Journal of Logic and Computation Advance Access published August 9, 2013

The syntactic concept lattice: Another algebraic theory of the context-free languages?

ALEXANDER CLARK, *Department of Philosophy, King's College London, The Strand, London WC2R 2LS.*
E-mail: alexsclark@gmail.com

Abstract

The syntactic concept lattice is a residuated lattice associated with a given formal language; it arises naturally as a generalization of the syntactic monoid in the analysis of the distributional structure of the language. In this article we define the syntactic concept lattice and present its basic properties, and its relationship to the universal automaton and the syntactic congruence; we consider several different equivalent definitions, as Galois connections, as maximal factorizations and finally using universal algebra to define it as an object that has a certain universal (terminal) property in the category of complete idempotent semirings that recognize a given language, applying techniques from automata theory to the theory of context-free grammars (CFGs). We conclude that grammars that are minimal, in a certain weak sense, will always have non-terminals that correspond to elements of this lattice, and claim that the syntactic concept lattice provides a natural system of categories for representing the denotations of CFGs.

Keywords: Context-free grammars, universal automaton, distributional learning.

1 Introduction

The theory of finite automata as representations of regular languages is now very well developed; there are three types of canonical representations. First, the canonical minimal deterministic finite automation (DFA), based on the Myhill–Nerode congruence; secondly, the universal automaton (UA) [12, 27]; and finally, the canonical residual automaton [14] occupies a position between these two, being non-deterministic but quite close to the DFA. All of these are based in some sense on the relation between prefixes and suffixes: two strings p and s are related if their concatenation is in the language. Formally, we might write $p \sim_L s$ iff $ps \in L$. The Myhill–Nerode congruence says that two prefixes are identical if they are related to exactly the same set of suffixes; this gives a partition of the set of prefixes. On the other hand, the UA is based on the Galois connection between prefixes and suffixes, though the standard presentations do not present it in this way. This gives, rather than a partition, an overlapping hierarchy, a lattice, whose automata theoretic interpretation is therefore naturally non-deterministic.

When we move to the theory of context-free languages and grammars, we need to consider rather the relation between contexts and substrings, where a context is an ordered pair of strings or, equivalently, a string with a gap in it. We might write this as $(l,r) \sim_L u$ iff $lur \in L$. In this case we have two relevant structures: the syntactic congruence, which corresponds exactly to the Myhill–Nerode congruence, and the recently introduced syntactic concept lattice [6, 7, 9, 10] that corresponds to the UA. This is very closely related to the syntactic semiring [29, 30]; indeed for regular languages they are isomorphic.[1] The syntactic congruence has been classically studied (e.g. [31]) and has recently

[1]The syntactic semiring is a quotient of the set of finite sets of strings.

doi:10.1093/logcom/ext037

TABLE 1. Relationship between congruences and lattices when considering regular grammars (on the left) and context-free grammars (on the right)

	$p \sim s$	$(l,r) \sim u$
Equivalence relation	Myhill–Nerode relation	Syntactic congruence
Lattice	Universal automaton	Syntactic concept lattice

given rise to a number of algorithms for the grammatical inference of context-free grammars (CFGs) [5, 11] that are analogous to the results for regular inference using (DFAs) obtained by Angluin [1, 2]. We also note the result of Bollig *et al.* for learning residual automata [3]. However, whereas with regular languages, the restriction to deterministic automata does not reduce the class of languages that can be expressed,[2] with CFGs the situation is quite different. The class of languages that can be expressed using non-terminals that correspond to congruence classes under the syntactic congruence is a subclass of context-free languages, which while including all the regular languages, does not include such simple languages as the palindrome language over a two letter alphabet.

In this article we define the syntactic concept lattice, and give its basic properties, focusing on its relationships to the syntactic congruence and the UA. We also demonstrate that it has a desirable property analogous to that of the UA: namely that it contains a morphic image of every CFG for the language. Alternatively, we show that it is the unique (up to isomorphism) smallest structure that interprets a CFG for the language; in other words it is the smallest and simplest object that provides a semantics for the grammar. We can view it as the syntactic monoid lifted from monoids (semigroups with unit) to semirings; as lifting from strings to sets of strings. Table 1 shows the relationship between these structures.

In Section 2 we will define our notation, and define the basic algebraic structures we use: monoids, and complete idempotent semirings (CISs), as well as the basic notions of CFGs. Then, in Section 3 we define the syntactic concept lattice, and explain its basic properties. We then, in Section 4 start to develop an algebraic theory for the semantics of CFGs, algebraic structures that we use to interpret them. We then define the basic ideas of syntactic morphisms—i.e. morphisms of CFGs in Section 5, which leads us onto the universal morphism using the Syntactic Concept Lattice (Section 6).

We then look briefly at the theory of regular languages in Section 7, discussing the UA and its relationship(s) to the Syntactic Concept Lattice. We then conclude with some discussion of the relevance of this work for representational assumptions in linguistics.

2 Preliminaries

We use standard set theoretic notation; the empty set is $\emptyset$, and $\subseteq$, $\supseteq$, $\cup$ and $\cap$ have their normal meanings. We write $\mathcal{P}(X)$ for the power set of X, and $\mathcal{F}(X)$ for the collection of only finite subsets of X.

We have a finite non-empty set Σ which we call an alphabet. Let Σ^* be the set of all finite length strings over Σ. We write λ for the empty string. We will use letters at the beginning of the alphabet,

[2]It is worth recalling here that the size of the minimal DFA for a given language may be exponentially larger than the size of the minimal non-deterministic automaton (NFA) for the same language, so the two representations are not entirely equivalent.

a, b, c to refer to elements of Σ and also abusing notation slightly to strings of length 1. We write $|u|$ for the length of a string $u \in \Sigma^*$; so $|\lambda| = 0$. We will write $|u|_a$ for the number of times the element $a \in \Sigma$ occurs in the string u. We will write concatenation of strings u, v as uv and sometimes $u \cdot v$ when we want to be explicit.

A language is any subset of Σ^*. For two such sets A, B we define the product $AB = \{uv \mid u \in A, v \in B\}$. A context is an ordered pair of strings $(l, r) \in \Sigma^* \times \Sigma^*$. We can combine a context (l, r) and a string u using the wrapping operation which we write $(l, r) \odot u = lur$. (λ, λ) is the empty context; clearly $(\lambda, \lambda) \odot u = u$ for any u. We will lift these operations to sets in the standard way. If C is a set of contexts, and S is a set of strings, then $C \odot S = \{lur \mid (l, r) \in C, u \in S\}$. If X, Y are sets of strings then $X \times Y$ is the set of contexts $\{(x, y) \mid x \in X, y \in Y\}$. We also extend the wrap operation $\odot$ to contexts as $(x, y) \odot (p, q) = (xp, qy)$, so that $(x, y) \odot (p, q) \odot u$ is associative, and to sets of contexts in the natural way.

2.1 *Monoids*

We use a few standard algebraic structures; in order to make this self-contained we define all the basic properties here; we will sometimes be rather casual about the difference between an algebraic structure, and its reducts and expansions.

A monoid is a set M together with a unit, 1_M and an associative operation $\circ_M$. A monoid morphism from $\langle M, 1_M, \circ_M \rangle$ to $\langle N, 1_N, \circ_N \rangle$ is a function $h: M \to N$ such that $h(a \circ_M b) = h(a) \circ_N h(b)$ and $h(1_M) = 1_N$. Then, note that $\langle \Sigma^*, \lambda, \cdot \rangle$ is a monoid, the free monoid over Σ; the operation is simply concatenation.

We write M_1 for the trivial one element monoid.

2.2 *Semirings*

Rather than just a monoid (a semigroup with unit) we will consider a slightly more complicated structure which in addition to the concatenation operation $\circ$ has another operation which we will write as $\vee$.

Definition 2.1
An idempotent semiring is a structure which we write as $\langle M, \vee, \circ, \bot, 1 \rangle$, where

- $\langle M, \vee, \bot \rangle$ is a commutative idempotent monoid with $\bot$ as unit; i.e. $x \vee \bot = x = \bot \vee x$. Idempotent means that $x \vee x = x$.
- $\langle M, \circ, 1 \rangle$ is a monoid.
- $\bot \circ x = x \circ \bot = \bot$.
- $\circ$ distributes over $\vee$, so $a \circ (b \vee c) = (a \circ b) \vee (a \circ c)$ and $(b \vee c) \circ a = (b \circ a) \vee (c \circ a)$.

The free idempotent semiring over Σ is the collection of finite sets of strings of Σ, $\mathcal{F}(\Sigma^*)$, where $1 = \{\lambda\}$, $\bot = \emptyset$, $\vee = \cup$ and $X \circ Y = XY$ is concatenation extended to sets of strings.

We need to add some infinitary operations here—in particular we want to be able to take the $\vee$ (the union, intuitively) over infinite sets. This structure is then a CIS [15, 16, 20].

Definition 2.2
A complete idempotent semiring (CIS) is an idempotent semiring $\langle M, \vee, \circ, \bot, 1 \rangle$, where additionally

- $\vee$ is defined over arbitrary sets. Given a family of sets $\{x_i \mid i \in I\}$, there is an element $\bigvee_{i \in I} x_i$.

- $\circ$ distributes over $\vee$: if X is a collection of elements then $a \circ (\bigvee_{i \in I} x_i) = \bigvee_{i \in I} a \circ x_i$ and $(\bigvee_{i \in I} x_i) \circ a = \bigvee_{i \in I} (x_i \circ a)$.

The free structure generated by this over Σ is $\mathcal{P}(\Sigma^*)$.

The elements of this are finite or infinite sets of finite length strings over Σ. If we had just an idempotent semiring, the free structure would be the collection of finite sets of strings—since the languages we use might be infinite, we need to allow infinite sets and therefore a complete semiring.

These CIS have a natural partial order.

Proposition 2.3
Given a CIS, we can define a relation $\leq$ by $X \leq Y$ iff $Y = X \vee Y$. This relation is a partial order.

Complete idempotent semirings are the appropriate structures for interpreting CFGs as systems of equations [19, 25].[3]

Note that a CIS is an ω-complete partial order. Every infinite ascending chain $a_1, a_2, \ldots$, where $a_i \leq a_{i+1}$, has a least upper bound, which is equal to $\bigvee_i a_i$.

2.3 Residuated lattices

Definition 2.4
A complete lattice is a structure $\langle S, \leq \rangle$, where every subset A of S has a greatest lower bound and least upper bound denoted by $\bigwedge A$ and $\bigvee A$. In such a lattice we can define binary operations $\vee, \wedge$ on S by $x \vee y = \bigvee \{x, y\}$ and $x \wedge y = \bigwedge \{x, y\}$, and top element $\top = \bigvee S$, and bottom element $\bot = \bigwedge S$.

Definition 2.5 [17]
A complete residuated lattice is a complete lattice $\langle S, \leq \rangle$, with three additionally binary operations $\circ, /, \backslash$ and a unit 1, where $\langle S, 1, \circ \rangle$ is a monoid and the three binary operations satisfy the identities, for all $X, Y, Z \in S$.

$$Y \leq X \backslash Z \text{ iff } X \circ Y \leq Z \text{ iff } X \leq Z / Y$$

Proposition 2.6 [23]
Any complete idempotent semiring can be expanded to a complete residuated lattice by defining

$$X \wedge Y = \bigvee \{Z \mid Z < X, Z < Y\}$$
$$X / Y = \bigvee \{Z \mid Z \circ Y \leq X\}$$
$$X \backslash Y = \bigvee \{Z \mid X \circ Z \leq Y\}.$$

2.4 Congruences

A congruence on an algebra is an equivalence relation on the elements that respects the operations of the algebra. A congruence of a monoid then is an equivalence relation $\cong$ such that if $u \cong u$ and $v \cong v'$ then $u \circ v \cong u' \circ v$. We write $[u]_{\cong}$ for the equivalence class of u.

