IEML Semantic Topology

A Formal Model of the Circuits of the Information Economy

Prof. Pierre Lévy, CRC, FRSC, University of Ottawa

23 août 2010

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Abstract

The model of the IEML language and of its semantic variables is presented in Chapter 2. We show that IEML is a *regular language*, a class of languages that is extremely efficient at computational tasks involving sequencing and repetition, and is furthermore recognized by *finite state machines*.

The *group structure* describes the mathematical concept of symmetry, where symmetry can be understood as invariance under some transformations. Symmetry allows the recognition of similarities, and to discover which properties of elements do not change under transformations. We show in Chapter 3 that IEML semantic variables possess characteristics of *groups* and *rings*.

We consider the *computability* of transformations applied to the IEML language in Chapter 4. Using finite state machines as the underlying computational model, we show that transformations defined for the semantic variables of the IEML language are computable.

While IEML language is used to create IEML expressions, semantic graphs are a representation of the *semantic relations* occurring within IEML expressions. The model of IEML relations is presented in Chapter 5. Combination of a plurality of semantic relations results in *semantic circuits*. In Chapter 6, we show that semantic circuits form a groupoid, and present algorithms for functions applicable to rhizomes, a type of semantic circuits.

Chapter 7 introduces quantitative criteria used to compare any two semantic circuits. These criteria can be used as a basis for defining a notion of *semantic distance* between two semantic circuits.

Acknowledgments The research that led to IEML semantic topology has been funded by the Canada Research Chair federal program and by the Social Sciences and Humanities Research Council of Canada. I would like to thank Andrew Roczniak, PhD, who worked with me to formalize IEML during seven years and Nick Soveiko, PhD, who reviewed the final version of this text and is the main contributor of chapter 7. I am nevertheless the sole responsible for any error or inconsistency.

Chapitre 2

Model

2.1 Model of IEML Language

Let Σ be a nonempty and finite set of symbols, $\Sigma = \{S, B, T, U, A, E\}$. Let a string s be a finite sequence of symbols chosen from Σ. The length of this string is denoted by $|s|$. An empty string ϵ is a a string with zero occurrence of symbols and its length is $|\epsilon| = 0$. The set of all strings of length k composed with symbols from$\Sigma$is defined as $\Sigma^k \triangleq \{s \ : \ |s| = k\}$. The set of all strings over Σ is defined as :

$$\Sigma^* \triangleq \Sigma^0 \cup \Sigma^1 \dots \tag{2.1}$$

The IEML language over Σ is a subset of Σ^*, $L_{IEML} \subseteq \Sigma^*$ where $L = 6$:

$$L_{IEML} \triangleq \left\{ s \in \Sigma^* \mid 0 \leq l \leq L, \ |s| = 3^l \right\} \tag{2.2}$$

Proposition 2.1.1. *IEML language given in equation 2.2 is a regular language [14].*

Démonstration. Consider the definition of regular languages :
- $L = \{\emptyset\}$ and $L = \{\epsilon\}$ are regular languages,
- $L = \{\sigma \mid \sigma \in \Sigma\}$ are regular languages,
- if L_1 and L_2 are regular languages, then so are $L_1 \cup L_2$ and $L_1 \cdot L_2$ (concatenation).

Since L_{IEML} can be constructed from its alphabet $\Sigma = \{S, B, T, U, A, E\}$ and using only statements from the above definition, it is a regular language. □

We note that any string belonging to the L_{IEML} language can be also obtained from the Σ alphabet by the application of the *triplication* function. String concatenation takes two strings and produces a third string which is composed of symbols of the first string followed by the symbols of the second string. The IEML triplication function is a specialization of the string concatenation, where three strings (a, b, c) belonging to L_{IEML} and of the same length are concatenated and where the length of each string is at most 3^{L-1} :

$$f_t(a, b, c) \triangleq (abc \ : \ |a| \leq 3^{L-1} \wedge \ |a| = |b| = |c| \ \wedge \ a, b, c \in L_{IEML}) \tag{2.3}$$

Regular languages are extremely efficient at computational tasks involving sequencing and repetition and can be recognized by finite state machines, which are discussed in section 4.1.1.

2.2 Model of Semantic Sequences

Definition 2.2.1. *A string s is a semantic sequence if and only if $s \in L_{IEML}$.*

Unless otherwise specified, 'sequence' and 'semantic sequence' are used interchangeably in the remainder of the text. To denote the p_n'th primitive of a sequence s, we use a superscript n where $1 \leq n \leq 3^l$ and write s^n. Note that for any sequence s of layer l, s^n is undefined for any $n > 3^l$. Two semantic sequences are distinct if and only if either of the following holds : a) their layers are different, b) they are composed from different primitives, c) their primitives do not follow the same order : for any s_a and s_b,

$$s_a = s_b \iff \forall n,\ s_a^n = s_b^n \ \wedge\ |s_a| = |s_b| \tag{2.4}$$

Let's now consider binary relations between semantic sequences in general. These are obtained by performing a Cartesian product of two sets [1]. For any set of semantic sequences X, Y where $s_a \in X$, $s_b \in Y$ and using equation 2.4, we define four binary relations *whole* $\subseteq X \times Y$, *substance* $\subseteq X \times Y$, *attribute* $\subseteq X \times Y$ and *mode* $\subseteq X \times Y$ as follows :

$$\texttt{whole} \triangleq \{(s_a, s_b) \mid s_a = s_b\} \tag{2.5}$$

$$\texttt{substance} \triangleq \{(s_a, s_b) \mid 1 \leq n \leq |s_b|,\ s_a^n = s_b^n \ \wedge\ |s_a| = 3|s_b|\} \tag{2.6}$$

$$\texttt{attribute} \triangleq \{(s_a, s_b) \mid 1 \leq n \leq |s_b|,\ s_a^{n+|s_b|} = s_b^n \ \wedge\ |s_a| = 3|s_b|\} \tag{2.7}$$

$$\texttt{mode} \triangleq \{(s_a, s_b) \mid 1 \leq n \leq |s_b|,\ s_a^{n+2|s_b|} = s_b^n \ \wedge\ |s_a| = 3|s_b|\} \tag{2.8}$$

Equation 2.5 states that any two semantic sequences that are equal are in a `whole` relationship (we can also write s_b `whole` s_a). Equations 2.6, 2.7 and 2.8 state that any two semantic sequences that share specific subsequences may be in `substance`, `attribute` or `mode` relationship. For any two semantic sequences s_a and s_b, if they are in one of the above relations, then we say that s_b plays a *role w.r.t* s_a and we call s_b a *seme* of sequence :

Definition 2.2.2. *For any semantic sequence s_a and s_b, if*

$$(s_a, s_b) \in \texttt{whole} \cup \texttt{substance} \cup \texttt{attribute} \cup \texttt{mode}$$

then s_b plays a role w.r.t s_a and s_b is called a seme.

We can now group distinct semantic sequences together into sets. A useful grouping is based on the layer of those semantic sequences, as discussed in section 2.3.

1. A Cartesian product of two sets X and Y is written as follows : $X \times Y = \{(x, y) \mid x \in X, y \in Y\}$

2.3 Model of Semantic Categories

A *category* of L_{IEML} of layer l, is a subset of L_{IEML} such that all semantic sequences of that subset have the same length :

$$c_l \triangleq \left\{ s \mid s \in L_{IEML} \ \wedge \ |s| = 3^l \right\} \tag{2.9}$$

Definition 2.3.1. *A semantic category c is a set containing semantic sequences at the same layer.*

Unless otherwise specified, '*category*' and 'semantic *category*' are used interchangeably in the remainder of the text. The layer of any *category* c is exactly the same as the layer of the semantic sequences included in that *category*. The set of all *categories* of layer L is given as the powerset [2] of the set of all strings of layer L of L_{IEML} :

$$C_L \triangleq \mathcal{P}(\{c_L\}) \tag{2.10}$$

Two *categories* are distinct if and only if they differ by at least one element. For any c_a and c_b :

$$c_a = c_b \iff c_a \subseteq c_b \wedge c_b \subseteq c_a \tag{2.11}$$

A weaker condition can be applied to *categories* of distinct layers (since two *categories* are different if their layers are different) and is written as :

$$\ell(c_a) \neq \ell(c_b) \Longrightarrow c_a \neq c_b \tag{2.12}$$

where $\ell(\cdot)$ denotes the layer of a *category*.

Analogously to sequences, we consider binary relations between any *categories* c_i and c_j where $\ell(c_i), \ell(c_j) \geq 1$. For any set of *categories* X, Y where $c_a \in X$, $c_b \in Y$ and using equations 2.11, 2.6, 2.7 and 2.8 we define four binary relations $whole_C \subseteq X \times Y$, $substance_C \subseteq X \times Y$, $attribute_C \subseteq X \times Y$ and $mode_C \subseteq X \times Y$ as follows :

$$\texttt{whole}_C \triangleq \{(c_a, c_b) \mid c_a = c_b\} \tag{2.13}$$

$$\texttt{substance}_C \triangleq \{(c_a, c_b) \mid \forall\ s_a \in c_a, \exists s_b \in c_b, (s_a, s_b) \in \texttt{substance}\} \tag{2.14}$$

$$\texttt{attribute}_C \triangleq \{(c_a, c_b) \mid \forall\ s_a \in c_a, \exists s_b \in c_b, (s_a, s_b) \in \texttt{attribute}\} \tag{2.15}$$

$$\texttt{mode}_C \triangleq \{(c_a, c_b) \mid \forall\ s_a \in c_a, \exists s_b \in c_b, (s_a, s_b) \in \texttt{mode}\} \tag{2.16}$$

For any two *categories* c_a, c_b, if they are in one of the above relations, then we say that c_b plays a *role* with respect to c_a and c_b is called a seme of *category* :

Definition 2.3.2. *For any category c_a and c_b, if*

$$(c_a, c_b) \in \texttt{whole}_C \cup \texttt{substance}_C \cup \texttt{attribute}_C \cup \texttt{mode}_C$$

then c_b plays a role with respect to c_a and we call c_b a seme.

