions* [Si89] corresponds to composition of paths in  $P(G)$ , using the retraction homomorphism  $r : T(G) \rightarrow P(G)$ . We use the notation  $\theta_2 \circ \theta_1$  to represent the composition of substitutions. If  $A * \rightarrow_{d_1} S(t)$  is an A-term in  $T(G)$ , then the instantiated term  $(\theta_2 \circ \theta_1)t$  is the derivation composition

$$\begin{aligned} A * \rightarrow_{r(t)} S(r(t)) * \rightarrow_{r(d(\theta_1))} S(r(\theta_1 t)) \\ * \rightarrow_{d(\theta_2)} S((\theta_2 \circ \theta_1)t) . \end{aligned}$$

Composition of substitutions is associative.  $\square$

### 2.3 Unification and Ambiguity in Syntactic Algebras

Recall that a CFG  $G$  is ambiguous if there exist a terminal string  $e \in T^*$  and distinct leftmost derivations  $\sigma * \rightarrow_{d_1} e$  and  $\sigma * \rightarrow_{d_2} e$ . It is possible for a  $G$  to have symbols and productions that cannot occur in any derivation  $\sigma * \rightarrow_d e$  with domain, the start nonterminal  $\sigma$ . There are effective algorithms for deleting useless symbols and productions from  $G$  without affecting the language  $L(G)$  it generates. The "effectiveness proofs" presented in, for example, [HoU179] or [LePa81], produce algebraically equivalent expression sets. We assume that the CFG's under consideration in the remainder of the paper are "useful" (that is, have no useless symbols or productions) and generate a nontrivial language (that is,  $L(G) \neq \{\epsilon\}$ ). In this case, the definition of ambiguity is more general:

**Proposition 2.3.1:** *The useful CFG  $G$  is ambiguous if and only if there exists a nonterminal  $A$  and there exist distinct left-most  $A$ -derivations  $A * \rightarrow_{d_1} w$  and  $A * \rightarrow_{d_2} w$  in  $P(G)$ .*  $\square$

#### Definition 2.3.2:

- (a) For each CFG  $G$ , there is a syntactic congruence relation, the *ambiguity congruence*  $\mathcal{A}(G)$ , for the free derivation category  $F(G)$ . Two derivations  $A_1 * \rightarrow_{d_1} S(d_1)$  and  $A_2 * \rightarrow_{d_2} S(d_2)$  are ambiguity congruent, denoted  $d_1 \equiv d_2$ , if and only if  $A_1 = A_2$  and  $S(d_1) = S(d_2)$ .

This ambiguity relation  $\mathcal{A}(G)$  :

- is the pre-order congruence [Mac71] on  $F(G)$  ;
  - includes the interchange congruence  $\mathcal{I}(G)$  ;
  - identifies any two derivations having the same domains and codomains.
- (b) The quotient algebra  $P(G) / \mathcal{A}(G)$  is the typed monoidal algebra  $S(G)$  of *sentential forms* generated by  $G$  . Each ambiguity congruence class  $[A * \rightarrow_d S(d)]$  in  $S(G)$  can be viewed as a sentential form  $w = S(d)$  of type  $A$  , since the actual derivation  $d$  is not "remembered" in  $S(G)$  .
- (c) The quotient algebra  $T(G) / \mathcal{A}(G)$  is the typed monoidal algebra  $E(G)$  of *expressions (terms)* generated by  $G$  .
- (d) The ambiguity congruence  $\mathcal{A}(G)$  is said to be *(finitely) generated* by a (finite) set  $A$  ,  $A \subseteq \mathcal{A}(G)$  , if  $\mathcal{A}(G) = [A]$  , where  $[A]$  is the smallest monoidal congruence containing the set  $A$  of derivation pairs.  $\square$

For a nontrivial useful CFG  $G$  , a second characterisation of ambiguity is:

**Proposition 2.3.3:** *A CFG  $G$  is ambiguous if and only if either of the following equivalent conditions hold:*

- (1) *The interchange congruence  $\mathcal{I}(G)$  is a proper subcongruence of the ambiguity congruence  $\mathcal{A}(G)$  .*
- (2) *There is a nonempty set  $A$  of pairs of distinct leftmost derivations that generates the ambiguity congruence  $\mathcal{A}(G)$  .*  $\square$

In order to demonstrate that the Unification and Homotopy Extension Problems are equivalent to the undecidable Ambiguity and Intersection Problems, we present the concept of unification in  $T(G)$  and then show that giving a finite unification algorithm for  $T(G)$  is equivalent to giving a finite set  $A$  of generators for the ambiguity congruence  $\mathcal{A}(G)$  .

Recall that  $G$  is a CFG which is useful, nontrivial and has typed variable productions  $\{A \rightarrow_v v_n\}$  for each nonterminal type  $A$  .

**Definition 2.3.4:**

- (a) Let  $t_1$  and  $t_2$  be  $A$ -terms in  $T(G)$  . A substitution pair  $(\theta_1, \theta_2)$  is a *unifier* for the  $(t_1, t_2)$  if and only if  $(\theta_1 t_1, \theta_2 t_2)$  is an ambiguity derivation pair in  $\mathcal{A}(G)$  ; that is, if and only if the derivation pair

$$(A * \rightarrow_{r(t_1)} S(r(t_1)) * \rightarrow_{d(\theta_1)} S(\theta_1 t_1) ,$$

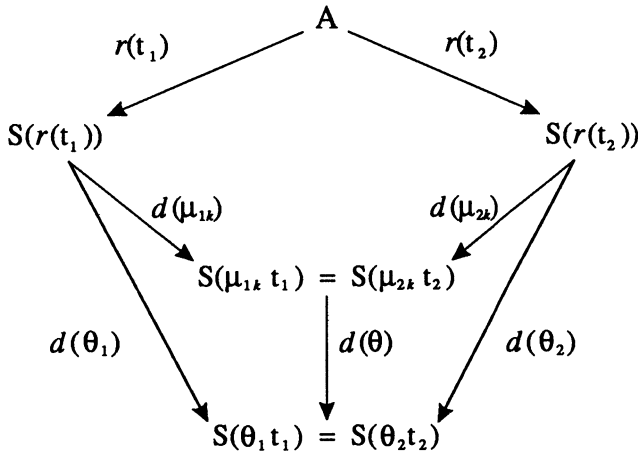
$$A \xrightarrow{r(t_1)} S(r(t_1)) \xrightarrow{d(\theta_1)} S(\theta_1 t_1) \in \mathcal{A}(G) ,$$

with  $S(\theta_1 t_1) = S(\theta_2 t_2)$ . The term pair  $(t_1, t_2)$  is said to be *unifiable* if and only if the set  $u(t_1, t_2)$  of all  $(t_1, t_2)$ -unifiers is nonempty.