The equivalence classes under a congruence form an algebra where $[1]_{\cong}$ is a unit, and the concatenation operation is $[u]_{\cong} \circ [v]_{\cong} = [uv]_{\cong}$. This is well defined since it is a congruence.

[3] I am grateful to Hans Leiss for making this point.

A homomorphism is analogously a function between algebras which respects the operations. The congruences and the surjective homomorphisms are closely related. Given a surjective homomorphism h from $A \to B$, the equivalence relation on A given by $u \cong v$ iff $h(u) = h(v)$ is a congruence. Conversely, given a congruence $\cong$, the natural map $h : u \to [u]_{\cong}$ is a surjective homomorphism.

Given a free algebra, like the free monoid Σ^*, the congruences form a lattice; and therefore also do the corresponding quotient monoids. The bottom element is Σ^*, the finest congruence, and the top is the coarsest congruence where everything is equivalent to everything else and there is only one element in the quotient. For any surjective homomorphism from Σ^* onto a monoid A, we can identify a congruence which is an element of this lattice. Similarly if we have two congruences, one of which is finer than the other, then there is a surjective homomorphism from the finer to the coarser.

The same applies, as we shall see, for congruences and homomorphisms of CIS.

2.5 *The syntactic congruence*

The syntactic concept lattice, which is the main topic of this article, is best understood in relation to the syntactic congruence [15]; therefore, we recall the elementary properties of the syntactic congruence in a notation consistent with the rest of this article.

Definition 2.7
For a language L and $w \in \Sigma^*$; define the distribution of w to be

$$C_L(w) = \{(l,r) \in \Sigma^* \times \Sigma^* \mid lwr \in L\}.$$

Definition 2.8
We say that two strings $u, v \in \Sigma^*$ are congruent w.r.t. a language L, written $u \equiv_L v$ iff $C_L(u) = C_L(v)$.

Proposition 2.9
$\equiv_L$ is an equivalence relation and a congruence of the monoid Σ^*; i.e. $u \equiv_L v$ implies that for any $z \in \Sigma^*$, $uz \equiv_L vz$ and $zu \equiv_L zv$.

Definition 2.10
We write $[u]_L$ for the equivalence class of u under $\equiv_L$.

$$[u]_L = \{v \in \Sigma^* \mid u \equiv_L v\}$$

Proposition 2.11
For any language L and any strings, u, v

$$[uv]_L \supseteq [u]_L [v]_L.$$

The following proposition is classical, attributed to Myhill [31].

Proposition 2.12
The number of congruence classes of L is finite iff L is regular.

Example 2.13
Suppose $L = \Sigma^*$. Then there is one congruence class which consists of all strings.

EXAMPLE 2.14
Consider the regular language $L=(ab)^*$. This has the following 6 congruence classes which are all regular sets. $(ab)^+$, $\{\lambda\}$, $a(ba)^*$, $b(ab)^*$, $(ba)^+$, and finally all other strings (i.e. not in the preceding 5 classes) which we can express using the regular expression: $\Sigma^* bb \Sigma^* \cup \Sigma^* aa \Sigma^*$.

EXAMPLE 2.15
Consider the language $O_1 = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$. Define $c(w) = |w|_a - |w|_b$. Clearly $[u]_L = \{v | c(u) = c(v)\}$, and so this has an infinite number of congruence classes indexed by the integers where for $n \in \mathbb{Z}$, $C_n = \{w \mid c(w) = n\}$.

DEFINITION 2.16
A congruence $\sim$ saturates a language L if $x \sim y$ implies that $x \in L$ iff $y \in L$.

Informally, this just means that all of the classes are either inside or outside the language.

PROPOSITION 2.17
The syntactic congruence saturates L, and is the coarsest congruence that saturates L.

PROOF. Suppose $\sim$ is a congruence that saturates L. and suppose $x \sim y$. Take some l, r. Then $lxr \sim lyr$ as it is a congruence. Since $\sim$ saturates L then $lxr \in L$ iff $lyr \in L$. Therefore $x \equiv_L y$ and so $\sim$ is finer than $\equiv_L$. ■

DEFINITION 2.18
Given a monoid M and a morphism h from Σ^* to M, we say that the monoid *recognizes* the language L through h if there is a set $X \subseteq M$ such that $h^{-1}(X) = L$, i.e. such that

$$\{w \in \Sigma^* \mid h(w) \in X\} = L.$$

Note that if M recognizes L through the morphism h, then $h(M)$ also recognizes L through the surjective morphism h. $h(M)$ is a submonoid of M if h is a homomorphism. In this case we are not interested in all of M but only in the submonoid $h(M)$. The rest of M consists of elements that are not related to the language at all. We will therefore restrict our consideration to the case where h is surjective.

DEFINITION 2.19
The syntactic monoid, $\Sigma^*/\equiv_L$ is a monoid whose elements are the congruence classes of Σ^* under $\equiv_L$, concatenation being defined as $[uv] = [u] \circ [v]$, with left and right identity $[\lambda]$.

Note that the operation in the quotient is well-defined, as usual, by the fact that it is a congruence.

EXAMPLE 2.20
Consider again the language O_1. This has a syntactic monoid which is isomorphic to the integers under addition.

The syntactic monoid has a certain universal property [32].

DEFINITION 2.21
The natural map from $\Sigma^* \to \Sigma^*/\equiv_L$, written η_L, is $u \to [u]_L$. This is a surjective monoid homomorphism.

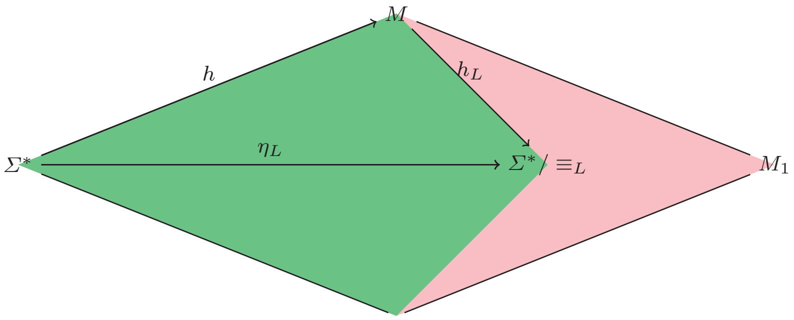

FIGURE 1. The syntactic congruence: the coarsest congruence that recognizes L. Fine congruences are on the left and on the far right is the coarsest congruence of all with only one congruence class; the quotient monoid is the trivial one element monoid M_1. Congruences that recognize L are shaded green. Congruences that do not are shaded red. If M is a monoid that recognizes L through the morphism h then there is a homomorphism from M to $\Sigma^*/\equiv_L$, h_L such that $\eta_L = h \cdot h_L$.

PROPOSITION 2.22
The syntactic monoid of L recognizes L through the morphism η_L.

PROOF. This is perhaps too obvious to merit a detailed proof; but if $u \in L$ and $u \equiv_L v$ then v is in L; so clearly $L = \bigcup\{[u]_L \mid u \in L\}$. ■

The syntactic monoid of L is the smallest (coarsest) monoid that recognizes L, and it is unique up to isomorphism.

We use the $\cdot$ symbol to represent function composition here.[4] If $f: A \to B$ and $g: B \to C$ then $f \cdot g: A \to C$.

PROPOSITION 2.23
Suppose L is recognized by the monoid M via the surjective morphism h then there is a morphism h_L from M to the syntactic monoid such that $\eta_L = h \cdot h_L$.

See the diagram in Figure 1 for an illustration of this.

PROOF. Define the congruence $\equiv_h$ to be $u \equiv_h v$ iff $h(u) = h(v)$. This is a congruence that saturates L. Define h_L to be $h_L(h(x)) = \eta_L(x)$. Since $\equiv_h$ saturates L, this is well-defined. Suppose x and y are such that $h(x) = h(y)$; then $x \equiv_h y$ which implies $x \equiv_L y$ which implies $\eta_L(x) = \eta_L(y)$. ■

Note that all of this follows quite directly from basic principles of universal algebra: this is an immediate consequence of Noether's third isomorphism theorem, namely that if we have two congruences of an algebra such that one is finer than the other, then there is a homomorphism from the quotient of the finer to the quotient of the coarser. We do not rely on this body of mathematics in this article but prove things directly each time. This leads to a little repetition and a lack of maximal generality, but makes the presentation more self-contained, concrete and comprehensible.

[4]We do this to avoid confusion with the more standard symbol $\circ$ which here we use merely as concatenation in monoids.

2.6 Context-free grammars

A CFG over an alphabet Σ is a tuple $\langle \Sigma, V, S, P\rangle$, where V is a finite set of non-terminals, $S \in V$ is the unique start symbol, P is a finite set of productions written $N \to \alpha$, where $N \in V$ and $\alpha \in (V \cup \Sigma)^*$. We define the derivations in the standard way: we define the one-step derivation relation by $\beta N \gamma \Rightarrow \beta\alpha\gamma$, when $N \to \alpha \in P$ and $\beta, \gamma \in (V \cup \Sigma)^*$. We use the notation $\alpha \overset{*}{\Rightarrow} \beta$ for a derivation using zero or more steps : the reflexive transitive closure of $\Rightarrow$.

Definition 2.24
For a CFG, and a non-terminal N we define $\mathcal{L}(G,N) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$ and $\mathcal{C}(G,N) = \{(l,r) \in \Sigma^* \times \Sigma^* \mid S \overset{*}{\Rightarrow} lNr\}$. We also extend $\mathcal{L}(G,\alpha)$ for $\alpha \in (V \cup \Sigma)^*$ in the usual way, with $\mathcal{L}(G,\lambda) = \{\lambda\}$, $\mathcal{L}(G,a) = \{a\}$ for $a \in \Sigma$ and $\mathcal{L}(G,\alpha\beta) = \mathcal{L}(G,\alpha)\mathcal{L}(G,\beta)$.

Note that by this definition $(\lambda,\lambda) \in \mathcal{C}(G,S)$ since $S \overset{*}{\Rightarrow} S = \lambda S \lambda$, even if S does not occur on the right-hand side of any production in G.

$L(G)$ is defined to be $\mathcal{L}(G,S)$, for the unique start symbol S. A context-free language is a language L_* such that there is a CFG G such that $L(G) = L_*$.

Proposition 2.25
If we have a grammar G that defines a language L then for any non-terminal N in G,

$$\mathcal{C}(G,N) \odot \mathcal{L}(G,N) \subseteq L.$$

We will give the trivial proof now.

Proof. If $(l,r) \in \mathcal{C}(G,N)$, then there is a derivation $S \overset{*}{\Rightarrow} lNr$. If $u \in \mathcal{L}(G,N)$, then there is a derivation $N \overset{*}{\Rightarrow} u$. There is therefore a derivation $S \overset{*}{\Rightarrow} lNr \overset{*}{\Rightarrow} lur$, and so $(l,r) \odot u = lur \in L$. ■

This proposition implies that a non-terminal in a CFG defines a decomposition or factorization of L into a set of contexts and a set of substrings.