2. A powerset of S is the set of all subsets of S, including the empty set $\emptyset$

2.3.1 Types of semantic categories

Categories will usually be generated using the following approach. Taking the powerset of Σ (the set of all subsets of Σ, including the empty set $\emptyset$), we represent it by : $\Sigma_{IEML} \triangleq \mathcal{P}(\Sigma)$. Σ_{IEML} contains sets such as $\{S\}$, $\{A\}$, $\{U, A\}$, $\{E\}$, $\{S, B, T\}$, $\{\emptyset\}$. We note that the order within the members of the set Σ_{IEML} is irrelevant, so for example symbols $\{U, A\}$ and $\{A, U\}$ are considered to be one and the same. Σ_{IEML} is the set of all *IEML semantic characters* and we define the *category language* as the language over :

$$\bar{\Sigma} \triangleq \Sigma_{IEML} \backslash \{\emptyset\} \tag{2.17}$$

Let a string s be a finite sequence of symbols chosen from $\bar{\Sigma}$. The length of this string is denoted by $|s|$. An empty string ϵ is a a string with zero occurrence of symbols and its length is $|\epsilon| = 0$. The set of all strings of length k composed with symbols from $\bar{\Sigma}$ is defined as $\bar{\Sigma}^k \triangleq \{s \,:\, |s| = k\}$. The set of all strings over $\bar{\Sigma}$ is defined as :

$$\Sigma^*_{cat} \triangleq \bar{\Sigma}^0 \cup \bar{\Sigma}^1 \ldots \tag{2.18}$$

The category language over $\bar{\Sigma}$ is a subset of Σ^*_{cat}, $L_{cat} \subseteq \Sigma^*_{cat}$ where $L = 6$:

$$L_{cat} \triangleq \left\{ s \in \Sigma^*_{cat} \mid 0 \leq l \leq L,\ |s| = 3^l \right\} \tag{2.19}$$

IEML sequences are obtained from any sequence $s \in L_{cat}$ by performing a Cartesian product between all the symbols of that sequence s. For instance, the sequence $\{S, B, T\}\{U, A\}\{E\}$ gives the IEML set of sequences $\bar{s}$:

$$\bar{s} = \{SUE, SAE, BUE, BAE, TUE, TAE\} \tag{2.20}$$

We now introduce the following category types :

Definition 2.3.3. *A category c of layer l is singular if the following holds : $|c| = 1$ and the sequence $s \in c$ is composed by symbols s^n where $s^n \in \bar{\Sigma} \wedge |s^n| = 1$ for $1 \leq n \leq 3^l$.*

All categories that are not singular are *plural*.

Definition 2.3.4. *A category c of layer l is simple if the following holds : $|c| = 1$ and the sequence $s \in c$ is composed by symbols s^n where $s^n \in \bar{\Sigma}$ for $1 \leq n \leq 3^l$.*

All categories that are not simple are *complex*. We note that all singular categories are also simple, since $\bar{\Sigma}$ contains all symbols which contain only one member ($\{S\}$, $\{B\}$, $\{T\}$, $\{U\}$, $\{A\}$, $\{E\}$).

2.4 Model of Catsets and USLs

A catset is a set of distinct *categories* of the same layer :

Definition 2.4.1. *A catset κ is a set containing categories $\kappa_l = \{c \mid l(c) = l\}$ such that $\forall a, b \in \kappa_l,\ a \neq b$*

The layer of a catset [3] is given by the layer of any of its members : if some $c \in \kappa$, then $\ell(\kappa) = \ell(c)$. A USL is composed of up to seven catsets of different layers:

Definition 2.4.2. *A USL u is a set containing catsets of different layers :* $u = \{\kappa\}$ *such that* $\forall a, b \in u,\ l(a) \neq l(b)$

Note that since there are 7 distinct layers, a USL can have at most seven members. All standard set operations on USLs are always performed on sets of *categories* (and therefore on sets of sequences), layer by layer.

3. Note that a *category* c can be written as $c \in C_L$, while a catset κ can be written as $\kappa \subseteq C_L$

Chapitre 3

Symmetry Properties

3.1 Generalities

The mathematical concept of *group* consists of a *set,* of a *binary operation*, and some *properties* when the operation is applied to members of the set. The binary operation $\circledast$ on some set S associates to elements x and y of S a third element $x \circledast y$ of S. Properties are associativity ($\forall x, y, z \in S,\ (x \circledast y) \circledast z = x \circledast (y \circledast z)$), existence of an identity element and existence of an inverse element for each element of S.

By denoting the identity element of a group S as x^1 and inverse element x^{-1} for each $x \in S$, the following are the basic properties of groups :

Proposition 3.1.1. *A group $(S, \circledast)$ has exactly one identity element x^1 satisfying $\forall x \in S,\ x \circledast x^1 = x^1 \circledast x = x$*

Démonstration. Assume that some $i \in S$, has the property $i \circledast x = x,\ \forall x \in S$. Then we have $i = i \circledast x^1 = x^1$. Similarly, if $i \in S$ has the property $x \circledast i = x,\ \forall x \in S$, then we have $i = x^1 \circledast i = x^1$. □

Proposition 3.1.2. *A group $(S, \circledast)$ has exactly one inverse element x^{-1} for each $x \in S$.*

Démonstration. From the group properties, there exists an element $x^{-1} \in S$ with the property $x \circledast x^{-1} = x^{-1} \circledast x = x^1,\ \forall x \in S$. For any $i \in S$ with the property $x \circledast i = x^1$, then $i = x^1 \circledast i = (x^{-1} \circledast x) \circledast i = x^{-1} \circledast (x \circledast i) = x^{-1} \circledast x^1 = x^{-1}$. Similarly for any element with the property $i \circledast x = x^1$, then $i = x^{-1}$, which implies that the inverse of any $x \in S$ is uniquely determined. □

Proposition 3.1.3. *For any elements x, y of a group $(S, \circledast)$, $(x \circledast y)^{-1} = y^{-1} \circledast x^{-1}$*

Démonstration. We verify that $(x \circledast y) \circledast (y^{-1} \circledast x^{-1}) = x \circledast (y \circledast (y^{-1} \circledast x^{-1})) = x \circledast (y \circledast y^{-1} (\circledast x^{-1})) = x \circledast (x^1 \circledast x^{-1}) = x \circledast x^{-1} = x^1$. Similar result is obtained for $(y^{-1} \circledast x^{-1}) \circledast (x \circledast y)$, which implies that the inverse of $(x \circledast y)$ is $y^{-1} \circledast x^{-1}$. □

The group structure can be extended by adding operations, leading to a two-operation structure called a *ring* ; or by adding properties (for example, the addition of the commutativity property - $x \circledast y = y \circledast x,\ \forall x, y \in S$ - creates an *abelian* group). Removing properties leads to simpler structures called *groupoids.*

3.2 Semantic Categories

For any *category* c we can define a function $f : c \to c$ that is injective, $\forall x, y \in c, f(x) = f(y) \to x = y$ (one-to-one function) and total, $\forall x \in c,\ f(x) \in c$. We gather all distinct functions in a set :

$$F_c = \{f_i \,|\exists x \in c, i \neq j, f_i(x) \neq f_j(x)\} \tag{3.1}$$

There are $|c|!$ such functions, and they represent all the possible permutations of the set c. For any functions f_i,$f_j \in F_c$ the output of one can be used as input to the other, resulting in function composition : $(f_i \circ f_j)(s) \triangleq f_i(f_j(s))$. We now consider *symmetry groups* [4].

Proposition 3.2.1. *The group $G = (F_c, \circ)$, where the group operation is the function composition, is a symmetry group.*

Démonstration. Closure : clearly, successive applications of any $f \in F_c$ is a one-to-one mapping from the set c to the set c. Since F_c contains all one-to-one functions for the set c, then $f_i \circ f_j = f_k \in F_c$.

Associativity : using the definition of function composition, $\forall f_i, f_j, f_k \in F_c, ((f_i \circ f_j) \circ f_k)(s) = (f_i(f_j(f_k(s))))$. On the other hand, $(f_i \circ (f_j \circ f_k))(s) = (f_i(f_j(f_k(s))))$.

Identity : let $f_0(s) = s, \forall s \in c$. Clearly $f_0 \in F_c$. We now have $(f_0 \circ f_i)(s) = f_0(f_i(s)) = f_i(s)$, and on the other hand $(f_i \circ f_0)(s) = f_i(f_0(s)) = f_i(s)$.

Inverse : let $f_i(s) = z, \forall f_i \in F_c$ We define $f_i^{-1}(z) = s$. Then, $(f_i \circ f_i^{-1})(z) = f_i(f_i^{-1}(z)) = z = f_0(z)$ and on the other hand, $(f_i^{-1} \circ f_i)(s) = f_i^{-1}(f_i(s)) = s = f_0(s)$. Since $f_i^{-1}(z)$ is a one-to-one function on c, then $f_i^{-1}(z) \in F_c$. □

3.3 Catsets and USLs

From definition 2.4.1, a catset contains distinct *categories* of the same layer. Since the language L_{IEML} is finite, the number of distinct *categories* of the same layer are given by the size of the set C_L. In general then, by considering C_L as a group, we obtain a special type called a *ring* [4] as shown in the following proposition :

Proposition 3.3.1. *The group $G = (C_L, \oplus, \otimes)$ with the set difference (Δ) and the set intersection ($\cap$) operations, is a ring.*

Démonstration. Symmetric set difference results in members which are in either set, but not in both : for sets A and B, it is $(A \backslash B) \cup (B \backslash A)$. Then $\forall A, B, C \subseteq C_L$:

The group $(C_L, \oplus)$ is an *Abelian group* since $\forall c_i, c_j \in C_L,\ c_i \oplus c_j \in C_L$ by the powerset definition (closure), $(A \oplus B) \oplus C = A \oplus (B \oplus C)$ by associativity of the set union operation (associativity), $A \oplus B = B \oplus A$ by commutativity of the set union operation (commutativity), $\exists c_0 \in C_L,\ \forall c_i \in C_L,\ c_i \oplus c_0 = c_0 \oplus c_i = c_i$ where $c_0 = \emptyset$ (identity), $\exists c_i^{-1} \in C_L,\ \forall c_i \in C_L,\ c_i \oplus c_i^{-1} = c_i^{-1} \oplus c_i = c_0$ where $c_i^{-1} = c_i$ (inverse) .

The group $(C_L, \otimes)$ is a *monoid* since $\forall c_i, c_j \in C_L,\ c_i \otimes c_j \in C_L$ by the powerset definition (closure), $(A \otimes B) \otimes C = A \otimes (B \otimes C)$ by associativity of the set intersection operation (associativity), $\exists c_0 \in C_L,\ \forall c_i \in C_L,\ c_i \otimes c_0 = c_0 \otimes c_i = c_i$ where $c_0 = \emptyset$ (identity).

The multiplication distributes over the addition operation : $\forall c_i, c_j, c_k \in C_L$, $c_i \otimes (c_j \oplus c_k) = (c_i \otimes c_j) \oplus (c_i \otimes c_k)$ and $(c_i \oplus c_j) \otimes c_k = (c_i \otimes c_k) \oplus (c_j \otimes c_k)$ by distributivity of the set intersection operation over the set symmetric difference operation. □

From definition 2.4.2, a USL is a set of catsets of different layers. Since at each layer L there is $|C_L|$ distinct catsets, the whole semantic space is defined by the tuple

$$\mathcal{USL} = C_0 \times C_1 \times C_2 \times C_3 \times C_4 \times C_5 \times C_6 \tag{3.2}$$

Considering $\mathcal{USL}$ as a group, we obtain the following proposition :

Proposition 3.3.2. *The group $G_{\mathcal{USL}} = (\mathcal{USL}, \oplus, \otimes)$ with the symmetric set difference (Δ) and the set intersection ($\cap$) operations applied at each layer, is a ring [4].*

Démonstration. The proof follows the proof for proposition 3.3.1. □

3.4 Transformational Symmetry

We describe transformational symmetries using the concept of *categories* [1],

$$\mathbb{C} = (O, M, \circ) \tag{3.3}$$

where O is a collection of objects, M a collection of morphisms between two members of the objects' collection, and a binary composition operation between compatible morphisms. The following also holds : $\forall a, b, c \in O$, if $u = (a \to b) \in M$ and $v = (b \to c) \in M$ then $\exists w = u \circ v = (a \to c) \in M$; $\forall u, v, w \in M$, if $(u \circ v) \circ w \in M$ then $(u \circ v) \circ w = u \circ (v \circ w) \in M$; $\forall a \in O$, $\exists i_a \in M$ such that $\forall u = (a \to b)$, $i_a \circ u = u \ \wedge \ u \circ i_a = u$.