- (b) Let  $d_1$  and  $d_2$  be A-paths in  $P(G)$ . The pair  $(A \xrightarrow{d_1} S(d_1), A \xrightarrow{d_2} S(d_2))$  is said to be *unifiable* if and only if the term pair  $(t(d_1), t(d_2))$  is unifiable in  $T(G)$ , where  $t : P(G) \rightarrow T(G)$  is the extension homomorphism (given in Definition 2.2.5(a)).
- (c) A substitution  $\theta = \{(v_n \Rightarrow (A_i \xrightarrow{r(t_n)} S(t_n)))\}$  is said to be *simple* whenever, for each term  $t_n$ , the retracted derivation  $A_i \xrightarrow{r(t_n)} S(t_n)$  has each  $r(t_n)$ -concatenation component derivation either a production  $A_k \rightarrow_{p_k} S(p_k)$  or an identity  $A_k \rightarrow A_k$ .
- (d) A unifier  $(\theta_1, \theta_2) \in u(t_1, t_2)$  is said to be *simple* if the substitutions  $\theta_1$  and  $\theta_2$  are simple.
- (e) A set  $mg_u(t_1, t_2) = \{(\mu_{1k}, \mu_{2k})\}$  of unifiers for the pair  $(t_1, t_2)$  is said to be the *most general* (or *minimally complete* [Si89]) if and only if, for any unifier  $(\theta_1, \theta_2) \in u(t_1, t_2)$ , there is a *unique* most general unifier  $(\mu_{1k}, \mu_{2k})$  and a (not necessarily unique) substitution  $\theta$ , such that

$$(\theta \circ \mu_1)t_1 \equiv \theta_1 t_1 \equiv \theta_2 t_2 \equiv (\theta \circ \mu_2)t_2 ;$$

that is, the following graph exists in  $T(G)$  :



For each most general unifier  $(\mu_{1k}, \mu_{2k}) \in mgu(t_1, t_2)$ , we use the notation  $mgu_k$  to represent the *most general unified expression*

$$mgu_k = S(\mu_1 \ t_{1k}) = S(\mu_{2k} \ t_2)$$

of type A in  $E(G)$ .  $\square$

**Example 2.2.5:** For the term algebra  $T(G_{FP})$ , consider the sequence E-term pair  $(t_L, t_R)$ , where

$$\begin{aligned} t_L &= E \xrightarrow{d_L} \langle EL \rangle \xrightarrow{*} \langle x_1 l_1 \rangle \\ t_R &= E \xrightarrow{d_R} \langle LE \rangle \xrightarrow{*} \langle l_2 x_2 \rangle. \end{aligned}$$

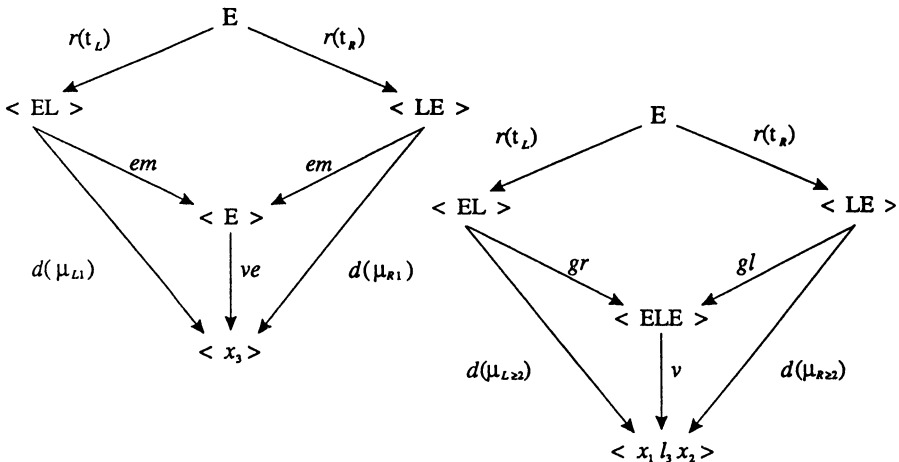
The set  $mgu(t_L, t_R)$  consists of the following two *simple* unifiers:

$$mgu(t_L, t_R) = \{(\mu_{L1}, \mu_{R1}), (\mu_L \geq 2, \mu_R \geq 2)\}$$

where

- $\mu_{L1} = \{ (x_1 \Rightarrow (E \xrightarrow{*_{ve}} x_3)), (l_1 \Rightarrow (L \xrightarrow{*_{em}} \epsilon)) \}$
- $\mu_{R1} = \{ (l_2 \Rightarrow (L \xrightarrow{*_{em}} \epsilon)), (x_2 \Rightarrow (E \xrightarrow{*_{ve}} x_3)) \}$
- $\mu_L \geq 2 = \{ (x_1 \Rightarrow (E \xrightarrow{*_{ve}} x_1)), (l_1 \Rightarrow (L \xrightarrow{*_{d_1}} l_3 \ x_2)) \}$
- $\mu_R \geq 2 = \{ (l_2 \Rightarrow (L \xrightarrow{*_{d_r}} x_1 \ l_3)), (x_2 \Rightarrow (E \xrightarrow{*_{ve}} x_2)) \}$ .

The most general unified terms are represented by the following graphs in  $T(G_{FP})$ :



The most general unified expressions are  $mg u_1 = \langle x_3 \rangle$  and  $mg u_{\geq 2} = \langle x_1 l_3 x_2 \rangle$ .

That

$$mg u(t_L, t_R) = \{(\mu_{L1}, \mu_{R1}), (\mu_L \geq 2, \mu_R \geq 2)\}$$

follows from the observation that, if  $(\theta_L, \theta_R) \in u(t_L, t_R)$  (that is,  $(\theta_L t_L, \theta_R t_R)$  is an ambiguity pair in  $\mathcal{A}(G_{FP})$ ), then the sequential expression  $S(\theta_L t_L) = S(\theta_R t_R)$  must have either one (top-level) component, or two or more components.  $\square$

**Remark 2.3.6:** If term  $t_1$  is the A-variable  $A \xrightarrow{*}_v v_n$  and this variable occurs in non-variable A-terms  $t_2$ , then, for any substitution  $\theta = \{(v_n \Rightarrow (A \xrightarrow{*}_{t_3}))\}$ ,

$$\theta t_1 = \theta v_n = t_3 \neq \theta t_2 ;$$

hence,  $u(t_1, t_2) = u(v_n, t_2) = \emptyset$ .

There exist algorithms, called *occurrence checkers* (see, for example, [Si89] or [Kn89]), that determine whether the variable  $A \xrightarrow{*}_v v_n$  occurs in  $t_2$ . For PROLOG where unification and resolution form the basis of program evaluation [ShSt87], such occurrence checkers are not necessarily effective. Because the variables occur as part of the grammar, the parsing algorithm can be used as an effective occurrence checking algorithm for  $T(G)$ .  $\square$

**Remark 2.3.7:** For a unifier  $(\theta_1, \theta_2) \in u(t_1, t_2)$ , it is possible for the substitution components

$$(v_n \Rightarrow (A_i \xrightarrow{*}_{t_{1n}} S(t_{1n})) \in \theta_1 \quad (v_n \Rightarrow (A_i \xrightarrow{*}_{t_{2n}} S(t_{2n})) \in \theta_2$$

for the same variable  $A_i \xrightarrow{*}_v v_n$ , to have  $S(t_{1n}) \neq S(t_{2n})$ ; that is the derivation pair

$$(A_i \xrightarrow{*}_{t_{1n}} S(t_{1n}), A_i \xrightarrow{*}_{t_{2n}} S(t_{2n}))$$

may not be an ambiguity pair in  $\mathcal{A}(G)$ . Consider the example CFG  $G_1$  given in Table 3:

Table 3: An Example CFG  $G_1$ 

**Terminals:**  $T_1 = \{a, b, c, \} \cup V(S) \cup V(A) \cup V(B)$

**Nonterminals:**  $N_1 = \{S, A, B\}$

**Start Nonterminal:**  $S$

**Productions:**

| Production Type | Labeled Production       | Production Type | Labeled Production       |
|-----------------|--------------------------|-----------------|--------------------------|
| • concatenation | $S \rightarrow_{ca} AB$  | • variable      | $S \rightarrow_{wa} x_i$ |
| • atom          | $A \rightarrow_a a$      | • atom          | $B \rightarrow_c c$      |
| • atom          | $A \rightarrow_{ab} ab$  | • atom          | $B \rightarrow_{bc} bc$  |
| • variable      | $A \rightarrow_{wa} a_i$ | • variable      | $B \rightarrow_{wb} b_i$ |