Proposition 2.26
If there is a production $N \to \alpha$, then $\mathcal{L}(G,N) \supseteq \mathcal{L}(G,\alpha)$.

Proposition 2.27
More specifically, if the rule is of the form $N \to PQ$ then $\mathcal{L}(G,N) \supseteq \mathcal{L}(G,P)\mathcal{L}(G,Q)$. If the rule is of the form $N \to w$ then $\mathcal{L}(G,N) \supseteq \{w\}$.

Therefore we have two relations between the grammar and the language it describes. One is that the non-terminal determines a factorization into a set of contexts and a set of strings. The other is that a production determines an inclusion relation between the sets of strings that correspond to non-terminals. As we shall see later—and in exact analogy to the UA (see, e.g. [27])—we can reverse this process and define a grammar where the non-terminals are based on these decompositions—more precisely on the maximal decompositions, and the productions are the ones that satisfy the inclusion relations.

In what follows we will occasionally restrict ourselves to grammars where the productions are either of the form $N \to PQ$ or $N \to a$ for $a \in \Sigma$, or $N \to \lambda$; a restriction along the lines of Chomsky normal form.

3 The syntactic concept lattice

Given any fixed language L, we can define an object called the syntactic concept lattice [6, 9]. This is a canonical object in the sense that it is uniquely defined for each language.

There are several equivalent ways of defining this lattice; we will describe two. The first is based on the Galois connection between sets of contexts and sets of strings [13]; this is exactly the same as Formal Concept Analysis [18]. We consider some fixed language L in what follows. We define a *polar* map from sets of strings to sets of contexts. For a set of strings S, we define a set of contexts $S^{\triangleright}$ as follows:

$$S^{\triangleright} = \{(l,r) \in \Sigma^* \times \Sigma^* \mid \forall u \in S, lur \in L\}. \tag{1}$$

This is the shared distribution of the set of strings S. Dually, we define a polar map from a set of contexts C to the set of all strings that can occur in all of these contexts.

$$C^{\triangleleft} = \{u \in \Sigma^* \mid \forall (l,r) \in C, lur \in L\} \tag{2}$$

We now give some basic properties of these maps.

Proposition 3.1
If $S \subseteq T$ are two sets of strings, then $S^{\triangleright} \supseteq T^{\triangleright}$. If S is empty, then $S^{\triangleright} = \Sigma^* \times \Sigma^*$.

Proposition 3.2
If X, Y are sets of strings then $(X \cup Y)^{\triangleright} = X^{\triangleright} \cap Y^{\triangleright}$. Similarly if X_i, for $i \in I$ is a family of sets, finite or infinite, then

$$\left(\bigcup_{i \in I} X_i\right)^{\triangleright} = \bigcap_{i \in I} X_i^{\triangleright}.$$

Dually we have the following two propositions.

Proposition 3.3
If $C \subseteq D$ are two sets of contexts, then $C^{\triangleleft} \supseteq D^{\triangleleft}$.

Proposition 3.4
If C, D are sets of contexts then $(C \cup D)^{\triangleleft} = C^{\triangleleft} \cap D^{\triangleleft}$. Similarly if C_i, for $i \in I$ is a family of contexts, finite or infinite, then

$$\left(\bigcup_{i \in I} C_i\right)^{\triangleleft} = \bigcap_{i \in I} C_i^{\triangleleft}.$$

Proposition 3.5
For any set of strings S, $S \subseteq S^{\triangleright\triangleleft}$. For any set of contexts C, $C \subseteq C^{\triangleleft\triangleright}$.

Proof. Suppose $w \in S$. Then the set of contexts that w can occur in contains $S^{\triangleright}$. Therefore $w \in S^{\triangleright\triangleleft}$. Similarly if $(l,r) \in C$, the set of strings that can occur in (l,r) is a superset of $C^{\triangleleft}$ and so $(l,r) \in C^{\triangleleft\triangleright}$. ■

Proposition 3.6
For any set of strings S, $S^{\triangleright} = S^{\triangleright\triangleleft\triangleright}$.
For any set of contexts C, $C^{\triangleleft} = C^{\triangleleft\triangleright\triangleleft}$.

Proof. $(S^{\triangleright})^{\triangleright\triangleleft} \supseteq S^{\triangleright}$ by the previous lemma (putting $C = S^{\triangleright}$). But $S^{\triangleright\triangleleft} \supseteq S$, so $(S^{\triangleright\triangleleft})^{\triangleright} \subseteq S^{\triangleright}$ by Proposition 3.1. This completes the first proof; the second is exactly parallel. ■

DEFINITION 3.7
We say that a set of strings S is *closed* iff $S = S^{\rhd\lhd}$. Similarly, we say that a set of contexts C is *closed* iff $C = C^{\lhd\rhd}$.

PROPOSITION 3.8
For any set of strings S, $S^{\rhd}$ is closed, and so is $S^{\rhd\lhd}$. For any set of contexts C, $C^{\lhd}$ is closed, and so is $C^{\lhd\rhd}$.

PROPOSITION 3.9
L is always a closed set of strings.

$(\lambda,\lambda) \in L^{\rhd}$. Therefore if $w \in L^{\rhd\lhd}$, w must occur in the context (λ,λ) which implies that $w \in L$.

PROPOSITION 3.10
Suppose X and Y are sets of strings; then $(X \cup Y)^{\rhd\lhd} = (X^{\rhd\lhd} \cup Y^{\rhd\lhd})^{\rhd\lhd}$. More generally, if X_i is a family of sets of strings, then $\bigcup_i X_i^{\rhd\lhd} = (\bigcup_i X_i^{\rhd\lhd})^{\rhd\lhd}$.

PROOF. $(X^{\rhd\lhd} \cup Y^{\rhd\lhd})^{\rhd} = X^{\rhd\lhd\rhd} \cap Y^{\rhd\lhd\rhd} = X^{\rhd} \cap Y^{\rhd} = (X \cup Y)^{\rhd}$. Similarly $(\bigcup_i X_i^{\rhd\lhd})^{\rhd} = \bigcap_i X_i^{\rhd\lhd\rhd} = \bigcap_i X_i^{\rhd} = (\bigcup_i X_i)^{\rhd}$. ■

DEFINITION 3.11
A *concept* is an ordered pair $\langle S, C\rangle$ of a set of strings S and a set of contexts C such that $S^{\rhd} = C$ and $C^{\lhd} = S$.

Though it seems like these concepts might be hard to find, it is easy to prove the following proposition.

PROPOSITION 3.12
- For any set of strings S, $\langle S^{\rhd\lhd}, S^{\rhd}\rangle$ is a concept.
- For any set of contexts C, $\langle C^{\lhd}, C^{\lhd\rhd}\rangle$ is a concept.
- For any set of strings S we define $\mathcal{C}(S)$ to be the concept $\langle S^{\rhd\lhd}, S^{\rhd}\rangle$.

Indeed it is easy to see that if $\langle S, C\rangle$ is a concept, then S is a closed set of strings and C is a closed set of concepts. There is a bijection between the set of closed sets of strings and the set of closed sets of concepts; we can either consider the concepts as sets of strings, or sets of contexts, or both.

An alternative way of defining the concepts is as maximal decompositions of a language into sets of strings and contexts.

DEFINITION 3.13
A concept is an ordered pair $\langle S, C\rangle$ of a set of strings S and a set of contexts C such that $C \odot S \subseteq L$ and S and C are both maximal. That is to say, if $\langle T, D\rangle$ is such that $D \odot T \subseteq L$ and $T \supseteq S, D \supseteq C$ then $T = S$ and $C = D$.

It is easy to verify that these two definitions are equivalent; this second definition makes the links with the UA more explicit.

EXAMPLE 3.14
Suppose $L = \Sigma^*$. Then there is exactly one concept: $\langle \Sigma^*, \Sigma^* \times \Sigma^*\rangle$.

EXAMPLE 3.15
Suppose $L = \emptyset$. Then there are exactly two concepts: $\langle \Sigma^*, \emptyset\rangle$ and $\langle \emptyset, \Sigma^* \times \Sigma^*\rangle$.

Definition 3.16
Given a language L, we denote the set of concepts of L, by $\mathfrak{B}(L)$. We call this the *syntactic concept lattice* of L.

As we shall see this forms a very rich algebraic structure.

Definition 3.17
We define a partial order by $\langle S,C\rangle \leq \langle T,D\rangle$ iff $S \subseteq T$.

Definition 3.18
Suppose $\{\langle S_i,C_i\rangle \mid i \in I\}$ is a collection of concepts. We define $\bigvee\{\langle S_i,C_i\rangle \mid i\in I\} = \mathcal{C}(\bigcup_i S_i)$. For two concepts, we define $\langle S_x,C_x\rangle \vee \langle S_y,C_y\rangle = \mathcal{C}(S_x \cup S_y)$.

It is clear that this is a least upper bound. We use the notation $\mathfrak{B}(L)$ to emphasize the link to formal concept analysis where the fraktur *B* stands for *Begriff*. We now examine the relationship between the syntactic congruence and the syntactic concept lattice.

Proposition 3.19
$\{u\}^{\rhd\lhd} = \{v\}^{\rhd\lhd}$ iff $u \equiv_L v$.

Proposition 3.20
$[u] \subseteq \{u\}^{\rhd\lhd}$.

Proposition 3.21
If $w \in S$ for some closed set of words S, then $[w] \subseteq S$. In other words, closed sets of strings are unions of congruence classes: $S = \bigcup_{w\in S}[w]$.

Proposition 3.22
$\mathfrak{B}(L)$ has finitely many elements iff L is regular.

Proof. Note that a language L has finitely many congruence classes iff it is regular. By Proposition 3.19 this means that if L is not regular then $\mathfrak{B}(L)$ is infinite. By Proposition 3.21, if L is regular, then $\mathfrak{B}(L)$ is finite. ■

Figure 2 contains a diagram which shows the syntactic concept lattice of the regular language $(ab)^*$. This consists of 7 concepts, which we arrange in a Hasse diagram according to the natural partial order.

Figure 3 contains a similar diagram for a non-regular language, which of course has an infinite number of concepts.

Whereas the syntactic monoid is always countable; and infinite iff the language is regular, the lattice may have uncountably many elements, even for a context-free language; of course the cardinality of the sets can be at most $2^{\aleph_0}$, if Σ is finite.

Proposition 3.23
There is a context-free language L such that $\mathfrak{B}(L)$ has uncountably many elements. (Ryo Yoshinaka, p.c.)