3.4.1 IEML language as a category

If we consider objects in the collection O to be subsets of L_{IEML} from equation 2.2 as an example, $O = \{o_i \mid o_i \subset L_{IEML}; \forall s \in o_i, \ |s| = 3^i; \ 0 \leq i \leq 6\}$ then morphisms are given by $M = \{m_i \mid m_i = (o_i \to o_{i+1}); \ 0 \leq i \leq 5\}$. The specific function represented by the morphism is the *triplication* total function, $f_t : o_i \to o_{i+1}$ defined in equation 2.3.

3.4.2 Role permutation symmetry

The collection M in equation 3.3 also contains an automorphism $m_r = (o_i \to o_i)$ represented by an unary operation acting on specified roles of each sequence of a semantic category (see section 2.3).

3.4.3 Seme permutation symmetry

The collection M in equation 3.3 also contains an automorphism $m_s = (o_i \to o_i)$ represented by an binary operation acting on the specified role and specified semes of each sequence of a semantic category (see section 2.3).

Chapitre 4

Computability

4.1 Generalities

In order to demonstrate the computability of semantic transformations $f : S \rightarrow S$ where S represents semantic variables (either categories, catsets or USLs), we will use the finite state machine formalism. All operations based on transformations of semantic variables are computable following the properties of finite state machines.

4.1.1 Finite state machines

A finite state machine (FSM) can be defined [8] as :

Definition 4.1.1. *(FSM) A finite state machine M is a synchronous system represented by a quintuple $M = (\Sigma, \Gamma, Q, \delta, \omega)$ with a finite input alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_i\}$, a finite output alphabet $\Gamma = \{\gamma_1, \ldots, \gamma_j\}$, a finite state set $Q = \{q_1, \ldots, q_n\}$ and a pair of characterizing functions δ and ω given by $q_{v+1} = \delta(q_v, \sigma_v)$, $\gamma_v = \omega(q_v, \sigma_v)$ where σ_v, γ_v, q_v are the input symbol, output symbol and state of M at $v = 1, 2, \ldots$*

Thus the input to the machine is a sequence of symbols $\sigma_1 \ldots \sigma_k$, and the output of a machine is a sequence of symbols $\gamma_1 \ldots \gamma_k$.

In general, machines may change state even if there is no input, may have multiple starting states and there may be none or more executable transition rules. To accommodate those differences a transition mapping is defined as $\delta_n : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$, where the return value is represented as a powerset of Q. The output mapping can similarly be defined as $\omega_n : Q \times \Sigma^* \times Q \rightarrow \Gamma^*$. In those nondeterministic cases, an alternative notation is more convenient, and instead of the mappings δ_n and ω_n, the transition relation Δ is used :

Definition 4.1.2. *(NFSM) A nondeterministic finite state machine M is a tuple $M = (\Sigma, \Gamma, Q, Q_0, \Delta, F)$ where Σ is a finite input alphabet, Γ is a finite output alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ are the initial states of the machine, $\Delta \subset Q \times \Sigma^* \times \Gamma^* \times Q$ is the transition relation and F represent the acceptance condition where $F \subseteq Q$.*

Each transition relation $(p, \sigma, \gamma, q) \in \Delta$, or edge from state p to state q can be represented as $p \xrightarrow{\sigma|\gamma} q$, where σ and γ play the role of input and output

Condition		Result	
State i	Input	State $i+1$	Output
q_0	-	q_2	1
q_1	-	q_0	0
q_2	0	q_3	0
q_2	1	q_1	0
q_3	0	q_0	0
q_3	1	q_1	0

TABLE 4.1 – Representation of a machine using a transition table

labels of the edge respectively. Each input to the machine given an initial state will describe a *path* which is a finite sequence [3] of relations represented as : $q_0 \xrightarrow{\sigma_1|\gamma_1} q_1 \xrightarrow{\sigma_2|\gamma_2} q_2 \cdots \xrightarrow{\sigma_n|\gamma_n} q_n$.

The input and output labels of the path are

$$\sigma_{in} = \sigma_1\sigma_2 \ldots \sigma_n \tag{4.1}$$

and

$$\gamma_{out} = \gamma_1\gamma_2 \ldots \gamma_n \tag{4.2}$$

respectively. The machine thus computes the relation $(\sigma_{in}, \gamma_{out})$ on $\Sigma^* \times \Gamma^*$. A path is called *successful* if it starts from an initial state $q_s \in Q_0$ and ends in some state q_f that fulfills an acceptance condition, usually defined as $q_f \in F \subseteq Q$. In general then, and in contrast to deterministic machines, a nondeterministic machine can have multiple successful paths for the same input label σ_{in} resulting in a set of relations

$$R = \{(\sigma_{in}, \gamma_{out}), (\sigma_{in}, \kappa_{out}) \ldots (\sigma_{in}, \eta_{out})\} \tag{4.3}$$

The description of a finite state machine is usually given in a transition table (table 4.1) or transition graph (figure 4.1) [11] from which the δ and ω functions can be obtained. For example, the same machine can be represented by a transition table where rows are a subset of $(Q \times \Sigma \times Q \times \Gamma)$ relations, or a transition graph where edges are labeled with a subset of $(\Sigma \times \Gamma)$ relations.

4.1.2 Finite state automata

A special form of the finite state machine is a deterministic automaton $A_D = (\Sigma, Q, q_0, \delta, F)$. An extended transition function $\hat{\delta}$ that returns a state p when starting from any state q given any valid input sequence w is given by :

$$\hat{\delta}(q, w) = p \tag{4.4}$$

where $q \in Q$ and $w \in \Sigma^*$. This function is defined for all automata and we show this by induction on the length of the input string :

Basis : $q = \hat{\delta}(q, \epsilon)$, which ensures that there is no state transition when there is no input ;

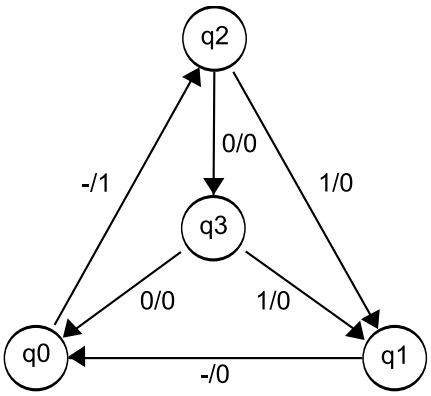

FIGURE 4.1 – Representation of a machine using a graph

Induction : set w as ua, where a is the last symbol and u is the original string without the last symbol. Then,

$$\hat{\delta}(q, ua) = \delta(\hat{\delta}(q, u), a) \tag{4.5}$$

A language [1] L of A_D denoted $L(A_D)$ [12] then is :

$$L(A_D) = \{w \mid \hat{\delta}(q_0, w) \in F\} \tag{4.6}$$

We can define the state equivalence [2] in terms of the set F and equation 4.4 as follows. Two states $p, q \in Q$ are equivalent if and only if $\forall s \in \Sigma^*$, $\hat{\delta}(p, s) \in F \ \wedge \ \hat{\delta}(q, s) \in F$. In general, two automata are equivalent iff $L(A_i) = L(A_j)$, that is, $L(A_i) \subseteq L(A_j)$ and $L(A_j) \subseteq L(A_i)$. Looking forward, state equivalence gives us a method to determine if USLs are equivalent.

Languages accepted by finite-state automata are called regular languages. Given two languages $L(A_i)$ and $L(A_j)$, then the operations shown in Table 4.2 result in some other regular language accepted by automaton A_k [10]. The Kleene-star of a language represents the set of those strings that can be obtained by taking any number of strings from that language and concatenating them.

4.1.3 Finite state transducers

Depending on the problem at hand, finite-state transducers can be interpreted in the following ways [16]. We can view finite-state transducers as finite-state automata with an alphabet $\Sigma = \Sigma_1 \times \Sigma_2$, accepting or rejecting string pairs (a, b) where $a \in \Sigma_1^*$ and $b \in \Sigma_2^*$. This automaton is given by $A = (\Sigma, Q, q_0, \delta, F)$ where $F \subseteq Q$ represents some acceptance condition consistent with the problem at

1. A *regular expression* can also be used to specify all regular languages. Algorithmic procedures exists to convert from a regular expression to a finite state automaton and vice versa.

2. If R is a relation on $S \times S$, then R is reflexive if $(a, a) \in R \ \forall \ a \in S$, R is symmetric if $(b, a) \in R$ then $(a, b) \in R$ and R is transitive if $(a, b) \in R$ and $(b, c) \in R$ then $(a, c) \in R$. An equivalence relation ($\equiv$) is reflexive, symmetric and transitive. Equivalence relations give rise to equivalence classes that contain all and only related members.

Operation	*Representation*
Union	$L(A_k) = L(A_i) \cup L(A_j)$
Intersection	$L(A_k) = L(A_i) \cap L(A_j)$
Difference	$L(A_k) = L(A_i) - L(A_j)$
Complementing	$L(A_k) = \Sigma^* - L(A_j)$
Concatenation	$L(A_k) = L(A_i)L(A_j)$
Kleene-star	$L(A_k) = L(A_i)^*$

TABLE 4.2 – Operations that result in regular languages

hand, and [3]

$$\delta = \{(p, (a, b), q) \mid (p, a, b, q) \in Q \times \Sigma_1^* \times \Sigma_2^* \times Q\} \tag{4.7}$$

We can view finite-state transducers as translators when we consider a class of mappings from strings defined on Σ^* to sets of strings $\mathcal{P}(\Gamma^*)$. This mapping is rational if it can be realized by some finite-state transducer. Assuming that $\delta(q, \sigma)$ may return a set of states and using

$$\hat{\omega}(q, s) = \hat{\omega}(q, \sigma s') = \omega(q, \sigma)\hat{\omega}(\delta(q, \sigma), s') = s'' \tag{4.8}$$

we can define a rational transduction $t : \Sigma^* \to \mathcal{P}(\Gamma^*)$ such that

$$t(s_i) = \{s_o \mid \exists\ \hat{\omega}(q_0, s_i), s_i \in \Sigma^*\} \tag{4.9}$$

and a rational function if $|t(s_i)| \leq 1, \forall s_i \in \Sigma^*(|t(s)|$ represents the number of mappings for an input s.)