If  $t_1$  is the term  $S \xrightarrow{*}_{t_1} a_1 b_1$  and  $t_2$  is  $S \xrightarrow{*}_{t_2} a_1 b_1$ , with  $a_1, b_1$  and  $b_2$  variables, then

$$mgu(t_1, t_2) = \{(\mu_{11}, \mu_{21})\},$$

where the simple substitution  $\mu_{11}$  is

$$\mu_{11} = \{(a_1 \Rightarrow (A \xrightarrow{*}_a a)), (b_1 \Rightarrow (B \xrightarrow{*}_{bc} bc))\}$$

and the simple substitution  $\mu_{21}$  is

$$\mu_{21} = \{(a_1 \Rightarrow (A \xrightarrow{*}_{ab} ab)), (b_2 \Rightarrow (B \xrightarrow{*}_c c))\}$$

with unified expression  $mgu_1 = abc$ . Since  $a \neq ab$ , the term pair

$$(A \xrightarrow{*}_a a, A \xrightarrow{*}_{ab} ab) \notin \mathcal{A}(G_1);$$

in fact,  $mgu(A \xrightarrow{*}_a a, A \xrightarrow{*}_{ab} ab) = \emptyset$ . □

**Definition 2.3.8:**

(a) The path algebra  $P(G)$  is said to be  $\mathcal{A}(G)$ -free [Si89] if and only if, for each production

$$A \rightarrow_p s_1 A_1 s_2 A_2 \dots A_{k-1} s_k A_k s_{k+1}, \quad k \geq 1,$$

then the  $p$ -concatenation

$$\hat{p} : |A_1| \times |A_2| \times \dots \times |A_k| \rightarrow |A|$$

is a one-to-one function in  $P(G)/\mathcal{A}(G)$ .

- (b) For production  $A \rightarrow_p s_1 A_1 s_2 A_2 \dots A_{k-1} s_k A_k s_{k+1}$ ,  $k \geq 1$ , the  $p$ -concatenation

$$\hat{p} : |A_1| \times |A_2| \times \dots \times |A_k| \rightarrow |A|$$

commutes with unification in  $P(G)/\mathcal{A}(G)$  if and only if

$$\begin{aligned} & \text{mgu}(\hat{p}(t_{11}, \dots, t_{1k}), \hat{p}(t_{21}, \dots, t_{2k})) \\ & \equiv \hat{p}(\text{mgu}(t_{11}, t_{21}), \dots, \text{mgu}(t_{1k}, t_{2k})) \end{aligned}$$

where the right hand side  $\hat{p}$  is the monoidal algebraic extension of  $\hat{p}$  in  $T(G)$  to the algebra  $\Omega(G)$  of *substitution pairs*, and where  $\equiv$  denotes the extension of the ambiguity congruence to the substitution algebra  $\Omega(G)$  [Si89].  $\square$

In [La89a] it is shown that the FFP path algebra  $P(G_{\text{FP}})$  is  $\mathcal{A}(G_{\text{FP}})$ -free. Note that for the example CFG  $G_1$  above,  $P(G_1)$  is not  $\mathcal{A}(G_1)$ -free, since  $\hat{ct} : |A| \times |B| \rightarrow |S|$  is not a one-to-one function in  $P(G_1)/\mathcal{A}(G_1)$ . Also, in [La89a] it is shown that the two conditions given in Definition 2.3.8 are equivalent; that is, if the path algebra  $P(G)$  is  $\mathcal{A}(G)$ -free, then the "structural reduction" can be used to compute mgu sets:

**Theorem 2.3.9:** *For a nontrivial useful CFC  $G$  with typed variables, the following conditions are equivalent:*

- (A) *The path algebra  $P(G)$  is  $\mathcal{A}(G)$ -free;*  
 (B) *For each production*

$$A \rightarrow_p s_1 A_1 s_2 A_2 \dots A_{k-1} s_k A_k s_{k+1},$$

$k \geq 1$ , the  $p$ -concatenation

$$\hat{p} : |A_1| \times |A_2| \times \dots \times |A_k| \rightarrow |A|$$

commutes with unification in  $P(G)/\mathcal{A}(G)$ .  $\square$

Unfortunately, deciding whether the path algebra  $P(G)$  is  $\mathcal{A}(G)$ -free is equivalent to the Ambiguity Problem. In [La89a] some "punctuation properties" of a CFG are described. These properties are sufficient to guarantee the  $\mathcal{A}(G)$ -freeness of  $P(G)$ . Also in [La89a] a second condition is presented for the mgu set  $\text{mgu}(t(A \rightarrow_{p_1} S(p_1)), t(A \rightarrow_{p_2} S(p_2)))$  of produc-



tion term pairs  $(t(A \rightarrow_{p_1} S(p_1)), t(A \rightarrow_{p_2} S(p_2)))$  so that, when  $P(G)$  is  $\mathcal{A}(G)$ -free and this second condition holds, both a unification algorithm exists for  $T(G)$  and  $\mathcal{A}(G)$  is finitely generated:

**Theorem 2.3.10:** *Suppose  $G$  is a useful nontrivial CFG with typed variables. If:*

- (1) *The path algebra  $P(G)$  is  $\mathcal{A}(G)$ -free;*
- (2) *For each A-production pair  $(A \rightarrow_{p_1} S(p_1), A \rightarrow_{p_2} S(p_2))$ , there exists a finite set*

$$mgu(t(A \rightarrow_{p_1} S(p_1)), t(A \rightarrow_{p_2} S(p_2)))$$

*of simple most general unifiers,*  
then

- (i) *There exists a finite unification algorithm for  $T(G)$  (that is, for any A-term pair  $(t_1, t_2)$ , there exists a finite set  $mgu(t_1, t_2)$ , of mgu's);*
- (ii) *The ambiguity congruence  $\mathcal{A}(G)$  is "finitely generated" by production mgu sets (that is, if  $(\mu_{1k}, \mu_{2k}) \in mgu(t(p_1), t(p_2))$ , then the retract derivation pair  $(r(\mu_{1k}t(p_1)), r(\mu_{2k}t(p_2)))$  is an ambiguity generating pair).  $\square$*

It is undecidable whether Condition (2) of Theorem 2.3.10 holds (that is, whether there exists a finite set of mgu's for each production pair). This problem is equivalent to the Intersection Problem, given in terms of the union grammar. In [La89a] the CFG  $G_{FP}$  is shown to satisfy Condition 2 of Theorem 2.3.10. The only nontrivial production pair most general union set is

$$mgu(t(L \rightarrow_{g_l} EL), t(E \rightarrow_{g_r} LE))$$

(see Example 2.3.5).

**Example 2.3.11:** The Example CFG  $G_2$ , given below in Table 4, is due to Q. Nguyen. Because  $G_2$  is regular, it satisfies Condition 1 of Theorem 2.3.10 (that is, the path algebra  $P(G_2)$  is  $\mathcal{A}(G_2)$ -free). However,  $G_2$  does not satisfy Condition 2 (that is, each production pair mgu set is a set of *simple* substitution pairs). We show that no finite unification algorithm exists for  $T(G_2)$ .