Proof. Let $O_1^c = \{w \in \{a,b\}^* \mid |w|_a \neq |w|_b\}$. Here $|w|_a$ means the number of occurrences in the string w of the symbol a. Define $c(w) = |w|_a - |w|_b$, and $c(l,r) = -c(lr)$. We can see that $u \equiv_L v$ iff $c(u) = c(v)$. For each $X \subseteq \mathbb{Z}$, define the concept $C(X) = \langle \{w \mid c(w) \in X\}, \{(l,r) \mid c(l,r) \notin X\}\rangle$. It is easy to verify that these are concepts, and therefore $\mathfrak{B}(L)$ is in bijection with $2^{\mathbb{Z}}$. ■

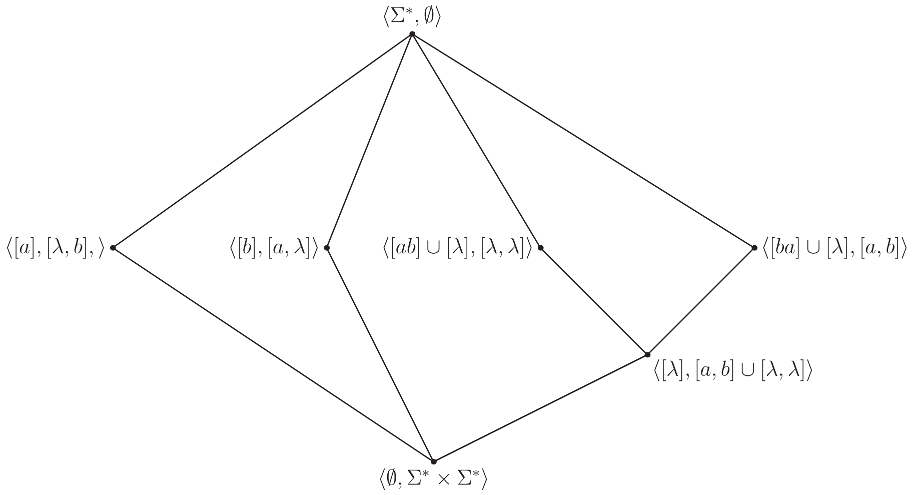
$\langle \Sigma^*, \emptyset \rangle$
$\langle [a], [\lambda, b], \rangle$
$\langle [b], [a, \lambda] \rangle$
$\langle [ab] \cup [\lambda], [\lambda, \lambda] \rangle$
$\langle [ba] \cup [\lambda], [a, b] \rangle$
$\langle [\lambda], [a, b] \cup [\lambda, \lambda] \rangle$
$\langle \emptyset, \Sigma^* \times \Sigma^* \rangle$

FIGURE 2. The syntactic concept lattice of the regular language $(ab)^*$.

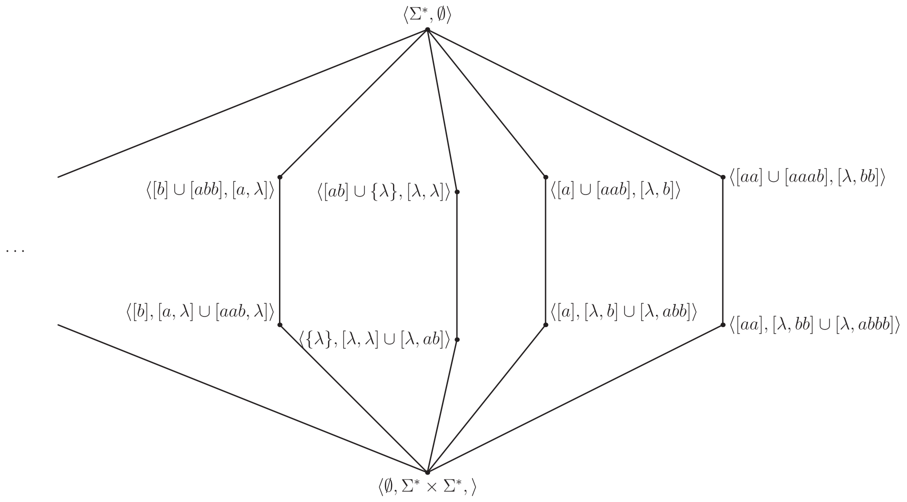
$\langle \Sigma^*, \emptyset \rangle$
$\langle [b] \cup [abb], [a, \lambda] \rangle$
$\langle [ab] \cup \{\lambda\}, [\lambda, \lambda] \rangle$
$\langle [a] \cup [aab], [\lambda, b] \rangle$
$\langle [aa] \cup [aaab], [\lambda, bb] \rangle$
...
$\langle [b], [a, \lambda] \cup [aab, \lambda] \rangle$
$\langle \{\lambda\}, [\lambda, \lambda] \cup [\lambda, ab] \rangle$
$\langle [a], [\lambda, b] \cup [\lambda, abb] \rangle$
$\langle [aa], [\lambda, bb] \cup [\lambda, abbb] \rangle$
$\langle \emptyset, \Sigma^* \times \Sigma^*, \rangle$

FIGURE 3. A fragment of the syntactic concept lattice of the context-free language $\{a^n b^n \mid n \geq 0\}$. The lattice is finite so we just show the components generated by the short strings. The remainder of the lattice is structurally similar to the fragment shown here and contains concept containing strings like a^n, b^n and so on, where $n > 2$.

Note that in this example, the syntactic monoid of O_1^c is isomorphic to $\mathbb{Z}$ under addition.[5]

3.1 Concatenation

We now define the natural concatenation operation and its associated residuals.

[5]Note that this implies the existence of an interesting language class: the class of languages with a countable concept lattice.

DEFINITION 3.24
For two closed sets of strings R, T define $R \circ T = (RT)^{\rhd\lhd}$. For two concepts, $\langle S_R, C_R\rangle \circ \langle S_T, C_T\rangle = \langle (S_R S_T)^{\rhd\lhd}, (S_R S_T)^{\rhd}\rangle = \mathcal{C}(S_R S_T)$.

DEFINITION 3.25
Given a language L, we define the unit in the lattice to be $\lambda_L = \langle \{\lambda\}^{\rhd\lhd}, \{\lambda\}^{\rhd}\rangle$.

The closure operation interacts nicely with the concatenation operation. Just as distributional equivalence ($\equiv_L$) is a congruence of the monoid, implying that $[uv] \supseteq [u][v]$, so we have a related result for the lattice operations, lifted to sets of strings. Indeed as we shall see later, this defines a congruence on sets of strings—a congruence on $\mathcal{P}(\Sigma^*)$.

PROPOSITION 3.26
Suppose we have two sets of strings Y, Z. Then,

$$(YZ)^{\rhd\lhd} \supseteq Y^{\rhd\lhd} Z^{\rhd\lhd}.$$

PROOF. [17, 25] Suppose $y \in Y^{\rhd\lhd}$ and $z \in Z^{\rhd\lhd}$. So $\{y\}^{\rhd} \supseteq Y^{\rhd}$ and $\{z\}^{\rhd} \supseteq Z^{\rhd}$. Take $(l,r) \in (YZ)^{\rhd}$. So $lYZr \subseteq L$. This means that $(l, Zr) \subseteq Y^{\rhd}$. So $(l, Zr) \subseteq Y^{\rhd} \subseteq \{y\}^{\rhd}$. So $lyZr \subseteq L$. This means that $(ly, r) \subseteq Z^{\rhd} \subseteq \{z\}^{\rhd}$. so $lyzr \in L$. So $(l,r) \in (yz)^{\rhd}$ and so $(l,r) \in (Y^{\rhd\lhd} Z^{\rhd\lhd})^{\rhd}$. Therefore $(YZ)^{\rhd} \subseteq (Y^{\rhd\lhd} Z^{\rhd\lhd})^{\rhd}$. Therefore $(YZ)^{\rhd\lhd} \supseteq (Y^{\rhd\lhd} Z^{\rhd\lhd})^{\rhd\lhd} \supseteq Y^{\rhd\lhd} Z^{\rhd\lhd}$. ■

If $Y = \{u\}$ and $Z = \{v\}$ then this means that $\{uv\}^{\rhd\lhd} \supseteq \{u\}^{\rhd\lhd}\{v\}^{\rhd\lhd}$; contrast this with the result in Proposition 2.11.

PROPOSITION 3.27
Suppose we have sets of strings X, Y, Z such that $X \supseteq YZ$. Then $X^{\rhd\lhd} \supseteq Y^{\rhd\lhd} Z^{\rhd\lhd}$.

PROPOSITION 3.28
For any sets of strings X, Y, $(XY)^{\rhd\lhd} = (X^{\rhd\lhd} Y^{\rhd\lhd})^{\rhd\lhd}$.

PROOF. By Proposition 3.26, $XY^{\rhd\lhd} \supseteq X^{\rhd\lhd} Y^{\rhd\lhd}$. So $XY^{\rhd\lhd} = (XY^{\rhd\lhd})^{\rhd\lhd} \supseteq (X^{\rhd\lhd} Y^{\rhd\lhd})^{\rhd\lhd}$. But $XY \subseteq X^{\rhd\lhd} Y^{\rhd\lhd}$, so $XY^{\rhd\lhd} \subseteq (X^{\rhd\lhd} Y^{\rhd\lhd})^{\rhd\lhd}$. ■

PROPOSITION 3.29
$\langle \mathfrak{B}(L), \circ, \lambda_L\rangle$ is a monoid.

PROOF. We can verify that the operation $\circ$ is associative easily using the previous lemmas. Given three closed sets of strings X, Y, Z, we can see that

$$((XY)^{\rhd\lhd} Z)^{\rhd\lhd} = (XYZ)^{\rhd\lhd} = (X(YZ)^{\rhd\lhd})^{\rhd\lhd}.$$

Similarly since $X = X\{\lambda\}$,

$$X^{\rhd\lhd} = (X\{\lambda\}^{\rhd\lhd})^{\rhd\lhd} = (\{\lambda\}^{\rhd\lhd} X)^{\rhd\lhd},$$

which establishes that λ_L is the identity. ■

PROPOSITION 3.30
$\mathfrak{B}(L)$ is a complete idempotent semiring.

PROOF. We define the bottom element as $\bot = \mathcal{C}(\emptyset)$; $\circ$ and $\vee$ are defined as before. In order to verify that it is a CIS, we need to check the axioms. By Proposition 3.29, we have the relevant monoid structure. By the definition of $\vee$ it is associative, commutative and idempotent. If X is closed then

by Proposition 3.10, $X = (X \cup \emptyset^{\rhd\lhd})^{\rhd\lhd}$ so $\bot$ is a unit with respect to $\vee$. The only remaining point to check that $X \circ \bigvee_i Y_i = \bigvee_i (X \circ Y_i)$. Suppose R, S_i are closed sets of strings; then $R(\bigcup_i S_i) = \bigcup_i RS_i$.
$(R(\bigcup_i S_i)^{\rhd\lhd}))^{\rhd\lhd} = (R(\bigcup_i S_i))^{\rhd\lhd} = (\bigcup_i RS_i)^{\rhd\lhd} = (\bigcup_i (RS_i)^{\rhd\lhd})^{\rhd\lhd}$.

■

Proposition 3.31
By Proposition 2.6, $\mathfrak{B}(L)$ is a residuated lattice.

We can now give a complete definition of all of the operations in this residuated lattice using only closed sets of strings.

Definition 3.32 [25]
The structure consisting of $\{X \subseteq \Sigma^* \mid X = X^{\rhd\lhd}\}$. with constants $\bot = \emptyset^{\rhd\lhd}, \top = \Sigma^*, 1 = \lambda^{\rhd\lhd}$ and partial order $X \leq Y$ iff $X \subseteq Y$, operations $X \vee Y = (X \cup Y)^{\rhd\lhd}$, $X \wedge Y = X \cap Y$, and $X \circ Y = (XY)^{\rhd\lhd}$, $X/Y = \{z \mid zY \subseteq X\}$, $X \backslash Y = \{z \mid Xz \subseteq Y\}$ is a residuated lattice.

In what follows we will largely take the concepts to be just closed sets of strings, implicitly using the mapping $\langle S, C\rangle \to S$ where appropriate.

4 Universal property

Some rhetoric first—we want a system of syntactic categories to interpret our grammars [19]. We have at a minimum two operations—concatenation and union—if we restrict ourselves to CFGs. One such system might be $\mathcal{P}(\Sigma^*)$, but that is in a sense too complex. For each language, we want the simplest such system: an algebraic structure that contains enough categories to represent the non-terminals but no more. In this section, we will show that for any language there is a single simplest system of categories, unique up to isomorphism, which is as we shall see the syntactic concept lattice; we provide a formal statement of this in Proposition 6.4.