Finally, a finite-state transducer can be viewed as computing relations between sets of strings [4]. By analogy with finite-state automata accepting languages if and only if they are regular, transducers accept (compute) relations if and only if they are regular.

Regular relations are defined as $[13] : \{\emptyset\}$ and $\{(a_i, a_j) \mid a_i, a_j \in A\}$, where A is some automata and a are strings. Furthermore, if R_a, R_b and R_c are regular relations, then the following also describe regular relations :

1. $R_a \cdot R_b = \{(a_i b_n, a_j b_m) \mid (a_i, a_j) \in R_a \ \wedge\ (b_n, b_m) \in R_b\}$
2. $R_a \cup R_b = \{(r_i, r_j) \mid (r_i, r_j) \in R_a \ \vee\ (r_i, r_j) \in R_b\}$
3. $\cup_{k=0}^{\infty} R_c^k = \emptyset \cup R_c \cup R_c \cdot R_c \cup R_c \cdot R_c \cdot R_c \ldots$

where R^k are k concatenations of R. By definition, thus, regular relations are closed under the concatenation, union and Kleene-star operations. Operations on regular relations that result in other regular relations are presented in Table 4.3. In contrast to the intersection operation on regular languages, regular relations are generally not closed under intersection. To use an oft-cited example, two regular relations [5] $R_1 = (a^n b^*, c^n)$ and $R_2 = (a^* b^n, c^n)$ relate a regular

3. This can also be seen as a class of directed graphs where states are vertices and labeled edges are given by $E \subseteq Q \times \Sigma_1^* \times \Sigma_2^* \times Q$

4. Any subset of the cross-product (including the empty set) between two sets, $A \times B = \{(a, b) \mid a \in A, b \in B\}$, is a binary relation over $A \times B$

5. where the superscript n denotes n-repetitions of a given symbol, and $*$ denotes an infinite repetition of a given symbol

Operation	*Representation*
Union	$R(M_k) = R(M_i) \cup R(M_j)$
Complementing	Generally not closed
Difference	Generally not closed
Concatenation	$R(M_k) = R(M_i)R(M_j)$
Kleene-star	$R(M_k) = R(M_i)^*$
Intersection	Generally not closed
Cross product	$R(M_k) = L(A_i) \times L(A_j)$
Composition	$R(M_k) = R(M_i) \circ R(M_j)$

TABLE 4.3 – Operations that result in regular relations

language into another regular language, however $R_1 \cap R_2 = (a^n b^n, c^n)$ relates a non-regular language into a regular language.

If we use the output of one machine as the input into another, the machines are then connected in series (or cascade). It is possible to combine the individual machines into one equivalent machine through the composition operation. Using the definitions for the finite-state machines $M_i = (\Sigma, H, Q_i, \delta_i, \omega_i)$ and $M_j = (H, \Gamma, Q_j, \delta_j, \omega_j)$, then we obtain $M_k = (\Sigma, \Gamma, Q_i \times Q_j, \delta_k, \omega_k)$ where,

$$\begin{aligned} &\delta_k((p,q),\sigma) = \{(p^{'}, q^{'}) \mid \delta_i(\sigma, p) = p^{'}, \delta_j(\omega_i(\sigma, p), q) = q^{'}\} \\ &\omega_k((p,q),\sigma) = \{\sigma' \mid \sigma' = \omega_j(\omega_i(\sigma, p), q)\} \end{aligned} \tag{4.10}$$

The input strings $L_I(M) = \{s \in \Sigma^* \mid \exists (s, z) \in R(M)\}$ and output strings $L_O(M) = \{z \in \Gamma^* \mid \exists (s, z) \in R(M)\}$ (and thus the input and output automata) can be retrieved from transducer M through projection operations.

We can also combine machines in parallel. Using definitions for the finite-state machines $M_i = (\Sigma, \Gamma, Q_i, \delta_i, \omega_i)$ and $M_j = (\Sigma, \Gamma, Q_j, \delta_j, \omega_j)$, we obtain $M_k = (\Sigma, \Gamma, Q_i \times Q_j, \delta_k, \omega_k)$ where,

$$\begin{aligned} &\delta_k((p,q),\sigma) = \{(p^{'}, q^{'}) \mid \delta_i(p, \sigma) = p^{'} \ \wedge \ \delta_j(q, \sigma) = q^{'} \ \wedge \ \exists\, \omega_k((p,q),\sigma)\} \\ &\omega_k((p,q),\sigma) = \{\sigma' \mid \omega_i(p, \sigma) = \sigma' \ \wedge \ \omega_j(q, \sigma) = \sigma'\} \end{aligned} \tag{4.11}$$

Parallel connection is associative and commutative and is only defined in case all machines are ϵ-free [6], and the number of resulting states is in practice less than the product of the number of states of the component machines since many states are undefined or unreachable.

4.2 Transformations of Categories

This section is concerned with showing that any category can be transformed into anther category (including itself). Our inputs and outputs are regular languages (the set of all strings recognized by some finite state automaton), and the space of all possible combinations consists of each and every input matched to each and every output. If a machine is presented with any pair of input and output strings and eventually halts in an accepting state, then we have shown

6. Machines that are ϵ-free do not have transitions without a labelling symbol, as would be the case in a spurious transition

that any category can indeed be transformed into any other category. The proof that transformation of a *category* into another category is computable, is based on the construction of an appropriate transducer. To show that a transducer can be created that recognizes $L(A_1) \times L(A_2)$ relations, we use the following reasoning : as our starting point, we use definitions of $L(A_1)$and $L(A_2)$ to define the transducer T. We then show that this transducer has the property that it maps every word in $L(A_1)$ to a word in $L(A_2)$. We do this by showing that T is in a final state if and only if both automata representing $L(A_1)$ and $L(A_2)$ are also in final states. We conclude that a transducer T that computes any valid input/output pair exists, unless $L(A_1)$ or $L(A_2)$ are themselves badly defined.

Theorem 4.2.1. *Given any category $c \neq \emptyset$, there exists a finite state transducer T that maps it to a subset of C_L : $\forall c \in C_L \ \exists T \mid c' = T(c),\ c' \subseteq C_L$. The computability of this mapping follows directly from the existence of the transducer T.*

Démonstration. Representing *categories* as languages (equation 4.6), we need to show that $L(A_1) \times L(A_2)$ relations are recognized by some transducer T. In this case, $L(A_1)$ is the language of the automaton $A_1 = (\Sigma, Q_1, q_1, F_1, \delta_1)$ representing the *category* c and $L(A_2)$ is the language of the automaton $A_2 = (\Sigma, Q_2, q_2, F_2, \delta_2)$ representing the *category* c'. We set $T = (\Sigma \cup \{\epsilon\}, Q_1 \times Q_2, (q_1, q_2), F_1 \times F_2, \delta_T)$. The transition function δ_T is defined as :

$$\delta_T((q_1, q_2), a, b) = \delta_1(q_1, a) \times \delta_2(q_2, b) \tag{4.12}$$

where $q_1 \in Q_1; q_2 \in Q_2; a, b \in \Sigma \cup \{\epsilon\}$ and we need to show that that for any pair (u, v) where $u, v \in \Sigma^*$, T is in a final accepting state $(f_n, f_m) \in F_1 \times F_2$ if and only if machines A_1 and A_2 are in accepting states f_n and f_m respectively. To achieve this, we must show that the following equation holds :

$$\hat{\delta}_T((q_1, q_2), x, y) = \hat{\delta}_1(q_1, x) \times \hat{\delta}_2(q_2, y) \tag{4.13}$$

By induction on the number of transitions (or length of input string if the transducer is deterministic) we have :

Basis : $\hat{\delta}_T((q_1, q_2), \epsilon, \epsilon) = \delta_T((q_1, q_2), \epsilon, \epsilon) = \delta_1(q_1, \epsilon) \times \delta_2(q_2, \epsilon) = \hat{\delta}_1(q_1, \epsilon) \times \hat{\delta}_2(q_2, \epsilon)$;

Induction : $\hat{\delta}_T((q_1, q_2), ua, vb) = \delta_T(\hat{\delta}_T((q_1, q_2), u, v), a, b)$ using equation 4.5 ; $\delta_T(\hat{\delta}_T((q_1, q_2), u, v), a, b) = \delta_T(\hat{\delta}_1(q_1, u) \times \hat{\delta}_2(q_2, v), a, b)$ is the inductive step from equation 4.13 ; $\delta_T(\hat{\delta}_1(q_1, u) \times \hat{\delta}_2(q_2, v), a, b) = \delta_1(\hat{\delta}_1(q_1, u), a) \times \delta_2(\hat{\delta}_2(q_2, v), b)$ using equation 4.12 ; and finally, $\delta_1(\hat{\delta}_1(q_1, u), a) \times \delta_2(\hat{\delta}_2(q_2, v), b) = \hat{\delta}_1(q_1, ua) \times \hat{\delta}_2(q_2, vb)$ using equation 4.5. The transducer T is therefore in accepting state if and only if both A_1 and A_2 are also in accepting states. □

4.3 Transformations of Catsets and USLs

Since catsets are sets of categories and USLs are sets of catsets, we use Theorem 4.2.1 to show that the transformation of catsets and USLs is also computable..

Theorem 4.3.1. *Given any catset $\kappa_i \neq \emptyset$, there exists a finite state transducer T that maps κ_i to some catset κ_o. The computability of this mapping follows directly from the existence of the transducer T.*

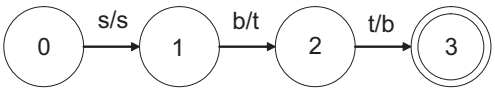

FIGURE 4.2 – Seme exchange (*attribute* ⇔ *mode*) operation for the *category* $\{s, b, t\}$

Démonstration. The transformation of a catset κ_i into a catset κ_o implies that each *category* in the catset $c_i \in \kappa_i$ is transformed into some *category* $c_o \in \kappa_o$. Both c_i and c_o represent languages, and we can represent catsets in terms of languages : $L_\kappa = \bigcup_{n=0,\ c_n \in \kappa}^{m} c_n$. Setting L_{κ_i} and L_{κ_o} to be languages of the catsets κ_i and κ_o respectively, where both L_{κ_i} and L_{κ_o} are subsets of L_{IEML} by equation 2.9, we first obtain two automata recognizing L_{κ_i} and L_{κ_o} and then we follow the same reasoning as in Theorem 4.2.1 to show the existence of a transducer T which encodes the relation between L_{κ_i} and L_{κ_o}. □

Theorem 4.3.2. *Given any USL $u_i \neq \emptyset$, there exists a finite state transducer T that maps u_i to some USL u_o. The computability of this mapping follows directly from the existence of the transducer T.*

Démonstration. To show the existence of a transducer T which performs the mapping $u_o = T(u_i)$, theorem 4.3.1 is used : since a transducer which performs the mapping $\kappa_o = t(\kappa_i)$ exists for every $\kappa_i \in u_i$ and $\kappa_o \in u_o$, and transducers are closed under union operation (see table 4.3), then $T = \bigcup_{n=0}^{6} t_n$, where t_n is the transducer which performs the mapping $\kappa_o^n = t_n(\kappa_i^n)$. □

4.4 Examples of Transformational Operations

This section discusses operations on IEML categories and is grounded in the result of Theorem 4.2.1. Although that theorem ensures that there is a machine for a particular operation, it does not detail how to create that machine. The following examples show machines performing a given operation.