**Table 4 : An Example CFG  $G_2$**

**Terminals:**  $T_2 = \{a, b, c, \} \cup V(A)$

**Nonterminals:**  $N_1 = \{ A \}$

**Productions:**

| Production Type   | Labeled Production           |
|-------------------|------------------------------|
| • atom            | $A \rightarrow_{p_0} c$      |
| • left production | $A \rightarrow_{p_1} ababaA$ |
| • left production | $A \rightarrow_{p_2} abA$    |
| • left production | $A \rightarrow_{p_3} babaA$  |
| • variable        | $A \rightarrow_v v_i$        |

A straightforward verification shows that

$$mgu(t(A \rightarrow_{p_i} S(p_i)), A \rightarrow_{p_0} c) = \emptyset ,$$

for  $1 \leq i \leq 3$ , since by left cancellation,  $|c| \cdot |T| \cap |S(p_i)| \cdot |T| = \emptyset$ . Similarly, direct verification of the definition shows

$$mgu(t(p_1), t(p_2)) = \{(\mu_{11}, \mu_{21})\} ,$$

where the simple substitution  $\mu_{21}$  is

$$\mu_{11} = \{(v_1 \Rightarrow (A \xrightarrow{p_3} (ba)^2 A \xrightarrow{v} (ba)^2 v_3))\}$$

and the substitution  $\mu_{21}$

$$\mu_{21} = \{(v_2 \Rightarrow (A \xrightarrow{p_2} ab A \xrightarrow{p_1} (ab)^3 a A \xrightarrow{v} (ab)^3 a v_3))\}$$

with unified expression

$$mgu_1 = a b a b a b a b a v_3 = (ab)^4 a v_3 .$$

Note that the retract  $r(d(\mu_{12}))$

$$r(d(\mu_{12})) = A \xrightarrow{p_2} ab A \xrightarrow{p_1} (ab)^3 a A$$

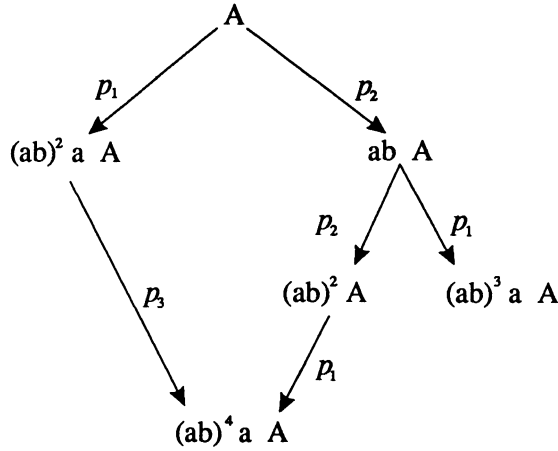
is not simple; so that Condition 2 of Theorem 2.3.10 is not satisfied.

Consider the term pair  $(t(p_1), t_2)$ , where

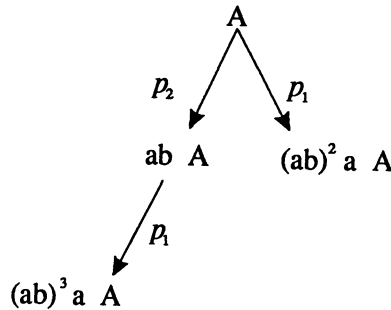
$$t(p_1) = A \xrightarrow{p_1} (ab)^2 a A \xrightarrow{v_1} (ab)^2 a v_1$$

$$t_2 = A \xrightarrow{p_2} ab A \xrightarrow{p_1} (ab)^3 a A \xrightarrow{v_2} (ab)^3 a v_2 .$$

Then, using that the path algebra  $P(G_2)$  is  $\mathcal{A}(G_2)$ -free and using the generating set  $mgu(t(p_1), t(p_2))$  of mgu's, the following retract diagram in  $P(G_2)$  is produced:



This diagram is "recursive", in that using structural reduction, there exists the "left-cancelled" subdiagram



which is the symmetric transpose of the retract pair  $(r(t(p_1)), r(t_2))$ . Using the unification process, the extending derivations in  $mgu(t(p_1), t_2)$  cannot be finite. Thus, these unifiers are not well-formed terms in  $T(G_2)$ .  $\square$

### 3. A Homotopy Theory for Syntactic Algebras

The concepts presented in the first two subsections are a review of some of those presented in [GaZi67], [La73], [La75], [La79], [LTW79], [LaFr81], and [Rot88]. This section includes:

- (1) A definition of the adjoint functor pair  $(cN)$  of categorical realization  $c : \mathcal{K} \rightarrow \mathcal{C}$  and nerve  $N : \mathcal{C} \rightarrow \mathcal{K}$  between the category  $\mathcal{C}$  of small categories and the category  $\mathcal{K}$  of simplicial sets;
- (2) A comparison between strong and weak homotopy for  $\mathcal{C}$ , including the weak homotopy equivalence of  $\mathcal{C}$  and  $\mathcal{K}$ , and including the definition of the strong and weak fundamental groupoid of a small category;
- (3) A discussion of how the fundamental groupoid models the ambiguity congruence  $\mathcal{A}(G)$  in the path algebra  $P(G)$ , and a discussion of how the strong homotopy extension property for the path algebra  $P(G)$  models unification in  $T(G)$ ;
- (4) A presentation of sufficient conditions for the existence of strong generating homotopies for the fundamental groupoid of the path algebra  $P(G)$ .

#### 3.1 Nerve and Categorical Realization

The following presentation is similar to that in [Mac71] and incorporates some of the formal homotopy theory presented in [GaZi67].

##### Notation 3.1.1:

- (a) Let  $\mathcal{C}$  denote the 2-category of small categories.
  - A small category  $A$  is a category with a *set* of objects and having, for each pair of objects  $p$  and  $q$ , the morphism collection  $A(p,q)$  a *set*.
  - $F : A \rightarrow B$  denotes a functor between the small categories  $A$  and  $B$ .
  - $\eta : F \xrightarrow{*} G : A \rightarrow B$  (or simply  $\eta : F \xrightarrow{*} G$ ) denotes a natural transformation from  $F$  to  $G$ ; that is, for each object  $P \in A$ , there is a morphism  $\eta_p : F(p) \rightarrow G(p)$  in  $B$ , such that, if  $a : p \rightarrow q$  in  $A$ , then

$$\eta_q \circ F(a) = G(a) \circ \eta_p .$$

- (b) Let  $\Delta$  denote the small category of finite ordinals  $[n] = \{0 < 1 < \dots < n\}$ ,  $n \geq 0$ , and order preserving maps  $f : [m] \rightarrow [n]$ .
  - $\iota : \Delta \rightarrow \mathcal{C}$  is the inclusion functor, where each ordi-

- nal poset is considered as a small category.
- $\mathfrak{I}([0])$  is the one point category  $0$ , having only one object "0" and an identity morphism  $1_0 : 0 \rightarrow 0$ .
  - $\mathfrak{I}([1])$  is the *categorical unit interval*  $1$ , with one nonidentity morphism  $0 \rightarrow 1$ .
  - The subcategory  $\{0,1\}$  of  $1$  consisting of the two end-points of the unit interval is called the *boundary* of  $1$  and is denoted by  $\partial 1$ .
- (c) For (not necessarily small) categories  $\mathcal{A}$  and  $\mathcal{B}$ , let  $[\mathcal{A}, \mathcal{B}]$  denote the functor category having as objects, functors  $F : \mathcal{A} \rightarrow \mathcal{B}$ , and as morphisms, natural transformations  $\eta : F \xrightarrow{*} G : \mathcal{A} \rightarrow \mathcal{B}$ .
- If  $\mathcal{S}$  denotes the category of sets, then the category  $\mathcal{K}$  of *simplicial sets* is the functor category  $[\Delta^{\text{op}}, \mathcal{S}]$ , where  $\Delta^{\text{op}}$  is the contravariant category of  $\Delta$ . The hom-functor  $\Delta(\_, [k]) = \Delta[k]$  is called the *standard k-dimensional simplicial set*.
  - If  $\mathcal{T}$  is the category of topological spaces, let  $\tau : \Delta \rightarrow \mathcal{T}$  denote the *standard simplex functor* and  $\Delta^k$  denote the standard  $k$ -dimensional simplex. The category  $\mathcal{W}$  of all *CW-complexes* is the functor category  $[(\tau(\Delta))^{\text{op}}, \mathcal{T}]$ .
- (d) Let  $| \_ | : \mathcal{K} \rightarrow \mathcal{W}$  denote the *geometric realization functor* (see [Mi57] or [GaZo67]), induced by mapping the standard  $k$ -dimensional simplicial set  $\Delta[k]$  to the  $k$ -dimensional simplex  $\Delta^k$ .
- (e) The *nerve functor*  $N : \mathcal{C} \rightarrow \mathcal{K}$  is the functor defined by, for each small category  $A$ ,

$$NA = [\mathfrak{I}(\_), A] : \Delta^{\text{op}} \rightarrow \mathcal{S}.$$

Thus  $NA$  is the simplicial set whose  $k$ -simplices are diagrams of the form

$$p_0 \rightarrow_{a_1} p_1 \rightarrow_{a_2} p_2 \rightarrow \cdots \rightarrow_{a_{k-1}} p_{k-1} \rightarrow_{a_k} p_k$$