Definition 4.1
Given a CIS, S, we say that an equivalence relation $\equiv$ on the elements of S is a CIS-congruence if for all $X, Y, Z \in S$:

– $X \equiv Y$ implies $X \circ Z \equiv Y \circ Z$ and $Z \circ X \equiv Z \circ Y$.
– $X \equiv Y$ implies $X \vee Z \equiv Y \vee Z$.
– For any collection of sets X_i, Y_i such that $X_i \equiv Y_i$, $\bigvee_i X_i \equiv \bigvee_i Y_i$.

Definition 4.2
Given two CIS, A and B a function $h : A \to B$ is a CIS-morphism iff

– $h(x \vee y) = h(x) \vee h(y)$.
– For any collection x_i, $h(\bigvee_i x_i) = \bigvee_i h(x_i)$.
– $h(x \circ y) = h(x) \circ h(y)$.

Proposition 4.3
If h is a surjective CIS-morphism from A to B, then $h(1_A) = 1_B$ and $h(\bot_A) = \bot_B$.

Proof. $h(a) = h(1_A \circ a) = h(1_A) \circ h(a)$ and $h(a) = h(a \circ 1_A) = h(a) \circ h(1_A)$. So $1_B = h(1_A) \circ 1_B = h(1_A)$. $h(a) = h(a \vee \bot_A) = h(a) \vee h(\bot_A)$. So $h(\bot_A) = h(\bot_A) \vee \bot_B = \bot_B$. ■

Proposition 4.4
Suppose $x \leq y$; then $h(x) \leq h(y)$.

PROOF. Note that $x \leq y$ iff $x = x \vee y$. If this is true then $h(x) = h(x \vee y) = h(x) \vee h(y)$ implies $h(x) \leq h(y)$. ■

We now define the residual map; informally this is a map which is analogous to the inverse.

DEFINITION 4.5
If we have a CIS-morphism h, from CIS A to B, we define h^* to be the map from B to A defined as $h^*(y) = \bigvee\{x \in A : h(x) \leq y\}$.

Now h and h^* form what is called a residuated pair.

PROPOSITION 4.6
For all $x \in A, y \in B$, $h(x) \leq y$ iff $x \leq h^*(y)$.

PROOF. Suppose $h(x) \leq y$. $h^*(y) = \bigvee_{z \in A : h(z) \leq y} z$. But since $h(x) \leq y$ this means that one of these z will be x, so $h^*(y) \geq x$. Conversely suppose $x \leq h^*(y)$; then $h(x) \leq h(h^*(y)) = h(\bigvee_{z \in A : h(z) \leq y} z)$ $= \bigvee_{z \in A : h(z) \leq y} h(z) \leq y$. Therefore $h(x) \leq y$. ■

PROPOSITION 4.7
$h(h^*(y)) \leq y$ and $h^*(h(x)) \geq x$.

PROOF. $h^*(y) \leq h^*(y)$ so $h(h^*(y)) \leq y$. $h(x) \leq h(x)$ so $x \leq h^*(h(y))$. ■

PROPOSITION 4.8
Both h and h^* are monotonic.

Note that if A is $\mathcal{P}(\Sigma^*)$ then the homomorphism is uniquely defined just by the values of the elements of Σ; once we know what $h(\{a\})$ is for each element $a \in \Sigma$ then this fixes the value for every string and thus every set of strings. This is rather like the way a linear map (a homomorphism of vector spaces) is fixed by the value of the function at the elements of a basis.

We are concerned exclusively with homomorphisms that arise from congruences of $\mathcal{P}(\Sigma^*)$ and are therefore all surjective. In this case we can slightly strengthen the results.

PROPOSITION 4.9
If h is a surjective CIS-homomorphism then $h(h^*(y)) = y$.

PROOF. By definition $h^*(y) = \bigvee\{x \in A : h(x) \leq y\}$. Consider $\{x \in A : h(x) = y\}$. Since h is surjective, this is non-empty. Therefore $\bigvee\{x \in A : h(x) \leq y\} = \bigvee\{x \in A : h(x) = y\}$ by monotonicity of h. So $h(h^*(y)) =$ $h(\bigvee\{x \in A : h(x) = y\}) = \bigvee\{y\} = y$. ■

PROPOSITION 4.10
If $h : A \rightarrow B$ is a surjective CIS-homomorphism then for any $X, Y \in B$, $h^*(X \circ Y) \geq h^*(X) \circ h^*(Y)$.

PROOF. Consider $h(h^*(X) \circ h^*(Y)) = h(h^*(X)) \circ h(h^*(Y)) = X \circ Y$. So $h^*(X) \circ h^*(Y) \in \{z \in A \mid h(z) \leq X \circ Y\}$. Now $h^*(X \circ Y) = \bigvee\{z \in A \mid h(z) \leq X \circ Y\} \geq h^*(X) \circ h^*(Y)$. ■

We now define the idea of a CIS recognizing a language which differs in some respects to the earlier definition of a monoid recognizing a language.

DEFINITION 4.11
We say that a CIS B recognizes L through the morphism h if h is a surjective morphism from $\mathcal{P}(\Sigma^*) \rightarrow B$ such that $h^*(h(L)) = L$.

EXAMPLE 4.12
$\mathcal{P}(\Sigma^*)$ recognizes L through the identity morphism.

Recall that this morphism is fixed just by the elements of $h(a)$ for $a \in \Sigma$. So if we have a morphism from $\mathcal{P}(\Sigma^*) \to B$ then $h(L)$ is the unique element that might represent L. Therefore, $h^*(h(L)) \geq L$ by the previous proposition. The condition thus just states that this element does in fact correspond to L and not some other larger set of strings. For example if B is the single element CIS, then $h^*(h(L)) = \Sigma^*$.

Having established these general properties of CIS-morphisms, and congruences we can now consider the specific examples we are interested in.

Definition 4.13 [34]
Given a language L we define an equivalence relation on sets of strings, i.e. on $\mathcal{P}(\Sigma^*)$, where the set of strings X is congruent to the set of strings Y, written $X \cong_L Y$ iff $X^{\rhd} = Y^{\rhd}$.

This congruence is on sets of strings—the syntactic congruence is a congruence of strings. They coincide on singleton sets of strings.

Proposition 4.14
The relation $\cong_L$ is a CIS-congruence.

Proof. For all sets of strings X, Y, W, Z.

- If $X^{\rhd} = Y^{\rhd}$ and $W^{\rhd} = Z^{\rhd}$ then by Proposition 3.28, $XW^{\rhd\lhd} = (X^{\rhd\lhd}W^{\rhd\lhd})^{\rhd\lhd} = (Y^{\rhd\lhd}Z^{\rhd\lhd})^{\rhd\lhd} = YZ^{\rhd\lhd}$. So $(XW)^{\rhd} = (YZ)^{\rhd}$.
- If $X^{\rhd} = Y^{\rhd}$ and $W^{\rhd} = Z^{\rhd}$ then $(X \cup W)^{\rhd} = X^{\rhd} \cap W^{\rhd} = Y^{\rhd} \cap Z^{\rhd} = (Y \cup Z)^{\rhd}$. (or by Proposition 3.10).
- Suppose X_i, Y_i are families of sets of strings indexed by I such that $X_i \cong_L Y_i$ for all $i \in I$. Then $(\bigvee_i X_i)^{\rhd} = \bigcap_i X_i^{\rhd} = \bigcap_i Y_i^{\rhd} = (\bigvee_i Y_i)^{\rhd}$.

■

Proposition 4.15
For any language L, $\mathfrak{B}(L)$ is CIS-isomorphic to $\mathcal{P}(\Sigma^*)/\cong_L$.

Proof. We define the natural function $\pi : \mathfrak{B}(L) \to \mathcal{P}(\Sigma^*)/\cong_L$, given by $\pi(X) = [X]_{\cong_L}$. We can verify that this is a bijection and that the operations of $\circ$ and $\vee$ are the same. ■

So the SCL is the quotient of $\mathcal{P}(\Sigma^*)$ under the congruence $X \cong Y$ iff $X^{\rhd\lhd} = Y^{\rhd\lhd}$ iff $X^{\rhd} = Y^{\rhd}$. The relation to the syntactic congruence thus becomes even clearer: the syntactic monoid is the quotient of Σ^* under the congruence $u \equiv v$ iff $\{u\}^{\rhd} = \{v\}^{\rhd}$.

There is a potential confusion here between sets of strings and sets of strings. The relation $\cong_L$ is a relation between sets of strings: $X \cong_L Y$, where X, Y are sets of strings. The equivalence class $[X]_{\cong_L}$ is thus a set of sets of strings; a collection whose elements are sets of strings that are congruent to X. Now it so happens that if Y, Z are two sets of strings in $[X]_{\cong_L}$ then so is $Y \cup Z$ (since it is a congruence wrt $\vee$), as is indeed the union of all the elements of $[X]_{\cong_L}$.

Proposition 4.16
Let $[X]_{\cong_L}$ be the set of all sets of strings congruent to X. Then $\bigcup\{Y \mid Y \cong_L X\} = X^{\rhd\lhd} \cong_L X$. Or, $\bigcup\{Y \mid Y \cong_L X\} \in [X]_{\cong_L}$.

$[X]_{\cong_L}$ is a set of sets of strings. The union of all the sets in $[X]_{\cong_L}$ is a set of strings $\bigcup_{Y \in [X]} Y$ which is equal to $X^{\rhd\lhd}$ which happens to be in $[X]$. So $\cong$ is a partition of the set of all sets of strings. A given set of strings will be in only one concept—its closure. A given string u may be in a variety of closed sets of strings, some large and some small—the largest will be Σ^* and the smallest will be $\{u\}^{\rhd\lhd}$.

We can now start to state the universal property of the syntactic concept lattice.

PROPOSITION 4.17
The natural map $\zeta_L : \mathcal{P}(\Sigma^*) \to \mathfrak{B}(L)$ given by $\zeta(X) = X^{\rhd\lhd}$ is a CIS-homomorphism.

PROPOSITION 4.18
The residual of this is $\zeta_L^* : \mathfrak{B}(L) \to \mathcal{P}(\Sigma^*)$ which is just the identity on sets of strings; the natural embedding of the closed sets of strings in $\mathcal{P}(\Sigma^*)$; $\zeta_L^*(\langle S, C\rangle) = S$.

We will now show that this is the coarsest congruence that recognizes L. If it is any coarser (i.e. fewer larger classes) then it will not have L in it. Therefore this is the unique smallest structure that we can use to interpret this language.

PROPOSITION 4.19
$\mathfrak{B}(L)$ recognizes L, through the morphism $X \to X^{\rhd\lhd}$.

PROOF. Immediate; since $L^{\rhd\lhd} = L$, so $h(L) = L$ and therefore $h^*(h(L)) = L$. ■

We now prove an important technical lemma; informally it says that any lattice that recognizes this must be 'finer' (larger, more elaborate) than the syntactic context lattice.