4.4.1 Seme exchange

The following machine performs a simple seme exchange on a *category* :

$$\begin{aligned} &\Sigma = \Gamma = \{s, b, t\} \\ &Q = \{0, 1, 2, 3\} \\ &Q_0 = \{0\} \\ &\Delta = \{(0, s, s, 1), (1, b, t, 2), (2, t, b, 3)\} \\ &F = \{3\} \end{aligned} \tag{4.14}$$

The machine defined by equation 4.14 takes as input a *category* $\{s, b, t\}$ and outputs the $\{s, t, b\}$ *category*, and is shown in figure 4.2.

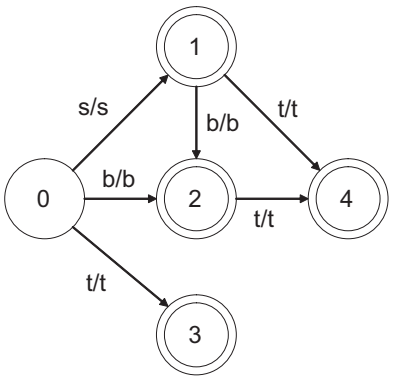

FIGURE 4.3 – Powerset operation for the *category* $\{s, b, t\}$

4.4.2 Powerset operation

Considering the *category* Power Set function, we can readily construct a finite state machine using definition 4.1.2 to represent it :

$$\begin{aligned} &\Sigma = \Gamma = \{s, b, t\} \\ &Q = \{0, 1, 2, 3, 4\} \\ &Q_0 = \{0\} \\ &\Delta = \{(0, s, s, 1), (0, b, b, 2), (0, t, t, 3), (1, b, b, 2), (1, t, t, 4), (2, t, t, 4)\} \\ &F = \{1, 2, 3, 4\} \end{aligned} \tag{4.15}$$

The machine defined by equation 4.15 produces/recognizes the set $\bar{s}$:

$$\bar{s} = \{\{s\}, \{b\}, \{t\}, \{s, b\}, \{s, t\}, \{b, t\}, \{s, b, t\}\} \tag{4.16}$$

, and is shown in figure 4.3. In that figure, the state 0 is the starting state and the double circles represent final - or accepting - states. As an example, if the input to the machine is s while the machine is in the starting state, the machine will output s and move to state 1. In this state it only accepts b and t which move the machine to states 2 and 4 respectively. At this point the machine produced/recognized the set $\bar{s}$:

$$\bar{s} = \{\{s\}, \{s, b\}, \{s, t\}\} \tag{4.17}$$

4.4.3 Partition operation

The partition operation requires a basis, role address and a partitioner. In mathematical terms, the basis is a set S (possibly of other sets), the role address is an expression of which part of any $s \in S$ partition occurs, and the partitioner specifies the exact partition to perform. This operation maps sets of layer l to sets of layer l, $S^l \rightarrow P^l$ such that $s \in S \iff s \in \bigcup_i p_i$ where $p_i \in P$ and $\bigcap_i p_i = \emptyset$.

A finite state machine that represents this operation is a non-deterministic transducer. For example, one partition of a *category*

$$f = \{u, a, s, b, t\} \tag{4.18}$$

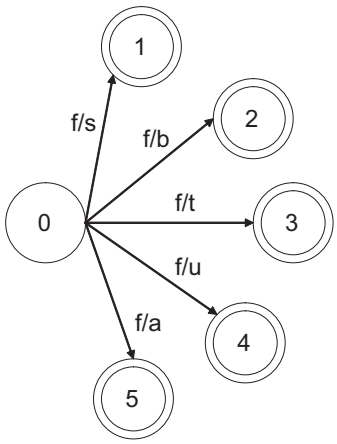

FIGURE 4.4 – Partition operation for the *category* $f = \{u, a, s, b, t\}$

is the set $\{\{s\}, \{b\}, \{t\}, \{u\}, \{a\}\}$ which is represented by the following machine :

$$\begin{gathered} \Sigma = \{s, b, t, u, a\} \\ \Gamma = \{s, b, t, u, a\} \\ Q = \{0, 1, 2, 3, 4, 5\} \\ Q_0 = \{0\} \\ \Delta = \{(0, f, s, 1), (0, f, b, 2), (0, f, t, 3), (0, f, u, 4), (0, f, a, 5)\} \\ F = \{1, 2, 3, 4, 5\} \end{gathered} \tag{4.19}$$

The machine defined by equation 4.19 produces/recognizes the set $\bar{s}$:

$$\bar{s} = \{\{s\}, \{b\}, \{t\}, \{u\}, \{a\}\} \tag{4.20}$$

, and is shown in figure 4.4. In that figure, the state 0 is the starting state and the double circles represent final - or accepting - states.

4.4.4 Rotation operation

The rotation operation requires a basis, role address and a rotor. In mathematical terms, the basis is a set S, the role address is an expression on which part of any $s \in S$ the rotation occurs, and the rotor specifies the exact rotation to perform. In general, this operation maps sets of layer l to sets of layer l, $S^l \rightarrow P^l$.

A finite state machine that represents this operation is a non-deterministic transducer. One rotation of a *category* $\{s, b, t\}$ is the set $\bar{s}$:

$$\bar{s} = \{\{s, s, t\}, \{s, b, t\}, \{s, t, t\}\} \tag{4.21}$$

assuming the rotor is given by $\{\{s\}, \{b\}, \{t\}\}$ and the role address is the attribute, which is represented by the following machine :

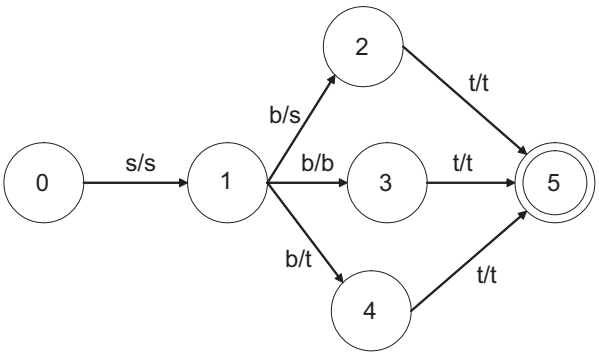

FIGURE 4.5 – Rotation operation for the *category* $\{s, b, t\}$ with $\{\{s\}, \{b\}, \{t\}\}$ rotor

$$\begin{gathered}\Sigma = \Gamma = \{s, b, t\} \\ Q = \{0, 1, 2, 3, 4, 5\} \\ Q_0 = \{0\} \\ \Delta = \{(0, s, s, 1), (1, b, s, 2), (1, b, b, 3), (1, b, t, 4), (2, t, t, 5), (3, t, t, 5), (4, t, t, 5)\} \\ F = \{5\}\end{gathered} \tag{4.22}$$

It is to be noted that the role address will decide where the non-determinism will occur : in the given example, it is on the second state. The machine defined by equation 5.4 produces/recognizes the set $\bar{s}$:

$$\bar{s} = \{\{s, s, t\}, \{s, b, t\}, \{s, t, t\}\} \tag{4.23}$$

and is shown in figure 4.5. In that figure, the state 0 is the starting state and the double circle represents final - or accepting - state.

4.4.5 Supertriplication operation

The supertriplication operation is the union of all applicable triplication operations (see equation 2.3). Once transducers for triplication operations are defined, a union operation on those transducers is performed, which results in another transducer (transducers are closed under union operation, see Table 4.3) The supertriplication operation is thus also represented by a (composite) transducer.

4.4.6 Superselection operation

For any given layer, assigning a particular order to sets, allows us to perform a superselection operation on those sets. In mathematical terms, given two ordered sets A and B where $|A| = |B|$, $A \times B \rightarrow C$, where $\forall c_{ij} \in C$, $a_i \in c_{ij} \wedge b_j \in c_{ij}$. The superselection operation results in a regular IEML matrix.

4.4.7 Matrix concatenation operation

Matrices can be concatenated together and is obtained by the union of sets representing the matrices. Note that this is not the same as a union operation on the underlying sets. Assume that $A \times B \rightarrow C$, $A^{'} \times B^{'} \rightarrow C^{'}$, then in the former case we have $C \cup C^{'} = C^{''}$ whereas in the latter we would obtain $(A \cup A^{'}) \times (B \cup B^{'}) = C \cup C^{'} \cup A \times B^{'} \cup A^{'} \times B$.

Chapitre 5

Relations Model

5.1 Generalities

Relations are modeled with graphs. A graph is a tuple $G = (V, E)$ of two sets, such that E is the set of two-element subsets of $V : E \subseteq [V]^2$. For instance, if $V = \{a, b, c, d\}$ then $[V]^2 = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}$ and E can be represented by $\{\{a, b\}, \{b, c\}\}$. In general, the number of members of the set $[V]^k$ (k-element subsets of V) is a combination of n elements picked k at-a-time without repetition and can be expressed as $\frac{n!}{k!(n-k)!}$.

If the vertices are the *categories* of a set, and the edges represent relations between those *categories*, the resulting graph models the network of relations between *categories* of a particular set. Similarly, if the vertices represent sets of *categories*, and the edges represent relations between those sets, the resulting graph models the network of relations between sets of *categories*.

The number of vertices of a graph is its order and is represented as $|G|$, while the number of edges is represented as $||G||$. Graphs can be finite, denumerable or infinite, depending on its order. Edges connect vertices, and two vertices a, b are said to be adjacent if the edge $ab \in G$.

5.1.1 Relation similarity and nesting

Two graphs are isomorphic if there exists a bijection $\phi : V \to V'$ with $ab \in E \iff \phi(a)\phi(b) \in E'$ for all $a, b \in V$. In essence there must exist a mapping between vertices of the two graphs such that all the edges are preserved. Existence of isomorphism signifies that the structure of a particular relation is similar to the structure of another relation.

A subgraph G' of G is denoted $G' \subseteq G$ and requires that $V' \subseteq V$and $E' \subseteq E$. For a subgraph G', if it contains all edges $ab \in E$ with $a, b \in V'$ then it is an induced subgraph of G and is denoted $G' = G[V']$. Subgraphs efficiently represent nesting of relations.

5.1.2 Connectedness of relations

The degree $d(v)$ of a vertex v is the number of edges at v. The minimum and maximum degree (not order) of a graph is represented as $\delta(G) \triangleq$

$min\{d(v)|v \in V\}$ and $\Delta(G) \triangleq max\{d(v)|v \in V\}$ respectively, while the average degree of a graph is given by : $\frac{1}{|V|}\Sigma_{v \in V} d(v)$. If all the vertices have the same degree, then the graph is regular (e.g. cubic graph, for degree 3).