(that is, a path in  $A$  of length  $k$ ) . □

Since  $\mathfrak{I} : \Delta \rightarrow \mathcal{C}$  is full and faithful,

$$N(\mathfrak{I}[k]) = [\mathfrak{I}(\_), \mathfrak{I}[k]] \simeq \Delta(\_, [k]) = \Delta[k].$$

In particular,  $N(\mathfrak{I}[0]) = N(0) = \Delta[0]$ , the *simplicial point* and  $N(\mathfrak{I}[1]) = N(1) = \Delta[1]$ , the *simplicial unit interval*.

The left adjoint of nerve  $N : \mathcal{C} \rightarrow \mathcal{K}$  is *categorical*

*realization*  $c : \mathcal{K} \rightarrow \mathcal{C}$  [GaZi67;II.4]. Although  $cN \simeq \text{Id}_{\mathcal{C}}$ , categorical realization "forgets" the higher dimensional structure of a simplicial set  $X$  and only uses the 2-dimensional skeletal structure  $Sk_2(X)$  in its definition.

Since  $N$  is a right adjoint, it preserves all limits; and in particular,  $N$  preserves all products; that is

$$N(A \times B) \simeq NA \times NB .$$

### 3.2 Strong and Weak Homotopy

In the category  $\mathcal{C}$  of small categories, the concept of a strong homotopy between functors corresponds to that of a natural transformation between functors. There exists no nice correspondence in  $\mathcal{C}$  for a weak homotopy between functors, because weak homotopy models homotopy in  $\mathcal{W}$  and because there exist more homotopies in  $\mathcal{W}$  than natural transformations in  $\mathcal{C}$  (that is, two functors  $F : A \rightarrow B$  and  $G : A \rightarrow B$  can be weakly homotopic, but there exist no natural transformation  $\eta : F \xrightarrow{*} G$ ). We show that for certain syntactic path algebras  $P(G)$  (considered as small categories), there exist enough strong homotopies. In [La77], [La79], [LTW79] and [FriLa81] it is shown that the nerve functor  $N : \mathcal{C} \rightarrow \mathcal{K}$  induces only a weak homotopy equivalence between  $\mathcal{C}$  and  $\mathcal{K}$ .

**Definition 3.2.1:** Suppose that  $F : A \rightarrow B$  and  $G : A \rightarrow B$  are functors in  $\mathcal{C}$ ; that is,  $F, G \in \mathcal{C}(A, B)$ .

- (a)  $F$  is said to be *strongly homotopic* to  $G$  if and only if any of the following equivalent conditions hold:
- (i) There exists a natural transformation

$$\eta : F \xrightarrow{*} G : A \longrightarrow B .$$

- (ii) There exists a functor  $H$  in  $\mathcal{C}$

$$H : A \times 1 \rightarrow B$$

such that the restriction  $H|_{A \times \{0\}} = F$  and the restriction  $H|_{A \times \{1\}} = G$ ;

- (iii) There exists a simplicial map  $h = NH$

$$h : NA \times \Delta[1] \rightarrow NB$$

such that the restriction  $h|_{NA \times \Delta[0]_0} = NF$  and the restrictions  $h|_{NA \times \Delta[0]_1} = NG$ .

Let  $\sim_s$  denote the *strong homotopy congruence relation* on  $\mathcal{E}$ . (That (ii) is equivalent to (iii) follows from the fact that nerve  $N : \mathcal{E} \rightarrow \mathcal{K}$  is full and faithful.)

- (b)  $F$  is said to be *weakly homotopic* to  $G$  if and only if  $|NF| : |NA| \rightarrow |NB|$  is homotopic to  $|NF| : |NA| \rightarrow |NB|$  in  $\mathcal{W}$ ; that is, there exists a map  $h$  in  $\mathcal{W}$

$$h : |NA| \times \Delta^1 \longrightarrow |NB|$$

such that the restriction  $h|_{|NA| \times \Delta^1(0)} = |NF|$  and the restriction  $h|_{|NA| \times \Delta^1(1)} = |NG|$ . Let  $\sim_w$  denote the *weak homotopy congruence relation* on  $\mathcal{E}$ .  $\square$

### 3.3 Strong Homotopy Extension Property and the Fundamental Groupoid for Syntactic Algebras.

For a CFG  $G$ , the strong fundamental groupoid  $\Pi(P(G))$  of the monoidal path algebra  $P(G)$  is modelled as the collection of "topological loops" (that is, ambiguity pairs in  $\mathcal{A}(G)$ ). For each nonterminal type  $A$ , the "based" fundamental groupoid  $\Pi(P(G,A))$  is the collection of ambiguity pairs  $(A \xrightarrow{*} d_1 w, A \xrightarrow{*} d_2 w)$ .

Similarly, the strong homotopy extension property in the monoidal path algebra  $P(G)$  looks like a "lattice completion" property (that is, a unification property) to a topological loop (that is, a unification pair).

The next group of definitions is a formal translation of the concept of fundamental groupoid in the category  $\mathcal{W}$  of CW complexes to the category  $\mathcal{E}$  of small categories.

#### Definition 3.2.1:

- (a) If  $A$  is a small category, then the *path category*  $P(A)$  of  $A$  is the free category of all composable morphisms  $p$  in  $A$

$$p = w_0 \rightarrow_{a_1} w_1 \rightarrow_{a_2} w_2 \rightarrow \dots \rightarrow_{a_{k-1}} w_{k-1} \rightarrow_{a_k} w_k$$

(that is,  $P(A) \simeq \bigcup NA_k$ ). Each path  $p$  can be viewed as a functor  $p : \mathbf{1} \rightarrow A$  with  $p(0) = w_0$ ,  $p(1) = w_k$ , and

$$p(0 \rightarrow 1) = a_1 \circ a_2 \dots \circ a_k : w_0 \rightarrow w_k.$$

- (b) Two paths  $p$  and  $q$ ,

$$p = w_0 \rightarrow_{a_1} w_1 \dots w_{k-1} \rightarrow_{a_k} w_k$$

$$q = y_0 \rightarrow_{b_1} y_1 \dots y_{m-1} \rightarrow_{b_m} y_m$$

are *strongly homotopic* relative to the boundary  $\partial \mathbf{1}$  of  $\mathbf{1}$ , if and only if there exists a homotopy  $H: \mathbf{1} \times \mathbf{1} \rightarrow A$  such that

$$\begin{aligned} H| \mathbf{1} \times \{0\} &= p & \text{and} & \quad H| \mathbf{1} \times \{1\} = q, \\ y_0 &= H| \{0\} \times \mathbf{1} = w_0 & \text{and} & \quad w_k = H| \{1\} \times \mathbf{1} = y_m; \end{aligned}$$

that is,  $p$  and  $q$  have the same domains and codomains, and there is a natural transformation (a natural method of translating) path  $p$  to path  $q$ .