LEMMA 4.20
Suppose, A is some CIS that recognizes L through a morphism h. Suppose there are two sets of strings X, Y such that $h(X) = h(Y)$; then $\zeta_L(X) = \zeta_L(Y)$. (i.e. $X^{\rhd} = Y^{\rhd}$)

PROOF. Suppose $(l, r) \in X^{\rhd}$. That means that $lXr \subseteq L$; so $h(lXr) \leq h(L)$ since h is monotonic. This implies that $h(\{l\}) \circ h(X) \circ h(\{r\}) \leq h(L)$ using the fact that h is a homomorphism. So $h(\{l\}) \circ h(Y) \circ h(\{r\}) \leq h(L)$ since $h(X) = h(Y)$. Therefore $h(\{l\}Y\{r\}) \leq h(L)$ since it is a homomorphism. So $h^*(h(\{l\}Y\{r\})) \leq h^*(h(L))$ since h^* is monotonic. Now $h^*(h(L)) = L$ since A recognizes L So using the fact that $h^*(h(X)) \geq X$ we have $\{l\}Y\{r\} \leq h^*(h(\{l\}Y\{r\})) \leq h^*(h(L)) = L$. So $(l, r) \in Y^{\rhd}$. The converse argument is identical and so $X^{\rhd} = Y^{\rhd}$. ■

See Figure 4 for a diagram that illustrates this. The following proposition says that the SCL is (isomorphic to) the smallest CIS that recognizes L; it is the terminal element of the appropriate category.

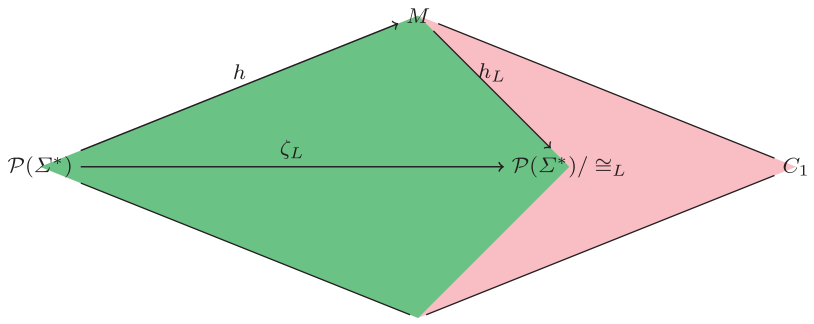

FIGURE 4. The syntactic concept lattice: the coarsest congruence of $\mathcal{P}(\Sigma^*)$ that recognizes L. Fine congruences are on the left and on the far right is the coarsest congruence of all with only one congruence class. Congruences that recognize L are shaded green. Congruences that do not are shaded red. C_1 is the trivial one element CIS. If M is a CIS that recognizes L through the morphism h then there is a homomorphism from M to $\mathcal{P}(\Sigma^*)/\cong_L$, h_L such that $\zeta_L = h \cdot h_L$.

PROPOSITION 4.21
If we have a lattice B that recognizes L through a morphism h, then there is a morphism h_L from B to $\mathfrak{B}(L)$ such that $\zeta_L = h \circ h_L$. Therefore the smallest CIS that recognizes L is unique up to isomorphism.

PROOF. Define $h_L(h(X)) = \zeta_L(X)$. This is well-defined by Lemma 4.20. We can verify easily that it is a morphism. ■

This is again a consequence of the isomorphism theorems.

5 CFG morphisms

We can now make clear the relation between these morphisms between semirings, and representations such as CFGs. The semirings, on our view, are merely a way of talking about CFGs and their morphisms in a more algebraic way; a method to talk about the 'semantics' of CFGs.[6] We now make the link to CFGs; and in particular we will link the semantic notion of a morphism—a homomorphism between CISs—with the syntactic notion—a morphism between grammars.

We start by defining some standard notions of mappings between CFGs; the definitions are straightforward extensions of the idea of morphisms between finite automata [26]; we will give them in full since we have not so far been able to find them in the literature.

DEFINITION 5.1
Assume we have two CFGs, $G_1 = \langle \Sigma, V_1, S_1, P_1 \rangle$ and $G_2 = \langle \Sigma, V_2, S_2, P_2 \rangle$, defined over the same terminal alphabet Σ. G_1 is a subgrammar of G_2 iff $V_1 \subseteq V_2$, $P_1 \subseteq P_2$ and $S_1 = S_2$.

PROPOSITION 5.2
If G_1 is a subgrammar of G_2, then $L(G_1) \subseteq L(G_2)$.

DEFINITION 5.3
Assume we have two CFGs, $G_1 = \langle \Sigma, V_1, S_1, P_1 \rangle$ and $G_2 = \langle \Sigma, V_2, S_2, P_2 \rangle$, defined over the same terminal alphabet Σ. Let ϕ be a function from V_1 to V_2; we extend it to a function from $V_1 \cup \Sigma$ to $V_2 \cup \Sigma$ by setting $\phi(a) = a$ for all $a \in \Sigma$, and thence to a function from $(V_1 \cup \Sigma)^*$ by saying that $\phi(\alpha\beta) = \phi(\alpha)\phi(\beta)$ and $\phi(\lambda) = \lambda$. For a production $N \to \alpha$ we say that $\phi(N \to \alpha) = \phi(N) \to \phi(\alpha)$. A *morphism* ϕ from G_1 to G_2 is such a function from V_1 to V_2, such that $\phi(S_1) = \phi(S_2)$, and $\phi(P_1) \subseteq P_2$.

EXAMPLE 5.4
Suppose G_1 has the three non-terminals S, A, B and productions $S \to AB, S \to ASB$ and $A \to a, B \to b$. $\mathcal{L}(G_1) = \{a^n b^n | n > 0\}$. Suppose G_2 has two non-terminals S, X and productions $S \to XX, S \to XSX$ and $X \to a, X \to b$. Then ϕ defined as $\phi(A) = X,' \phi(B) = X, \phi(S) = S$ is a morphism. $\phi(A \to a) = X \to a$, $\phi(B \to b) = X \to b$, and so on.

PROPOSITION 5.5
If there is a morphism from G_1 to G_2 then $L(G_1) \subseteq L(G_2)$.

We can state a more general result as follows.

[6] In this context the syntax/semantics distinction refers to the distinction between the rules of the grammar, and the language it defines, and has nothing to do with the meaning of a sentence or utterance [28].

PROPOSITION 5.6
If there is a morphism from G_1 to G_2 then for all non-terminals N in G_1:

- $\mathcal{L}(G_1,N) \subseteq \mathcal{L}(G_2,\phi(N))$
- $\mathcal{C}(G_1,N) \subseteq \mathcal{C}(G_2,\phi(N))$

PROOF. Take a derivation step $\beta N\gamma \Rightarrow_{G_1} \beta\alpha\gamma$ for some production $N \to \alpha \in P_1$. Then, since there is a morphism, ϕ, there is a production $\phi(N) \to \phi(\alpha)$ in P_2 and thus $\phi(\beta)\phi(N)\phi(\gamma) \Rightarrow_{G_2} \phi(\beta)\phi(\alpha)\phi(\gamma)$.

Suppose $N \Rightarrow \alpha_1 \Rightarrow \cdots \alpha_n \Rightarrow u$ is a derivation in G_1. Then $\phi(N) \Rightarrow \phi(\alpha_1) \Rightarrow \cdots \phi(\alpha_n) \Rightarrow \phi(u) = u$ is a derivation of u from $\phi(N)$ wrt G_2.

Suppose $S \Rightarrow \alpha_1 \Rightarrow \cdots \alpha_n \Rightarrow lNr$ is a derivation in G_1. Then $\phi(S) \Rightarrow \phi(\alpha_1) \Rightarrow \cdots \phi(\alpha_n) \Rightarrow \phi(lNr) = l\phi(N)r$ is a derivation in G_2. ■

Note that there is not the usual duality between contexts and substrings here—rather both sets get bigger (or at least no smaller) when we go from G_1 to G_2.

COROLLARY 5.7
If there is a morphism from G_1 to G_2 then for all non-terminals N in G_2:

- $\bigcup_{M:\phi(M)=N} \mathcal{L}(G_1,M) \subseteq \mathcal{L}(G_2,N)$
- $\bigcup_{M:\phi(M)=N} \mathcal{C}(G_1,M) \subseteq \mathcal{C}(G_2,N)$

DEFINITION 5.8
The image of G_1, $\phi(G_1) = \langle \Sigma, \phi(V_1), S_2, \phi(P_1) \rangle$ is a CFG, also which is called the morphic image of G_1 under the morphism ϕ. This will be a subgrammar of G_2.

DEFINITION 5.9
Given a CFG $G = \langle \Sigma, V, S, P \rangle$, and a function ψ from V into some arbitrary set X, we can define the morphic image of G to be the grammar $\psi(G) = \langle \Sigma, \psi(V), \psi(S), \psi(P) \rangle$. Clearly ψ is a morphism from G to $\psi(G)$.

A brief discussion: if ψ is injective—i.e. it is not the case that two distinct non-terminals are mapped to the same symbol, then this is just a relabelling of G and it defines the same language. The more interesting case is where we have two or more non-terminals that are mapped to the same symbol: two distinct non-terminals M,N such that $\psi(M) = \psi(N)$. The new grammar then will be strictly smaller than the original grammar. In this case it may be that the new grammar will define a larger language. To take an extreme case, we can map any grammar in CNF to a grammar with one non-terminal which will normally define the language Σ^* or Σ^+. This again is not very interesting—what we are most concerned with is the case when the resulting grammar, though smaller, defines the same language.

If there is a bijective morphism between two grammars, then we say they are isomorphic—they are identical up to a relabelling of non-terminals. We now consider the case where the morphic image of the grammar defines the same language.

DEFINITION 5.10
A morphism from G_1 to G_2 is called *exact* if $L(G_1) = L(G_2)$.

DEFINITION 5.11
For a CFG G_1 if there is a grammar G_2 such that there is an exact morphism $\phi : G_1 \to G_2$, (i.e. such that $L(G_1) = L(G_2)$), and there are two non-terminals M,N such that $\phi(M) = \phi(N)$, then we say that M and N are mergeable.

Definition 5.12
We say that a CFG is *minimal* if it does not have any mergeable non-terminals.

Note that this definition does not mean that this is the globally smallest grammar for the language $L(G)$. It might even have some redundant states that could be removed. Clearly if a grammar has two non-terminals that are mergeable then we can merge them and get another grammar which defines the same language but is strictly smaller. Suppose we are interested in minimal CFGs; then we want to find CFGs that do not have mergeable states; because if we do have mergeable states then the grammar is not minimal.

6 The universal morphism

We can now join up the (syntactic) CFG-morphisms and the (semantic) CIS-homomorphisms through the following lemma; any CIS that recognizes a language induces a morphism of the grammar where we map two non-terminals together if they 'mean' the same thing. If the CIS recognizes L then we will show that the morphism is exact.

Lemma 6.1
Suppose B is a CIS and h a surjective CIS homomorphism from $\mathcal{P}(\Sigma^*) \to B$. Then let G be a context-free grammar. Let ϕ be the CFG-morphism given by $\phi(N) = h(\mathcal{L}(G,N))$. Then for all non-terminals N in G,

$$\mathcal{L}(\phi(G), \phi(N)) \subseteq h^*(\phi(N)).$$

Proof. We will assume for clarity of exposition that the grammar is in CNF—the proof of the more general case is straightforward but less legible.

Suppose we have a rule $N \to PQ$ in G. Therefore $\mathcal{L}(G,N) \supseteq \mathcal{L}(G,P)\mathcal{L}(G,Q)$. So $h(\mathcal{L}(G,N)) \geq h(\mathcal{L}(G,P)) \circ h(\mathcal{L}(G,Q))$.