5.1.3 Hierarchical relations and trees

A relation may represent some hierarchy, which is represented as a *tree*. A graph G not containing any cycles is a forest. If such graph is furthermore connected, then it is called a tree. Some general properties of trees are as follows, and hold for any type of semantic relation that is represented as a tree :

Proposition 5.1.1. *For a graph G, if any two vertices are connected by a unique path, then G is a tree.*

Démonstration. The graph is connected since there is a path between any two vertices. A cycle requires two paths between the same vertices. A unique path between any two vertices in a graph therefore implies that there are no cycles in the graph. □

Proposition 5.1.2. *For a tree T, the path connecting any two vertices of T is unique.*

Démonstration. All vertices are connected, and since there are no cycles in T, the path connecting any two vertices must be unique. □

Proposition 5.1.3. *If a new edge joins two vertices in a tree T, then a cycle is formed.*

Démonstration. Since T is a tree, then there is a unique path from c to u and from c to v, $\forall\, c, u, v \in T$. If a new edge joins u and v, a cycle $c \dots uv \dots c$ is formed. □

Proposition 5.1.4. *A tree T with n vertices has $n-1$ edges.*

Démonstration. (informal) A tree with one vertex has no edges. Adding a second and subsequent vertex to the tree results in the addition of exactly one edge : it cannot be less since the graph would not be connected, and it cannot be more since a cycle would be created. □

5.1.4 Basic relation operations

Union of two graphs G and G' is, by definition, $G \cup G' \triangleq (V \cup V', E \cup E')$, that is, the union of the graphs respective vertices and edges. Similarly, intersection of two graphs is, by definition, $G \cap G' \triangleq (V \cap V', E \cap E')$. Difference between two graphs is obtained by deleting all common vertices and their incident edges . Many other operations on graphs exist. For example, if G and G' are disjoint graphs (where $V \cap V' = \emptyset$), then $G * G'$ is the graph obtained from $G \cup G'$ after joining all vertices of G with all vertices in G'. Another example is the complement on G, denoted $\bar{G}$, which is the graph on V with $\bar{E} = [V]^2 \backslash E$.

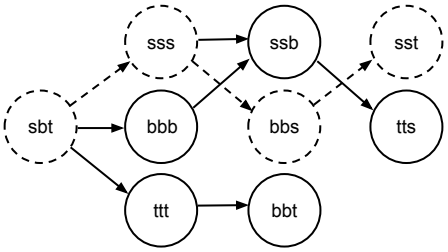

FIGURE 5.1 – Graphical representation of some relation. The highlighted portion is a path between *sbt* and *sst* and can be viewed as a *category* $\{sbt, sss, bbs, sst\}$.

5.1.5 Relation paths

A path is a graph $P = (V, E)$ such that

$$V = \{x_1, x_2, \ldots, x_n\} \tag{5.1}$$

and

$$E = \{x_1x_2, x_2x_3, \ldots x_{n-1}x_n\} \tag{5.2}$$

and can be represented as a sequence of vertices, $x_1x_2 \ldots x_n$. A graph G is connected if any two of its vertices are linked by some path in G. Paths can represent IEML categories (see figure 5.1) and such a representation provides an interesting manner to describe catsets and USLs.

5.1.6 Relation cycles

Given a path $P = x_1x_2 \ldots x_{n-1}$ where $n \geqslant 3$ then the graph $C = P + x_{n-1}x_1$ is a cycle (two edges connecting the same vertices in an undirected graph is treated as one edge). The minimum and maximum length of a cycle contained in a graph G are called girth and circumference respectively. In the case of a tree (see section 5.1.3), the girth is set to ∞ and the circumference to 0.

5.1.7 Relation distances

The distance between two vertices x and y of a graph G, $d_G(x, y)$ is the length of the shortest path between x and y. If no such path exists, the distance is ∞.

The eccentricity of any vertex x is the maximum distance between x and any other vertex in the graph G. A graph's diameter is given by the maximum of its vertice eccentricities, $diam_G = max_{y \in V(G)}(d_G(x, y))$ and its radius by the minimum of its vertice eccentricities, $rad_G = min_{x \in V(G)}(diam_G)$. A vertex in a graph whose distance to any other vertex is less or equal to the graph's radius is central in that graph. Radius and minimum degree of a graph that represents any type of semantic relation can be used for classification purposes. For example, a graph G of radius at most k and of degree $d \geq 3$ representing any type of semantic relation has less than $\frac{d}{d-2}(d-1)^k$ vertices.

Démonstration. Let z be a central vertex in G and D_i the set of vertices of G at a distance of i from z. The total number of vertices in G is given by $V(G) = \bigcup_{i=0}^{k} D_i$. The following is true: for $i = 0$, $|D_0| = 1$ since it contains only the vertex z, for $i = 1$, $|D_1| \leqslant d$ since it cannot exceed the graph's maximum degree. For $i \geqslant 1$, the following holds, $|D_{i+1}| \leqslant (d-1)|D_i|$ since each vertex in D_{i+1} shares at least one edge with a vertex in D_i. We obtain $|D_{i+1}| \leqslant d(d-1)^i, \forall i < k$. The number of vertices in G is therefore $|V(G)| \leqslant \sum_{i=0}^{k} |D_i| = 1 + \sum_{i=0}^{k-1} |D_{i+1}|$. The sum $s_k = \sum_{i=0}^{k-1}(d-1)^i$ can be rewritten as $s_k = 1 + (d-1) + (d-1)^2 + \ldots + (d-1)^{k-1}$ and can be subtracted from $s_k(d-1)$ which gives $s_k(d-1) - s_k = (d-1)^k - 1$. Solving for s_k gives $\frac{(d-1)^k - 1}{d-2}$. Replacing this in the previous equation, we obtain $|V(G)| \leqslant 1 + d\frac{(d-1)^k - 1}{d-2} < \frac{d}{d-2}(d-1)^k$ since $\frac{d}{d-2} > 1$ □

5.2 Relations and Semantic Graphs

Semantic graphs are a representation of the semantic relations found between IEML expressions : categories, catsets and USLs.

5.2.1 Linear order relations

Any graph $G = (V, E)$ that is a tree (see section 5.1.3) and where $\forall v \in V$, the degree (see section 5.1.2) always conforms to $0 < d(v) \leq 2$, describes a path (see section 5.1.5). The edges E of those graphs define a linear-order relation between *categories*, catsets and USLs. Many such graphs can exist, depending on the exact ordering criteria.

5.2.2 Set-subset relations

This type of relation occurs only between *categories* of the same layer. In the general case all *categories* that are in a set-subset relation are given by $C_L^r \subseteq C_L \times C_L$ where C_L^r is given by :

$$C_L^r = \{\{c_i, c_j\} \mid c_i, c_j \in C_L, c_i \subseteq c_j\} \tag{5.3}$$

The graph $G = (V, E)$ is given by :

$$\begin{aligned} V &= C_L \\ E &= C_L^r \end{aligned} \tag{5.4}$$

Set-subset relations can also be constructed for particular cases by obtaining the powerset of the *category* of interest which defines vertices V, and applying the formula in equation 5.4. A representation for the *category* $\{sbt, sbb, sss\}$ is shown in figure 5.2.

Set-subset relations are also applicable to catsets and USLs : two different catsets are in a set-subset relation if and only if there exists a *category* in both catsets that are in a set-subset relation ; two different USL are in a set-subset relation if and only if there exists a catset in both USLs that are in a set-subset relation.

In the general case, all catsets that are in a set-subset relation are given by $\kappa_L^r \subseteq C_L \times C_L$ where κ_L^r is given by :

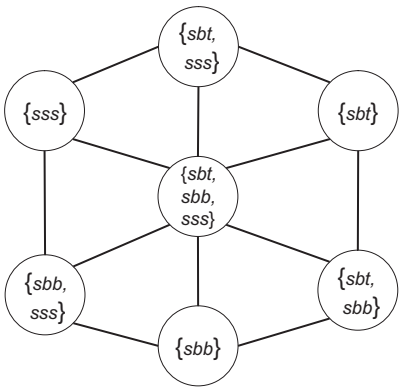

FIGURE 5.2 – Graphical representation of set-subset relation for the *category* $\{sbt, sbb, sss\}$.

$$\kappa_L^r = \{\{\kappa, \kappa_j\} \mid \exists c_i \in \kappa_i, \exists c_j \in \kappa_j, \{c_i, c_j\} \in C_L^r\} \tag{5.5}$$

Similarly, all USLs that are in a set-subset relation are given by $\mathcal{USL}^r \subseteq \mathcal{USL} \times \mathcal{USL}$ where $\mathcal{USL}^r$ is given by :

$$\mathcal{USL}^r = \{\{u_i, u_j\} \mid \exists \kappa_i \in u_i, \exists \kappa_j \in u_j, \{\kappa_i, \kappa_j\} \in \kappa_L^r\} \tag{5.6}$$

5.2.3 Symmetric relations

This type of relation occurs between *categories* c and c' of the same layer and of the same cardinality (*categories* must contain the same number of sequences), whenever there exists an automaton (see section 4.1.1) A which recognizes some sub-sequence $s \in L_{IEML}$ in both c and c', and a transducer (see section 4.1.3) T which computes some relation cTc' (transforms c into c'). The automaton A describes a similarity at the level of symbolic arrangement (or syntactic level) and is a necessary condition for the assertion of a semantic symmetry, while the transducer T describes some semantic invariance combined to a semantic variation. Both the A and T machines can be derived from the specific *directory* (see section 6.4.5) containing categories of interest.

Catsets are in a symmetric relation if and only if there exists a *category* in both catsets that are in a symmetric relation and USLs are in a symmetric relation if and only if there exists a catset in both USLs that are in a symmetric relation.

5.2.4 Etymological relations

An etymologic relation is in general a relation between a *category* at layer n and one of its semes at layer n–n' where $1 \leq n' < n$. Depending on the directory containing categories of interest, an etymological relation may or may not be present. The graph representation of these relations has a tree structure.

Chapitre 6

Semantic Circuits

6.1 Paradigmatic Characters and Sequences

Using the alphabet defined by equation 2.17, we distinguish ten symbols in $\bar{\Sigma}$: $\{T\}$, $\{B\}$, $\{S\}$, $\{A\}$, $\{U\}$, $\{E\}$, $\{S, B, T, U, A, E\}$, $\{S, B, T, U, A\}$, $\{S, B, T\}$ and $\{U, A\}$ which are a subset of the category language alphabet (see equation 2.17). These symbols, and only these symbols are called paradigmatic characters and form the paradigmatic alphabet $\bar{\Sigma}_P$. Usage of the $\bar{\Sigma}_P$ alphabet allows us to differentiate paradigmatic sequences among the sequences of the category language (see section 2.3.1). IEML sequences are obtained from paradigmatic sequences by performing a Cartesian product between all the sets of the paradigmatic sequence. For instance, the paradigmatic sequence $\{S, B, T\}\{U, A\}\{E\}$ gives the IEML set of sequences $\{SUE, SAE, BUE, BAE, TUE, TAE\}$. Using the definition in 2.1, paradigmatic sequences for $0 \leq l \leq 6$ are given by the general expression :

$$L^P_{IEML} = \{s \in \bar{\Sigma}^*_P \mid |s| = 3^l\} \tag{6.1}$$

where $L^P_{IEML} \subseteq L_{IEML}$.