- (c) For each path  $p \in P(A)$ , there exists an inverse path  $p^{-1} \in P(A^{\text{op}})$  that reverses the direction of the arrows. Expand the path category  $P(A)$  to  $P^{-1}(A)$  adding these formal path inverses.
- (d) The *strong fundamental groupoid*  $\Pi(A)$  of  $A$  is the expanded path algebra  $P^{-1}(A)$  modulo the strong homotopy relation  $\sim_s$  (relative to  $\partial \mathbf{1} \hookrightarrow \mathbf{1}$ ). If  $p \in P(A)$ , let  $[p]$  denote its homotopy congruence class in  $\Pi(A)$ .
- (e) Suppose  $E$  is an equational theory on  $A$  (that is,  $E$  consists of pairs  $(c_1, c_2) \in A(p, q)$  of composed morphisms, that are *formally identified*) and let  $[E]$  denote the smallest congruence relation on  $A$  generated by  $E$ . Two paths  $p$  and  $q$ ,

$$p = w_0 \rightarrow_{a_1} w_1 \dots w_{k-1} \rightarrow_{a_k} w_k$$

$$q = y_0 \rightarrow_{b_1} y_1 \dots y_{m-1} \rightarrow_{b_m} y_m$$

with  $w_0 = y_0$  and  $w_k = y_m$  are *strongly homotopic modulo*  $E$ ,  $p \sim_E q$ , if and only if  $[p] = [q]$  in the *E-fundamental groupoid*  $\Pi(A/[E])$ ; that is, there exists a finite "zig-zag"  $(p, p_1, \dots, p_{n-1}, q)$  of paths in  $A(w_0, w_k)$  such that for  $1 \leq i \leq n$ , the path pair  $(p_{i-1}, p_i) \in [E]$ .  $\square$

A careful translation of the definitions for the syntactic algebras guarantees:

**Theorem 3.3.2:** *For a nontrivial useful CFG  $G$  with typed variables, suppose that the ambiguity congruence  $\mathcal{A}(G)$  for the monoidal path algebra  $P(G)$  is generated by a (finite) collection  $A$  of ambiguity pairs (that is,  $\mathcal{A}(G) = [A]$ ). Then a derivation pair  $(A \xrightarrow{*}_{d_1} w, A \xrightarrow{*}_{d_2} w)$  is an ambiguity pair*



in  $\mathcal{A}(G) = [A]$  if and only if, as strong homotopy classes in  $\Pi(P(G)/[A])$ ,

$$[A \xrightarrow{*}_{d_1} w] = [A \xrightarrow{*}_{d_2} w] .$$

**Proof:** Suppose that the ambiguity congruence  $\mathcal{A}(G)$  for the monoidal path algebra  $P(G)$  is generated by the finite collection

$$A = \{(A_j \xrightarrow{*}_{d_{1j}} w_j, A_j \xrightarrow{*}_{d_{2j}} w_j) \mid 1 \leq j \leq m\} .$$

By definition of the monoidal congruence  $\mathcal{A}(G) = [A]$  on  $P(G)$ , each derivation pair

$$(A \xrightarrow{*}_{d_1} w, A \xrightarrow{*}_{d_2} w) \in \mathcal{A}(G) = [A]$$

if and only if there exists a finite "sequence"

$$((A \xrightarrow{*}_{d_1} w, A \xrightarrow{*}_{h_1} w), (A \xrightarrow{*}_{h_1} w, A \xrightarrow{*}_{h_2} w), \dots, (A \xrightarrow{*}_{h_{n-1}} w, A \xrightarrow{*}_{d_2} w))$$

of ambiguity pairs such that, for each  $1 \leq r \leq n$ , there exists  $(A_j \xrightarrow{*}_{d_{1j}} w_j, A_j \xrightarrow{*}_{d_{2j}} w_j) \in A$  with compositions:

$$\begin{array}{ccccccc} h_{r-1} = A * & \xrightarrow[e_r]{} & s_{r0} A_j s_{r1} * & \xrightarrow[d_{1j}]{} & s_{r0} w_j s_{r1} * & \xrightarrow[f_r]{} & w \\ \downarrow \text{Id} & & \downarrow \text{Id} & & \downarrow \text{Id} & & \downarrow \text{Id} \\ h_r = A * & \xrightarrow[e_r]{} & s_{r0} A_j s_{r1} * & \xrightarrow[d_{2j}]{} & s_{r0} w_j s_{r1} * & \xrightarrow[f_r]{} & w . \end{array} \quad (*)$$

But the above definition is precisely Definition 3.2.1(e) for  $\Pi(P(G)/[A])$ ; that is,  $[A \xrightarrow{*}_{d_1} w] = [A \xrightarrow{*}_{d_2} w]$  in  $\Pi(P(G)/[A])$  if and only if there exists a finite "zig-zag"  $(d_1, h_1, \dots, h_{n-1}, d_2)$  of paths from  $A$  to  $w$  such that, for  $1 \leq r \leq n$ , there exists a diagram of form  $(*)$  with  $h_{r-1} \sim_A h_r$ .  $\square$

**Corollary 3.3.3:** For a nontrivial useful CFG  $G$  with typed

variables, the Ambiguity Problem is equivalent to the Homotopy Generation Problem.  $\square$

The next definition makes precise the concept of homotopy extension in the path category  $\mathbf{P}(\mathbf{A})$ .

**Definition 3.3.4:** Suppose  $E$  is an equational theory on  $\mathbf{A}$  and  $(p, q) \in \mathbf{P}(\mathbf{A})$  is a path pair with

$$\begin{aligned} p &= w_0 \rightarrow_{a_1} w_1 \dots w_{k-1} \rightarrow_{a_k} w_k \\ q &= y_0 \rightarrow_{b_1} y_1 \dots y_{m-1} \rightarrow_{b_m} y_m \end{aligned}$$

with  $w_0 = y_0$ .

- (a) The path pair  $(p, q)$  satisfies the *strong homotopy extension property* with respect to  $E$  if and only if there exist extending paths

$$\begin{aligned} p_1 &= w_k \rightarrow_{a_{k+1}} w_{k+1} \dots w_{k+r-1} \rightarrow_{a_{k+r}} w \\ q_1 &= y_m \rightarrow_{b_{m+1}} y_{m+1} \dots y_{m+s-1} \rightarrow_{b_{m+s}} w \end{aligned}$$

such that the composed paths  $p_1 \circ p \sim_E q_1 \circ q$  (that is,  $[p_1 \circ p] = [q_1 \circ q]$  in  $\Pi(\mathbf{A}/[E])$ .)

- (b) A finite set  $mhe(p, q) = \{(p_{1k}, q_{1k})\}$  of homotopy extensions is a *minimal homotopy extension set* if and only if, for any homotopy extension pair  $(p_1 : w_k \rightarrow w_1, q_1 : y_m \rightarrow w_1)$  of  $(p, q)$ , there exists a *unique*  $(p_{1k}, q_{1k}) \in mhe(p, q)$  and a (not necessarily unique) path

$$h = w \rightarrow_{c_1} x_1 \rightarrow_{c_2} \dots \rightarrow_{c_{n-1}} x_{n-1} \rightarrow_{c_n} w_1$$

such that

$$p_1 \circ p \sim_E h \circ p_{1k} \circ p \sim_E h \circ q_{1k} \circ q \sim_E q_1 \circ q. \quad \square$$

Another careful translation of the definitions for the syntactic algebras guarantees:

**Proposition 3.3.5:** For a nontrivial useful CFG  $G$  with typed variables, suppose that the ambiguity congruence  $\mathcal{A}(G)$  for the monoidal path algebra  $\mathbf{P}(G)$  is generated by a (finite) collection  $A$  of ambiguity pairs (that is,  $\mathcal{A}(G) = [A]$ ). Then for a term pair  $(A \xrightarrow{\lambda_1} w_1, A \xrightarrow{\lambda_2} w_2) \in \mathbf{T}(G)$ ,

- (i) The unifier set  $u(t_1, t_2) \neq \emptyset$  if and only if the homotopy

extension set  $he(r(t_1), r(t_2)) \neq \emptyset$  (where  $r : T(G) \rightarrow P(G)$  is the retract homomorphism given in Definition 2.2.5(b)).