Then $h^*(h(\mathcal{L}(G,N))) \geq h^*(h(\mathcal{L}(G,P)) \circ h(\mathcal{L}(G,Q)))$ by monotonicity of h^*. By Proposition 4.10 $h^*(h(\mathcal{L}(G,N))) \geq h^*(h(\mathcal{L}(G,P))) \circ h^*(h(\mathcal{L}(G,Q)))$. Similarly suppose we have a rule $N \to a$ in G. So $h(\mathcal{L}(G,N)) \geq h(\{a\})$, and $h^*(h((\mathcal{L}(G,N))) \geq h^*(h(\{a\}))$.

Now suppose that $N' \overset{*}{\Rightarrow}_{\phi(G)} w$; we want to show that $w \in h^*(N')$. We proceed by induction on the length of the derivation. Base case: the production is of length 1. Suppose $N' \to a$ is a production in $\phi(G)$. Then for any production $N \to a$ in G such that $N' = \phi(N)$, $h^*(h((\mathcal{L}(G,N))) \geq h^*(h(\{a\})) \geq \{a\}$. Suppose it is true for all derivations of length at most k, and let $N' \overset{*}{\Rightarrow}_{\phi(G)} w$ be a derivation of length $k+1$ that starts with $N' \to P'Q' \to uv = w$, where $P' \overset{*}{\Rightarrow}_{\phi(G)} u$ and $Q' \overset{*}{\Rightarrow}_{\phi(G)} v$. These last two derivations must be of length at most k.

Then consider any production $N \to PQ$ in G such that $\phi(N) = N'$, $\phi(P) = P'$ and $\phi(Q) = Q'$ (of which there must be at least one). Now by the inductive hypothesis, $u \in h^*(P')$ and $v \in h^*(Q')$. So $uv \in h^*(P')h^*(Q') = h^*(h(\mathcal{L}(G,P))) \circ h^*(h(\mathcal{L}(G,Q)))$. Now since $h^*(h(\mathcal{L}(G,N))) \geq h^*(h(\mathcal{L}(G,P))) \circ h^*(h(\mathcal{L}(G,Q)))$ this means that $uv \in h^*(h(\mathcal{L}(G,N)))$. So $w \in h^*(\phi(N)$. By induction, it is therefore true for derivations of any length. ■

Clearly this implies the obvious fact that if h is the identity (and therefore so is h^*) this gives an exact morphism; the morphism which merges non-terminals which generate the same sets of strings.

As a simple corollary we have now the following lemma, which explains our definition of a CIS recognizing a language through a morphism.

PROPOSITION 6.2
Let G be a CFG that defines the language L and B a CIS that recognizes L through the morphism h. Then the morphism induced by map $\phi : N \to h(\mathcal{L}(G,N))$ is an *exact* CFG-morphism from G to $\phi(G)$.

PROOF. Since it is a morphism we know that $\mathcal{L}(\phi(G)) \supseteq \mathcal{L}(G)$. Since B recognizes L, $h^*(h(L)) = L$, so $\mathcal{L}(\phi(G), \phi(S)) \subseteq h^*(\phi(S)) = L$. ■

Note that this is only an implication: we can have a homomorphism that does not recognize the language, but where the induced grammar morphism is still exact, i.e. defines the same language. This could occur, e.g. when we merge two non-terminals M, N, where $\mathcal{C}(G,M) \odot \mathcal{L}(G,N) \subseteq L$ and $\mathcal{C}(G,N) \odot \mathcal{L}(G,M) \subseteq L$.

We can now define the universal morphism.

DEFINITION 6.3
Suppose G is a grammar that defines a language L; the *universal morphism* ϕ_L is the CFG-morphism that for any non-terminal N of G maps $N \to \mathcal{L}(G,N)^{\rhd\lhd}$.

Note that the universal morphism is an exact morphism by the previous lemma.

We can now state the following proposition, which informally states that if a grammar is minimal then there is at most one non-terminal for each concept.

PROPOSITION 6.4
If the grammar is minimal, then the universal morphism is injective.

PROOF. If there were two non-terminals for a concept, the image under the universal morphism would have one fewer non-terminal and would thus be smaller than the original grammar, contradicting the assumption of minimality. ■

We may not be able to compute, for an arbitrary CFG whether two non-terminals are mergeable, but that is to a certain extent irrelevant. This proposition tells us that we can restrict ourselves to non-terminals that are concepts. We have two consequences—one is that that for every context-free language we have a grammar where the non-terminals are closed sets of strings; secondly, the simplest grammars for every language will be of this type. In the next section, we will discuss the special case of regular languages where the exact computations are possible.

7 Regular languages

In the case of regular languages, we know the lattice will be finite and this simplifies the analysis in a number of respects. In this case, all of the concepts can be defined using a finite number of substrings. Accordingly, it is possible to define the same structure using only idempotent semirings rather than CISs. In this case the construction is called the *syntactic semiring* and was introduced in [29, 30]. We briefly summarize the syntactic semiring in a notation consistent with this article. Define $\mathcal{F}(X)$ to be the set of all finite subsets of a set X. Then $\mathcal{F}(\Sigma^*)$ is the free idempotent semiring over Σ. Consider a language $L \subseteq \Sigma^*$. We can define a congruence on $\mathcal{F}(\Sigma^*)$, given X, Y which are elements of $\mathcal{F}(\Sigma^*)$, $X \sim_L Y$ iff $X^{\rhd} = Y^{\rhd}$. That is to say merely the restriction of the congruence to the collection of finite sets of strings. The quotient $\mathcal{F}(\Sigma^*)/\sim_L$ is the syntactic semiring. It can be shown that this is isomorphic to the SCL in the case of regular languages. Note that since $\mathcal{F}(\Sigma^*)$ is countable so is the syntactic semiring. Consider the example O_1^c. In this case the syntactic semiring is countable, whereas the syntactic concept lattice is uncountable; and there is no element that corresponds to L.

We can also construct a canonical representation. This is the universal CFG for the language and will contain a morphic image of every (binarized) CFG for the language. We emphasize that in the case of a non-regular context-free language, we can do the same construction but the resulting object will be infinite and thus not strictly speaking a grammar. Nonetheless, it will contain as with the finite case an image of every grammar for the language.

DEFINITION 7.1
Given a regular language L, the universal context-free grammar for the language $\mathfrak{G}(L)$ is the CFG $\langle V, \mathcal{C}(L), \Sigma, P\rangle$, where V is the set of concepts of L, and P is the set of productions $\{Q \rightarrow RS \mid Q \geq R \circ S\} \cup \{N \rightarrow a | a \in N, a \in \Sigma \cup \{\lambda\}\}$.

We can now demonstrate an exact correspondence between the derivations in the grammar, and the pair of sets of contexts and sets of strings that constitute a non-terminal in the grammar.

PROPOSITION 7.2
For any language L and any concept $X = (S, C)$, we have

- $\mathcal{L}(\mathfrak{G}(L), (S, C)) = S$
- $\mathcal{C}(\mathfrak{G}(L), (S, C)) = C$.

PROOF. By induction on the length of a derivation. Suppose $S \overset{*}{\Rightarrow} lXr$. We know $\mathcal{C}(l) \overset{*}{\Rightarrow} l$ and $\mathcal{C}(r) \overset{*}{\Rightarrow} r$, by part 1, and $S \geq \mathcal{C}(l) \circ X \circ \mathcal{C}(r)$ which gives us the two rules. ■

PROPOSITION 7.3
Suppose we have a regular language L and a CFG G such that $\mathcal{L}(G) = L$. Then there is an exact morphism from G to $\mathfrak{G}(L)$.

PROOF. The morphism $N \rightarrow (\mathcal{L}(G,N)^{\rhd\lhd}, \mathcal{L}(G,N)^{\rhd})$ is such a morphism. ■

PROPOSITION 7.4
The universal grammar does not have any mergeable states.

PROPOSITION 7.5
Given a finite automaton generating a language L, we can effectively construct $\mathfrak{G}(L)$.

We sketch the technical details for constructing a structure isomorphic to $\mathfrak{B}(L)$ from a DFA for L. Take a minimal DFA for L and for each state q, pick a string that reaches q from the start state. Let this set of strings be A. Do the same thing for a minimal DFA for L^R, the reversal of L, to get a set of strings B. This defines a set of contexts $A \times B^R$. Now pick one string from each element of the syntactic monoid of L to get a set of strings M. The Galois connection between the set of contexts $A \times B^R$ and the set of strings M will be isomorphic to $\mathfrak{B}(L)$, from which it is straightforward to construct $\mathfrak{G}(L)$.

We will now give a complete example for a regular language. Consider $L = (ab)^*$. This has the 7 closed sets of strings: $\Sigma^*, \emptyset, L, \{\lambda\}, a(ba)^*, b(ab)^*, (ba)^*$. We therefore have 7 non-terminals in $\mathfrak{G}(L)$ that we denote $\top, \bot, S, \lambda, A, B, R$. The lexical rules in the grammar are: $\top \rightarrow a, \top \rightarrow b, \top \rightarrow \lambda, L \rightarrow \lambda, R \rightarrow \lambda, A \rightarrow a, B \rightarrow b$. The start symbol is S. There are 7^3 possible binary rules of which the following are in the grammar:

- Any production of the form $\top \rightarrow XY$.
- Any production of the form $X \rightarrow \bot Y$ or $X \rightarrow Y\bot$.
- Any production of the form $X \rightarrow \lambda X$ or $X \rightarrow X\lambda$.
- $L \rightarrow AB | LL$

- $A \to AR|LA$
- $B \to BL|RB$
- $R \to BA|RR$

$\mathfrak{G}(L)$ is a large but finite CFG. For any (binarized) grammar G that defines the same language, ϕ_L is an exact morphism from G to $\mathfrak{G}(L)$. That is to say this grammar contains a morphic image of every grammar for the language; this is analogous to the UA which we discuss now.

7.1 The universal automaton

We now consider the relation of the syntactic concept lattice to the UA [12, 27]. We recall the definition of the UA using the terminology of [27]; interestingly the Galois lattice structure of the UA does not form part of the standard presentations of the UA.

Definition 7.6
Given a language L, a factorization of L is a maximal pair of sets of strings P,Q such that $PQ \subseteq L$.

We can of course consider this as a Galois connection between the prefixes and suffixes.

Definition 7.7
For a set of strings X, define $X^{\blacktriangleleft} = \{y|Xy \subseteq L\}$ and $X^{\blacktriangleright} = \{y|yX \subseteq L\}$.

These are the left and right residuals. A left factor is a set X (of prefixes) such that it is left-closed: $X = X^{\blacktriangleleft\blacktriangleright}$; conversely a right factor is one where $X = X^{\blacktriangleright\blacktriangleleft}$. A factorization (P,Q) is thus a pair where $P^{\blacktriangleleft} = Q$ and $Q^{\blacktriangleright} = P$; clearly P is left-closed and Q is right-closed.

Definition 7.8
For a language L, let UA(L) be the set of all factorizations of L.

Definition 7.9
The UA of a regular language L is defined to be a finite automaton where the states are factorizations, and where the set of initial states is $\{(P,Q)|\lambda \in P\}$, the set of final states is $\{(P,Q)\,|\,\lambda \in Q\}$, and the set of transitions is defined as $\{(P,Q) \to^{a} (P',Q')\,|\,PaQ' \subseteq L\}$.

We now consider a number of close relationships between the UA and the SCL for a given language.

Proposition 7.10
Let (P,Q) be a factorization of a language. Then both P and Q are closed sets of strings, in the lattice $\mathfrak{B}(L)$. i.e. $P = P^{\rhd\lhd}$ and $Q = Q^{\rhd\lhd}$.