6.2 Paradigmatic Distance

Using Equation 3.2 we consider a set of functions $F = \{f_1, f_2, \cdots, f_n\}$ where

$$\forall f \in F,\ f : \mathcal{USL} \rightarrow \mathcal{USL} \tag{6.2}$$

and a directed graph $G_p = (\mathcal{USL}, E)$ where :

$$E = \{(\mu_i, \mu_o) \mid (\mu_i, \mu_o) \in [\mathcal{USL}]^2 \ \wedge\ \mu_o = f(\mu_i),\ f \in F\} \tag{6.3}$$

The *paradigmatic path* between any two nodes $a, b \in \mathcal{USL}$ of the graph G_p is a path on G_p (see Section 5.1.5), which leads to the following definition of a *paradigmatic distance :*

Definition 6.2.1. *The shortest path between any two vertices a, b of G_p is the paradigmatic distance between a and b.*

The shortest path can be calculated by algorithms presented in Section 7.2.

6.3 Semantic circuits

Semantic circuits S_ω are labeled and directed graphs $G = (V, E)$ where $V \subseteq L_{IEML}$ and all $e \in E$ are a ternary relation (s, m, d) where $(s, d) \in [V]^2$ and $m \in L_{IEML}$. We can show that the set of all semantic circuits O_c forms a *groupoid,* capturing symmetries described in section 3.4.3 by the following proof:

Démonstration. The binary permutation operation $\otimes$ is partial [1] implying that it is defined for some, but not all, members of O_c. The following properties hold :

- $\forall a, b, c \in O_c,\ (a \otimes b) \otimes c \to a \otimes (b \otimes c)$
- $\forall a \in O_c,\ \exists a^{-1} \in O_c,\ a \otimes a^{-1} = i_a$
- $\forall a, b \in O_c,\ a \otimes b \to a \otimes b \otimes b^{-1} = a$
- $a \otimes a^{-1} \otimes b = b\,; \forall a \in O_c,\ a \otimes a^{-1} \in O_c$

These properties are necessary and sufficient to characterize a groupoid. □

A particularly important subset of semantic circuits is the subset that preserves the category (see section 3.4) structure given by equation 3.3. Consider a subset of O_c such that $\forall a \in O_c^l \subset O_c,\ s = m = d = 3^l,\ 0 \leq l \leq 6$. The collection of O_c^l for $0 \leq l \leq 6$, together with a collection of morphisms $M_c = \{m_c \mid m_c = (o_c^l \to o_c^{l+1});\ 0 \leq l \leq 5\}$ and the triplication function from equation 2.3 applied to (s, m, d) form the *category* $\mathbb{S}$. We then can show that the functor F from *category* $\mathbb{C}$ to $\mathbb{S}$ is a mapping from $\mathbb{C}$ to $\mathbb{S}$ such that : $a_c \in O_c :\ F(a_c) \in O_s\,; (x_c \to y_c) \in M_c :\ (F(x_c) \to F(y_c)) \in O_s$ where $\forall a \in O_c,\ F(1_a) = 1_{F(a)}$ and $\forall a, b \in M_c,\ F(a \circ b) = F(a) \circ F(b)$. This is fundamental since it states that the *category* of IEML language can be mapped to the *category* of semantic circuits.

6.4 Rhizomes

A rhizome $G_\rho(V, E)$ is a type of semantic circuit where $V = \bigcup_i V_i$ and $\forall m, n \in V_i,\ \exists (m, n) \in E \wedge \exists (n, m) \in E$. This type of semantic circuit is thus composed of fully-connected subgraphs, or *cliques.*

6.4.1 Serial rhizome

A serial rhizome is a rhizome $G_s = (V_s, E_s)$ with the following properties : G_s is a path (see section 5.1.5), $V_s \subseteq L_{IEML}^P$, $\forall v_i, v_j, v_k \in V_s$, $|v_i| = |v_j|$, and there is a binary relation $\leq$ on V_s such that : $v_i \leq v_j \wedge v_j \leq v_i \to v_i = v_j$, $v_i \leq v_j \wedge v_j \leq v_k \to v_i \leq v_k$, $v_i \leq v_j \vee v_j \leq v_i$.

6.4.2 Etymological rhizome

A etymological rhizome is a rhizome $G_e = (V_e, E_e)$ with the following properties : G_e has a tree structure (see section 5.1.3), $V_e \subseteq L_{IEML}^P$, $\forall e_i, e_j \in E_e$, $e_i \neq e_j$, $e_i \in$ `substance` $\cup$ `attribute` $\cup$ `mode` of equations 2.6, 2.7 and 2.8.

1. Weisstein, Eric W. "Partial Function." From MathWorld, A Wolfram Web Resource. http ://mathworld.wolfram.com/PartialFunction.html

6.4.3 Taxonomic rhizome

A taxonomic rhizome is a rhizome $G_t = (V_t, E_t)$ with the following properties : G_t is a tree (see section 5.1.3), $V_t \subseteq L^P_{IEML}$ and $\forall e_i \in E_t,\ e_i = \{(v_i, v_j) \mid \forall j, v_i = \bigcup_j v_j \ \wedge \ \bigcap_j v_j = \emptyset\}$.

6.4.4 Paradigms

A paradigm $\mathbb{P}$ is the result of a union, intersection or symmetric difference of taxonomic, etymological or serial rhizomes (see section 5.1.4).

6.4.5 Dictionary

A dictionary $\mathbb{D}$ is a paradigm where vertices have a one-to-one mapping to natural languages.

6.5 Rhizomatic Operations

Algorithm 1 presents the general form of rhizomatic operations. The input to the algorithm is a paradigmatic rhizome of degree n [2], and the output is a paradigmatic rhizome of degree $n + 1$. The following terms are used in the algorithm and are listed with explanatory notes :

- $f(u)$ is one of the operations in section 4.4,
- `cast` promotes one parameter to the degree of the other parameter.

The complexity of algorithm is linear and proportional to the number of vertices in the input rhizome. Purpose-designed algorithms for a specific operation from section 4.4 may exhibit better performance characteristics.

6.6 Dialectic Function

A dialectical function $\mathbb{z}_d : S_\omega \times USL \rightarrow \mathbb{P}$ (see section 4.4) returns a paradigm given a semantic circuit and an USL.

6.7 Paradigmatic Function

A paradigmatic function $\mathbb{z}_p : S_\omega \times USL \rightarrow S_\omega$ (see section 6.3) returns a semantic circuit given a semantic circuit and an USL.

6.8 Syntagmatic Function

A syntagmatic function $\mathbb{z}_s : \mathbb{D} \times USL \rightarrow G_s$ (see section 6.4) returns a rhizome given a dictionary (see section 6.4.5) and an USL. This function is composed of sub-functions described in following sections.

2. A rhizome of degree n means that the longest path in its graph $G_\rho(V, E)$ is given by a sequence of n+1 nodes (see section 5.1.5).

Algorithme 1 : rhizomatic

input : Paradigmatic rhizome of degree n, $G_s^n(V, E)$ (see section 6.4)
output : Paradigmatic rhizome of degree $n+1$, $G_s^{n+1}(V, E)$

begin
  for *all* i **do**
1 $V_i \longrightarrow U \mid |U| = |V_i| - 1$;
    for *all* $u, w \in U$ **do**
      if $|u| \neq |w|$ **then**
2 $u \longleftarrow$ `cast`$(|w|)$;
      if $f \in$ *triplication (see equation 2.3)* **then**
3 $|U^*| = |u| + 1$;
4 $f(u) \longrightarrow U^*$;
      for *all* $x \in U^*$ **do**
5 $E_i \longleftarrow (u, x), (x, u)$;
        for *all* $y \in U^* \backslash x$ **do**
6 $E_i \longleftarrow (x, y), (y, x)$;
7 $G_s(V, E) \longleftarrow G(V, E)$;
end

6.8.1 Morphogenetic function

Morpheme creation is implemented by Algorithm 2 and based on a set of strings S_n where $S_n = \{s \mid s \in L_{cat}\}$ (see equation 2.19). This algorithm is recursive and takes as input $\mathbb{D}$ a state of the dictionary represented by S_1, X the *category* representedby S_2 and outputs M, the set of morphemes represented by S_3. Note that given $\mathbb{D}$ and X, there is only one M. The following functions are used :

- `break` which exits the enclosing loop,
- `max` which returns the longest string,
- `cut` which returns the remaining portion, or portions, of the string x after the substring x^* is removed from it. This function can be implemented using standard regular expressions [3],
- `comp` which compares substrings and returns the longest common substring [4].

The complexity of the algorithm is polynomial in the size of the set X.

6.8.2 Propositional genealogy function

Propositional genealogies are constructed by Algorithm 3. This algorithm takes as the input the *output* of algorithm 2 and produces a semantic circuit representing the syntagmatic structure of a proposition (propositional genealogy). The following functions are used :

- `group` method which groups together morphemes of same layer l,
- Algorithm 1 to create rhizomes of successive degree.

3. http ://en.wikipedia.org/wiki/Regular_expression
4. http ://en.wikipedia.org/wiki/Longest_common_substring_problem

Algorithme 2 : morphogenetic

```
   input  : Set of sequences 𝔻 and X
   output : Set of sequences M
   begin
1      |x*| ⟵ 0;
       for all x ∈ X do
           if |x| = 0 then
2              break;
           else
               for all d ∈ D do
3                  x* ⟵ max(x*, comp(x, d));
                   if |x*| ≠ 0 then
4                      break;
               if |x*| ≠ 0 then
5                  M ⟵ x*;
6                  morphogenetic(cut(x, x*), D);
   end
```

The complexity of the algorithm is linear in the size of the input set.

Algorithme 3 : propositional genealogy

```
   input  : Set of sequences M (see algorithm 2)
   output : Graph G_s = (V, E) (see section 6.4)
   begin
1      M_l ⟵ group(M, l);
       for all l do
2          V ⟵ ⋃(∀m ∈ M_l);
           for all k = |M_l − 1| do
3              E ⟵ {(a, b) | a ≠ b, a, b ∈ M_l};
4      G_s^0(V, E) ⟵ rhizomatic(V, E) (see algorithm 1);
       for i = 1...7 do
5          G_s^i(V, E) ⟵ rhizomatic(G_s^{i−1}(V, E));
6          G_s(V, E) ⟵ G_s(V, E) ∪ G_s^i(V, E)) (see section 5.1.4);
   end
```

6.8.3 Capillarity construction function

Construction of capillarities [5] is performed by Algorithm 4. This algorithm takes as input the *output* of Algorithm 3 and its output is a rhizome. It uses the following functions :

– `subgraph` method which finds all graphs of degree n,

5. Given a root node, capillarities are the relations between all the leaves of that root node.

– `fill` method which alternates source and destination of words, phrases, etc.,
– `isunivoque` determines if a clause/period is univoque or not,
– `clique` creates a complete graph respectively.