- (ii) A finite mgu set  $mgu(t_1, t_2)$  exists if and only if a finite minimal homotopy extension set  $mhe(r(t_1), r(t_2))$  exists.

**Proof:** The unifier set  $u(t_1, t_2)$  is nonempty (that is, there exists  $(\theta_1, \theta_2) \in u(t_1, t_2)$ ) if and only if the derivation pair

$$(A \xrightarrow{r(t_1)} S(r(t_1)) \xrightarrow{d(\theta_1)} w, \\ A \xrightarrow{r(t_2)} S(r(t_2)) \xrightarrow{d(\theta_2)} w) \in \mathcal{A}(G) = [A],$$

with  $S(\theta_1 t_1) = w = S(\theta_2 t_2)$ . By Theorem 3.3.2, this property is equivalent to

$$[A \xrightarrow{d(\theta_1) \circ r(t_1)} w] = [A \xrightarrow{d(\theta_2) \circ r(t_2)} w] \in \Pi(P(G)/[A]).$$

Hence, by Definition 3.3.4(a), the unifier set  $u(t_1, t_2) \neq \emptyset$  if and only if the homotopy extension set  $he(r(t_1), r(t_2)) \neq \emptyset$ .

Similarly, by Definition 2.3.4(e), a finite set  $\{(\mu_{1k}, \mu_{2k})\}$  of unifiers is a mgu set  $mgu(t_1, t_2)$  if and only if, for any unifier  $(\theta_1, \theta_2) \in u(t_1, t_2)$ , there is a unique  $(\mu_{1k}, \mu_{2k}) \in mgu(t_1, t_2)$  and a substitution  $\theta$  so that

$$\begin{aligned} d(\theta_1) \circ r(t_1) &\sim_A d(\theta) \circ d(\mu_{1k}) \circ r(t_1) \\ &\sim_A d(\theta) \circ d(\mu_{2k}) \circ r(t_2) \\ &\sim_A d(\theta_2) \circ r(t_2). \end{aligned}$$

By Definition 3.3.4(b), this property is equivalent to the finite set  $\{(d(\mu_{1k}), d(\mu_{2k}))\}$  of derivation extension pairs being a minimal homotopy extension set  $mhe(r(t_1), r(t_2))$ . Hence,  $mgu(t_1, t_2)$  exists if and only if  $mhe(r(t_1), r(t_2))$  exists.  $\square$

**Corollary 3.3.6:** For a nontrivial useful CFG  $G$  with typed variables, the undecidable Unification Problem is equivalent to the Homotopy Extension Problem.  $\square$

**Example 3.3.7:** For the FFP CFG  $G_{FP}$ , it is shown in Example 2.3.5, that the path/derivation pair  $(d_L, d_R)$ , where

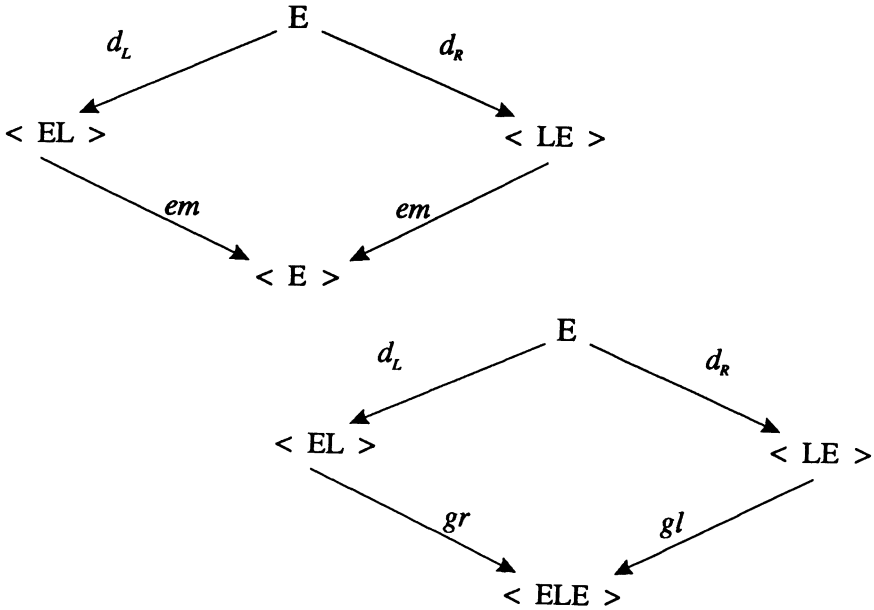
$$d_L = E \xrightarrow{\text{br}} \langle L \rangle \xrightarrow{\text{gl}} \langle EL \rangle$$

$$d_R = E \xrightarrow{\text{br}} \langle L \rangle \xrightarrow{\text{gr}} \langle EL \rangle ,$$

has two minimal homotopy extension pairs; that is

$$\text{whe}(d_L, d_R) = \{ (\langle EL \rangle \xrightarrow{\text{em}} \langle E \rangle , \langle LE \rangle \xrightarrow{\text{em}} \langle E \rangle , \\ (\langle EL \rangle \xrightarrow{\text{gr}} \langle E, LE \rangle , \langle LE \rangle \xrightarrow{\text{gl}} \langle ELE \rangle) \} .$$

The corresponding strong homotopy diagrams in  $P(G)$  are:



### 3.4 Sufficient Conditions for the Existence of Strong Generation Homotopies

Theorem 3.3.2, Proposition 3.3.5 and Theorem 2.3.10 guarantee that:

**Theorem 3.4.1:** Suppose that  $G$  is a nontrivial useful CFG with typed variables. If:

- (1) The path algebra  $P(G)$  is  $\mathcal{A}(G)$ -free;
- (2) For each  $A$ -production pair  $(A \xrightarrow{p_1} S(p_1), A \xrightarrow{p_2} S(p_2))$ , there exists a minimal finite homotopy extension set

$$\text{mhe}(A \xrightarrow{p_1} S(p_1) , A \xrightarrow{p_2} S(p_2)) =$$

$$\{(S(p_1) \xrightarrow{d_{1k}} w_k, S(p_2) \xrightarrow{d_{2k}} w_k)\}$$

of simple homotopy extensions;  
then the fundamental groupoid  $\Pi(\mathbf{P}(\mathbf{G})/\mathcal{A}(\mathbf{g}))$  is generated by the union

$$\cup\{(d_{1k} \circ p_1, d_{2k} \circ p_2) \mid (d_{1k}, d_{2k}) \in mhe(p_1, p_2)\}$$

(over all the production pairs  $(p_1, p_2)$ ) . □

The FFP grammar  $\mathbf{G}_{\text{FP}}$  satisfies Condition (1) of Theorem 3.4.1 and the unification condition equivalent to Condition (2). From Example 2.3.5, it follows that the strong fundamental groupoid  $\Pi(\mathbf{P}(\mathbf{G}_{\text{FP}})/\mathcal{A}(\mathbf{G}_{\text{FP}}))$  is generated by the L-derivation pairs:

$$\begin{aligned} &\bullet (L \xrightarrow{gl} E \ L \xrightarrow{em} E, L \xrightarrow{gr} L \ E \xrightarrow{em} E) \\ &\bullet (L \xrightarrow{gl} E \ L \xrightarrow{gr} ELE, L \xrightarrow{gr} L \ E \xrightarrow{gl} ELE) . \end{aligned}$$

#### 4. References

Baa89 BAADER, F., Unification properties of commutative theories: A categorical approach, *Proceedings of Category Theory and Computer Science*, Manchester, UK, September 1989, LNCS 389, Springer Verlag, New York, 1989, 273-299.