Proof. If P is left-closed, then it is closed since $P = \{(\lambda, P^{\blacktriangleleft})\}^{\lhd}$. Similarly if Q is right-closed, then $Q = \{(Q^{\blacktriangleright}, \lambda)\}^{\lhd}$. ■

However it is not necessarily the case that (P,λ) or (λ,Q) are closed sets of contexts.

Finally, we note that the set of contexts of the empty string are equivalent to the factorizations.

Proposition 7.11

$$\{\lambda\}^{\rhd} = \bigcup_{(P,Q)\in UA(L)} P \times Q$$

Proof. Suppose (P,Q) is a factorization; Therefore $P \times Q \odot \lambda = PQ \subseteq L$, so $P \times Q \subseteq \{\lambda\}^{\rhd}$. Conversely if (l,r) in $\{\lambda\}^{\rhd}$, then $lr \in L$ so (l,r) must be an element of at least one maximal factorization. ■

8 Discussion

The SCL appears quite naturally under several equivalent definitions; as the Galois connection between strings and contexts; as the set of maximal decompositions of the language into strings and contexts, and as the unique minimal CIS that recognizes a language. It can be seen as the lifting of the syntactic congruence to sets of strings rather than strings. It forms the basis for various algorithms for grammar induction. It also has deep relations with the syntactic congruence and the UA. These considerations suggest that it is worthy of sustained analysis and study, as it potentially can give rise to new algorithms for various problems associated with CFGs.

Additionally realizing that these structures are also residuated lattices, has as a consequence that they can be used to give a complete and sound semantics for the Lambek calculus [10, 33]. This gives a new perspective on the slightly troubled relationship between Lambek grammars and CFGs; the SCL is a point at which these two formalisms seems to make contact in a way very similar to Lambek's original conception. Lambek justifies the rules of his calculus by appeal to the soundness of the deduction system with respect to residuated lattices and later says [24]

> We shall assign type n to all expressions which can occur in any context in which all proper names can occur.

If we write N for the set of all proper names, then the set of contexts in which all proper names can occur is $N^{\triangleright}$, and so this states, in our terminology, that the type n should be assigned to the set $N^{\triangleright\triangleleft}$. Thus the types of the Lambek calculus were, it appears, intended to be closed sets of strings.

The phrase 'an algebraic theory of context-free languages' and indeed the now rather dated term 'algebraic language' for context-free language, has been used in a number of different contexts [4], as part of a program to base language theory not on phrase structure grammars or on machine models (finite-state automatons or push-down automata) but rather on some algebras (e.g. [21]). The work we present here can be considered as part of the same research program; rather than dealing directly with the productions of a CFG, we operate with an algebraic structure that interprets it; a CIS. Homomorphisms of this structure can then be used to model transformations, i.e. morphisms, of the original grammar. We then naturally arrive at a unique object for each grammar, the terminal element of a category, and thus a universal morphism for CFGs. This gives us a set of syntactic categories that can be used as non-terminals in a grammar; under the assumption that we want the grammar to be minimal in a fairly obvious sense, we can without loss of generality assume that the grammar uses only these non-terminals. Each production then has an easily interpretable form as an inequality (or equality) between these syntactic categories.

Though the computational problems of constructing the syntactic concept lattice from a CFG is in general undecidable, it is straightforward to approximate it. By picking some finite set of contexts and finite set of substrings, and using a standard parser, we can efficiently compute a fragment of the lattice. This process relies only on information about the set of strings in the language and can thus be used as the basis for a variety of learning algorithms. Non-terminals in the grammar are defined which correspond to specific concepts in the lattice of the language being learned. The algorithm can either pick a finite set of strings S, which defines the concept $\mathcal{C}(S)$, or a finite set of contexts C, which defines the concept $\mathcal{C}(C^{\triangleleft})$. Algorithms that use the former assumption are called *primal* and ones that use the latter are called *dual*. Given a suitable set of non-terminals defined primally, dually or using a combination of the two, the algorithm can then define a set of rules, which, given the objective nature of the non-terminals, is straighforward. Given non-terminals N, P, Q that correspond to concepts $\mathcal{N}, \mathcal{P}, \mathcal{Q}$ we want a production $N \to PQ$ if and only if $\mathcal{N} \geq \mathcal{P} \circ \mathcal{Q}$. Testing this inequality depends on how the non-terminals are defined, but in general the validity of the inequality depends only on the concepts used in the inequality. This means that we replace the problem of testing the global validity of an entire grammar with a test which is local and computationally tractable. This massively simplifies the process of constructing a grammar for a language, and allows for efficient learning algorithms under a number of different models [8, 34, 35].

The UA has been used as the basis for algorithms for minimizing regular grammars; for finding the smallest NFA that generates a given regular language. Though the problem of finding the minimal NFA that generates the same language as a given DFA is known to be PSPACE-complete [22], various algorithms have been proposed using the UA as a search space. Although the problem of finding the minimal CFG that generates the same language as a given CFG is clearly undecidable, we could use the SCL to find the smallest CFG that generates a given regular language, presented as an automaton, using an exhaustive search through the SCL.

This has a number of implications also for linguistics, where phrase structure grammars, of which CFGs are the simplest type, are widely used. Classically we identify the parse tree generated by the phrase structure grammar with the syntactic structure of the sentence, and the syntactic categories correspond therefore to the non-terminals of the grammar. From the perspective taken here, the minimality of grammars whose non-terminals are syntactic concepts is a very strong argument in favour of the use of syntactic concepts as syntactic categories in linguistic description. To rebut this argument would require an argument that the correct grammar is redundant in the sense that it contains mergeable non-terminals. In the absence of such an argument we can conclude, that at least in the domain of CFGs, that syntactic structure must supervene on distributional structure; which gives a strong in principle argument for the central importance of distributional analysis in linguistic theory.

Acknowledgements

I am very grateful to Ryo Yoshinaka and Hans Leiss for extensive technical discussions on the syntactic concept lattice, which have greatly improved, corrected and simplified the presentation; any remaining errors are of course my own. I am also especially grateful to Hans Leiss for pointing out the connection to compete idempotent semirings. I would also like to thank the anonymous reviewers for helpful suggestions.

References

[1] D. Angluin. Inference of reversible languages. *Journal of the ACM*, **29**, 741–765, 1982.

[2] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, **75**, 87–106, 1987.

[3] B. Bollig, P. Habermehl, C. Kern, and M. Leucker. Angluin-style learning of NFA. In *Proceedings of IJCAI 21*, 2009.

[4] N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. *Studies in Logic and the Foundations of Mathematics*, **35**, 118–161, 1963.

[5] A. Clark. Distributional learning of some context-free languages with a minimally adequate teacher. In *Grammatical Inference: Theoretical Results and Applications. Proceedings of the International Colloquium on Grammatical Inference*, J. Sempere and P. Garcia, eds, pp. 24–37. Springer, 2010.

[6] A. Clark. Efficient, correct, unsupervised learning of context-sensitive languages. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pp. 28–37, Uppsala, Sweden. Association for Computational Linguistics, 2010.

[7] A. Clark. Learning context free grammars with the syntactic concept lattice. In *Grammatical Inference: Theoretical Results and Applications. Proceedings of the International Colloquium on Grammatical Inference*, J. Sempere and P. Garcia, eds, pp. 38–51. Springer, 2010.

[8] A. Clark. Three learnable models for the description of language. In *Language and Automata Theory and Applications, Fourth International Conference, LATA 2010*, LNCS, C. Martín-Vide, A.-H. Dediu, and H. Fernau, eds, pp. 16–31. Springer, 2010.

[9] A. Clark. A learnable representation for syntax using residuated lattices. In *Formal Grammar*, P. Groote, M. Egg, and L. Kallmeyer, eds, vol. 5591 of *Lecture Notes in Computer Science*, pp. 183–198. Springer, 2011.

[10] A. Clark. Logical grammars, logical theories. In *Logical Aspects of Computational Linguistics*, D. Bechet and A. Dikovsky, eds, pp. 1–20. Springer, 2012.

[11] A. Clark and R. Eyraud. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, **8**, 1725–1745, 2007.

[12] J. H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.

[13] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002.

[14] F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using RFSAs. *Theoretical Computer Science*, **313**, 267–294, 2004.

[15] S. Eilenberg. *Automata, Languages, and Machines*. Academic Press, 1974.

[16] Z. Ésik and H. Leiß. Algebraically complete semirings and Greibach normal form. *Annals of Pure and Applied Logic*, **133**, 173–203, 2005.

[17] N. Galatos, P. Jipsen, T. Kowalski, and H. Ono. *Residuated Lattices: An Algebraic Glimpse at Substructural Logics*. Elsevier, 2007.

[18] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, 1997.

[19] S. Ginsburg and H. G. Rice. Two families of languages related to ALGOL. *Journal of the ACM (JACM)*, **9**, 350–371, 1962.

[20] J. S. Golan. *Semirings and their Applications*. Springer, 1999.

[21] G. Hotz. A representation theorem of infinite dimensional algebras and applications to language theory. *Journal of Computer and System Sciences*, **33**, 423–455, 1986.

[22] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, **22**, 1117–1141, 1993.

[23] P. Jipsen. From semirings to residuated Kleene lattices. *Studia Logica: An International Journal for Symbolic Logic*, **76**, 291–303, 2004.

[24] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, **65**, 154–170, 1958.

[25] H. Leiss. Learning CFGs with the finite context property: a note on A. Clark's algorithm. Manuscript, 2012.

[26] S. Lombardy. On the size of the universal automaton of a regular language. In *STACS 2007*, W. Thomas and P. Weil, eds, vol. 4393 of *Lecture Notes in Computer Science*, pp. 85–96. Springer, 2007.

[27] S. Lombardy, J. Sakarovitch. The universal automaton. In *Logic and Automata: History and Perspectives*, J. Flum, E. Grädel, and T. Wilke, eds, vol. 2, pp. 457–504. Amsterdam University Press, 2008.

[28] F. C. N. Pereira and S. M. Shieber. The semantics of grammar formalisms seen as computer languages. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd Annual Meeting on Association for Computational Linguistics*, pp. 123–129. Association for Computational Linguistics, 1984.

[29] L. Polák. Syntactic semiring of a language. In *Mathematical Foundations of Computer Science 2001*, J. Sgall, A. Pultr, and P. Kolman, eds, pp. 611–620. Springer, 2001.

[30] L. Polák. Syntactic semiring and universal automaton. In *Developments in Language Theory*, Z. Ésik, and Z. Fülöp, eds, pp. 411–422. Springer, 2003.

[31] M. O. Rabin and D. Scott. Finite Automata and their decision problems. *IBM Journal of Research and Development*, **3**, 114–125, 1959.

[32] J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[33] C. Wurm. Completeness of full Lambek calculus for syntactic concept lattices. In *Proceedings of the 17th conference on Formal Grammar 2012 (FG)*, 2012.

[34] R. Yoshinaka. Towards dual approaches for learning context-free grammars based on syntactic concept lattices. In *Developments in Language Theory*, G. Mauri and A. Leporati, eds, vol. 6795 of *Lecture Notes in Computer Science*, pp. 429–440. Springer, 2011.

[35] R. Yoshinaka. Integration of the dual approaches in the distributional learning of context-free grammars. In *LATA*, A.-H., Dediu, and M.-V., Carlos, eds, pp. 538–550. Springer, 2012.

Received 8 March 2013