BaDe87 BACKMAIR, L. & DERSHOWITZ, N., Completion for rewriting modulo a congruence, *Proceedings of Rewriting Techniques and Applications*, Bordeaux, France, May 1987, LNCS 256, Springer Verlag, New York, 1987, 192-203.

Ba78 BACKUS, J., Can programming be liberated from the von Neumann style? A functional style and its algebra of programs, *Communications of the ACM*, 21(1978) 613-641.

Be75 BENSON, D., The basic algebraic structures in categories of derivations, *Infor. and Contr.*, 28(1975) 1-29.

BiWa88 BIRD, R. & WALDER, P., *Introduction to Functional Programming*, Prentice Hall, New York, 1988.

BHS89 BÜRKERT, H.-J., HEROLD, A. & SCHNIDT-SCHAUSS, M. On equational theories, unification and (un)decidability, *J. Symbolic Computation*, 8(1989) 3-49.

BuRy86 BURSTALL, R. & RYDEHEARD, D., The unification of terms: A category-theoretic algorithm, *Proceedings of Category Theory and Computer Programming*, Guildford, UK, Sept. 1985, LNCS 240, Springer Verlag, New York, 1986, 493-505.

Co65 COHN, P., *Universal Algebra*, Harper & Row, New York, 1965.

DaWe83 DAVIS, M. & WEUYKER, E., *Computability, Complexity and Languages*, Academic Press, New York, 1983.

D<sup>2</sup>Qu78 DENNING, P., DENNIS, J. & QUALITZ, J., *Machines, Languages and Computation*, Prentice-Hall, Englewood Cliffs, NJ, 1978.

Ei87 EISENBACH, S., *Functional Programming: Languages, Tools and Architectures*, John Wiley & Sons, New York, 1987.

Fa84 FAGES, F., Associative-commutative unification, *Proceedings of the 7th C.A.D.E.*, LNCS 170, Springer Verlag, New York, 1984, 194-208.

Fr79 FRANK, G., *Virtual Memory Systems for Closed Applicative Language Interpreters*, PhD Dissertation, University of North Carolina at Chapel Hill, 1979.

FLPS82 FRANK, G., LATCH, D., PETRO, J. & STANAT, D., Applications of algebraic topology of small categories to the semantics of formal functional programming languages, UNC-CH Technical Report in Computer Science #82-009, Chapel Hill, NC, 1982.

FriLa81 FRITSCH, R. & LATCH, D., Homotopy inverses for nerve, *Math. Z.*, 177(1981) 147-179.

GaZi67 GABRIEL, P. & ZISMAN, M., *Calculus of Fractions and Homotopy Theory*, Springer Verlag, New York, 1967.

Gi66 GINSBURG, S., *Mathematical Theory of Context-Free Languages*, McGraw-Hill, New York, 1966.

Ha78 HARRISON, M., *Introduction to Formal Language Theory* Addison-Wesley, Reading, MA, 1978.

HaW<sup>3</sup>85 HALPERN, J., WILLIAMS, J., WIMMERS, E., & WINKLER, T., Denotational semantics and rewrite rules for FP,

*Proceedings of the Twelfth ACM Symposium of Principles of Programming Languages*, January 1985, 108-120.

HoUI79 HOPCROFT, J. & ULLMAN, J., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

Hu89 HUDAK, P., The conception, evolution and application of functional programming languages, to appear in *ACM Computing Surveys*.

Hue75 HUET, G., A unification algorithm for the typed  $\lambda$ -calculus, *Theor. Comp. Sci.*, 1(1975) 25-57.

Hue80 HUET, G., Confluent reductions: abstract properties and applications to term rewriting systems, *J. of Assoc. Comp. Mach.*, 27(1980) 797-821.

Kn89 KNIGHT, K., Unification: A multidisciplinary survey, *ACM Computing Surveys*, 21(1989) 93-124.

KnBe70 KNUTH, D. & BENDIX, P., Simple word problems in universal algebras, *Computational Problems in Abstract Algebra*, Leech J., ed., Pergamon Press, 1970, 263-297.

LaSc86 LAMBEK, J. & SCOTT, P., *Introduction to Higher Order Categorical Logic*, Cambridge University Press, London, 1986.

La73 LATCH, D., On derived functors of limit, *Trans. of Amer. Math. Soc.*, 181(1973) 155-163.

La75 LATCH, D., The uniqueness of homology for the category of small categories, *J. of Pure and Applied Algebra*, 9(1975) 221-237.

La79 LATCH, D., A fibred homotopy equivalence and homology theories for the category of small categories, *J. of Pure and Applied Algebra*, 15(1979) 247-269.

La89a LATCH, D., Finite generation of ambiguity in context free languages, *J. of Pure and Applied Algebra*, 57(1989) 229-263.

La89b LATCH, D., An application of minimal context-free intersection partitions to rewrite rule consistency checking, *Proceedings of the AMS-IMS-SIAM Conference on Categories in*

Computer Science and Logic, Boulder CO, 1987, AMS Contemporary Mathematics, 92(1989) 241 -260.

LaMi74 LATCH, D. & MITCHELL, B., On the difference between homological and cohomological dimensions, *J. of Pure and Applied Algebra*, 5(1974) 333-343.

LaSi88 LATCH, D. & SIGAL, R., Generating evaluation theorems for functional programming languages, *Proceedings of the Third International Symposium on Methodologies for Intelligent Systems*, Torino, Italy, October 1988, 47-58.

LSR90 LATCH, D., SIGAL, R. & RUGGERI, R., Generating non-deterministic evaluation theorems for functional languages, to appear in *Proceedings of the Fourth Workshop on Computer Science Logic*, Heidelberg, FRG, October 1990.

LTW79 LATCH, D., THOMASON, R. & WILSON, S., Simplicial sets from categories, *Math. Z.*, 164(1979) 195-214.

LePa81 LEWIS, H. & PAPADIMITRIOU, K., *Elements of the Theory of Computation*, Prentice Hall, Englewood Cliffs, NJ., 1981.

Mac71 MAC LANE, S., *Categories for the Working Mathematician*, Springer Verlag, New York, 1971.

Mi57 MILNOR, J., The geometric realization of a semi-simplicial complex, *Ann. Math.*, 65(1957), 357-362.

Ne80 NELSON, E., Categorical and topological aspects of formal languages, *Math. System Theory* 13(1980) 255-273.

Ro65 ROBINSON, J., A machine-oriented logic based on the resolution principle, *J. of Assoc. for Comp. Mach.*, 12(1965) 23-41.

Rot88 ROTMAN, J., *An Introduction to Algebraic Topology*, Springer Verlag, New York, 1988.

Ru87 RUST, T., An algebraic model for programming languages, *Computer Languages* 12(1987) 173-195.



RySt87 RYDEHEARD, D.E. & STELL, J.C., Foundations of equational deduction: A categorical treatment of equational proofs and unification algorithms, *Proceedings of Category Theory and Computer Science*, Edinburgh, UK, September 1987, LNCS 389, Springer Verlag, New York, 1987, 114-139.

ShSt87 STERLING, E. & SHAPIRO, R., *The Art of PROLOG*, MIT Press, Cambridge, MA, 1987.

Si84 SIEKMANN, J., Universal unification, *Proceedings of the 7th C.A.D.E.*, LNCS 170, Springer Verlag, New York, 1984, 1-42.

Si89 SIEKMANN, J., Unification theory, *J. Symbolic Computation*, 7(1989) 207-274.

September 14, 1990

Department of Mathematics  
North Carolina State University  
Raleigh, NC 27695-8205  
email: dml@ncsuvvm.bitnet