The complexity of the algorithm is polynomial in the size of the input graph $G_i(V, E)$.

Algorithme 4 : capillarity construction

input : Degree n, Graph $G_i(V, E)$ (see section 6.4)
output : Graph $G_o(V, E)$
begin
1 $G^n \longleftarrow \texttt{subgraph}(G_i(V, E))$;
for *all* $g \in G^n$ **do**
2 $E \longleftarrow E : g(V, E)$;
if $n = 0$ **then**
for *all* $v \in V$ **do**
if $v \neq \emptyset$ **then**
3 $E \longleftarrow \{(a, \emptyset, b) \mid a \neq b, a, b \in V\}$;
else
if $n = 1$ **then**
for *all* $v, w \in V$ **do**
if $v \in \texttt{mode} \wedge (w \in \texttt{substance} \vee w \in \texttt{attribute})$ **then**
4 $E \longleftarrow \{(a, \emptyset, b) \mid a \neq b, a, b \in V\}$;
else
for *all* $v \in V$ **do**
if $v = \texttt{isunivoque}$ **then**
5 $v_T \longleftarrow \texttt{substance}(v) \wedge \texttt{attribute}(v)$;
6 $v_A \longleftarrow \texttt{mode}(v)$;
7 $E \longleftarrow \texttt{fill}\{(v_T, v_A, v_T)\}$;
else
8 $v_T \longleftarrow \texttt{clique}(v)$;
9 $v_A \longleftarrow \emptyset$;
10 $E \longleftarrow \texttt{fill}\{(v_T, v_A, v_T)\}$;
11 $G_o(V, E) \longleftarrow g \cup G_o(V, E)$;
end

Chapitre 7

Quantitative Criteria for Semantic Circuits

7.1 Structural Similarity Criterion

7.1.1 Generalities

Definition 7.1.1. *Matrix* $\boldsymbol{A}$ *is a 2 dimensional array of objects of the same class over a field*[1] Ψ *:*

$$\boldsymbol{A} = [a_{ij}] = \begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1N} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{iN} \\ \vdots & & \vdots & & \vdots \\ a_{M1} & \cdots & a_{Mi} & \cdots & a_{MN} \end{bmatrix}, \tag{7.1}$$

$$\forall \left\{ i = \overline{1 \dots N}, j = \overline{1 \dots M}; M, N \in \mathbb{Z} \right\} : a_{ij} \in \Psi \therefore \boldsymbol{A} \in \Psi^{N \times M}, \tag{7.2}$$

where M, N *are called matrix dimensions.*

As long as the matrix entries a_{ij} have operations of addition and multiplication defined on field Ψ, we can define matrix addition and multiplication over matrices of compatible dimensions as follows.

Definition 7.1.2. *For* $\boldsymbol{A} \in \Psi^{N \times M}$ *and* $\boldsymbol{B} \in \Psi^{N \times M}$ *:*

$$\boldsymbol{A} + \boldsymbol{B} = [a_{ij} + b_{ij}]. \tag{7.3}$$

Definition 7.1.3. *For* $\boldsymbol{A} \in \Psi^{N \times M}$, $\boldsymbol{B} \in \Psi^{M \times K}$ *and* $\boldsymbol{C} \in \Psi^{N \times K}$, $\boldsymbol{C} = \boldsymbol{AB}$ *means that*

$$[c_{ij}] = \left[\sum_{m=1}^{M} a_{im} b_{mj} \right]. \tag{7.4}$$

1. A field is a ring (see 3.1) equipped with the operations of addition, substraction, multiplication and division that satisfy the axioms of closure, associativity, commutativity, identity, inverse, and distributivity [17].

It can be shown that matrix multiplication defined this way satisfies the usual multiplication properties of associativity [19] :

$$(\mathbf{AB})\,\mathbf{C} = \mathbf{A}\,(\mathbf{BC})\,, \tag{7.5}$$

and both left and right distributivity :

$$(\mathbf{A}+\mathbf{B})\,\mathbf{C} = \mathbf{AC}+\mathbf{BC} \tag{7.6}$$

$$\mathbf{C}\,(\mathbf{A}+\mathbf{B}) = \mathbf{CA}+\mathbf{CB}. \tag{7.7}$$

As for the commutativity, even if $\mathbf{BA}$ exists, in general case commutativity does not hold [18] :

$$\mathbf{AB} \neq \mathbf{BA}. \tag{7.8}$$

One can also define another two useful concepts : identity matrix and matrix inverse.

Definition 7.1.4. *An identity matrix $\boldsymbol{I}$ is a matrix that satisfies Proposition 3.1.2 :*

$$\forall \boldsymbol{A} \in \Psi^{N\times M} \exists \boldsymbol{I} \in \Psi^{M\times M} : \boldsymbol{AI} = \boldsymbol{A} \tag{7.9}$$

Lemma 7.1.1. *If Ψ is equipped with a scalar null element $\mathbb{0}$:*

$$\forall a \in \Psi\ \exists \mathbb{0} \in \Psi : a + \mathbb{0} = a \wedge \mathbb{0} \cdot a = \mathbb{0}, \tag{7.10}$$

and a scalar identity element $\mathbb{1}$:

$$\forall a \in \Psi\ \exists \mathbb{1} \in \Psi : \mathbb{1} \cdot a = a, \tag{7.11}$$

the identity matrix $\boldsymbol{I} = [\iota_{ij}] \in \Psi^{M\times M}$ for a matrix $\boldsymbol{A} \in \Psi^{N\times M}$ can then be computed as

$$\iota_{ij} = \begin{cases} \mathbb{1} & \Leftrightarrow i = j \\ \mathbb{0} & \Leftrightarrow i \neq j \end{cases} \tag{7.12}$$

Démonstration. Substituting properties (7.10) and (7.11) into equation (7.4) according to rule (7.12), we obtain

$$\mathbf{AI} = \left[\sum_{m=1}^{M} a_{im}\iota_{mj}\right] = [a_{ij}\iota_{jj}] = [a_{ij}\cdot\mathbb{1}] = [a_{ij}] = \mathbf{A} \tag{7.13}$$

□

It is trivial to also show that identity matrix multiplication is one of those special cases where commutativity property does hold as long as the underlying scalar multiplication on Ψ is commutative :

$$\mathbf{AI} = \mathbf{IA} = \mathbf{A}. \tag{7.14}$$

For the sake of simplicity, we will only define inverse of a square matrix $A \in \Psi^{N\times N}$. For a generalized treatment, the interested reader can be referred to the Moore-Penrose pseudoinverse in e.g. [9].

Definition 7.1.5. *Matrix* $\boldsymbol{A}^{-1} \in \Psi^{N\times N}$ *is called an inverse of* $\boldsymbol{A} \in \Psi^{N\times N}$ *if the following property holds :*

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}. \tag{7.15}$$

If $\boldsymbol{A}^{-1}$ *does not exist,* $\boldsymbol{A}$ *is called a singular matrix.*

Another useful concept in matrix algebra is a permutation matrix that allows to identify matrices whose differences can be expressed as permutations of their columns and rows.

Definition 7.1.6. *Matrix* $\boldsymbol{P}$ *is called a* permutation *matrix if it satisfies the following conditions :*

1. $\boldsymbol{P}$ *is a square binary matrix :* $\boldsymbol{P} \in \mathbb{B}^{N\times N}$, $\mathbb{B} := \{0, 1\}$.
2. $\boldsymbol{P}$ *has exactly one identity entry per each row and each column and null entries elsewhere :* $\forall i, j = \overline{1 \dots N} : \sum_{i=0}^{N} p_{ij} = 1 \wedge \sum_{j=0}^{N} p_{ij} = 1$.

7.1.2 Adjacency matrix

Matrices provide a convenient representation for the manipulation and study of graphs. One of such matrices is the *adjacency matrix* of a graph representing its connectivity [5].

Definition 7.1.7. *For a graph* $G = (V, E)$, $|E| = N$ *its adjacency matrix is a unique matrix* $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, *where* $\mathbb{R}$ *is the set of real numbers. For a simple graph* G *adjacency matrix* $\boldsymbol{A}$ *is computed the following way :*

$$a_{ij} = \begin{cases} |v_i v_j| & \Leftrightarrow i \neq j \\ \kappa |v_i v_i| & \Leftrightarrow i = j \end{cases} \tag{7.16}$$

where $|v_i v_j|$ *is the number of edges from vertex* i *to vertex* j, *and* $\kappa = 1$ *for directed graphs and* $\kappa = 2$ *otherwise.*

7.1.3 Graph isomorphism

Graphs that have the same structure and differ only in insignificant details, such as numbering on vertices and edges, are studied using the notion of *isomorphism* [7].

Definition 7.1.8. *Two graphs* G_1 *and* G_2 *with adjacency matrices* $\boldsymbol{A}_1$ *and* $\boldsymbol{A}_2$ *are called* isomorphic *if and only if there exists a permutation matrix* $\boldsymbol{P}$ *such that*

$$\boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^{-1} = \boldsymbol{A}_2 \tag{7.17}$$

Such definition of isomorphism naturally lends itself to testing if a particular graph transformation is isomorphic. If the transform operator generates a matrix that satisfies Definition 7.1.6, then the matrix is a permutation matrix and condition (7.17) is satisfied automatically. On the other hand, proving a negative proposition (i.e. that two graphs G_1 and G_2 are not isomorphic) requires a proof of a negative proposition that there exist no such permutation that transforms G_1 into G_2. A solution to this dilemma will be provided below in Section 7.1.4 (Lemma 7.1.3).

7.1.4 Graph spectral theory

Definition 7.1.8, although mathematically rigorous, is not very practical. More practical ways of studying graph isomorphisms, among other graph properties, are one of many applications of the spectral graph theory. Spectral graph theory studies properties of graphs in relation to the eigenvalues [9] of the matrices that completely describe the graph, such as the adjacency matrix, the distance matrix, or admittance matrix (also called graph Laplacian, [19]).

Definition 7.1.9. *A scalar $\lambda \in \mathbb{C}$ is an* eigenvalue *of a square matrix $\boldsymbol{A}$ if it satisfies the following equation :*

$$(\boldsymbol{A} - \lambda \boldsymbol{I})\, \boldsymbol{q} = 0, \tag{7.18}$$

where $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, $\boldsymbol{I} \in \mathbb{R}^{N\times N}$ and $\boldsymbol{q} \in \mathbb{C}^{N\times 1}$. The column vector $\boldsymbol{q}$ associated with that eigenvalue is called an eigenvector*.*

In scalar terms, system (7.18) has N equations for $N+1$ unknowns, so its solution $(\lambda, \mathbf{q})$ is not necessarily unique. It rather has up to N distinct solutions which form multisets $\{\lambda_1, \ldots, \lambda_N\}$ and $\{\mathbf{q}_1, \ldots, \mathbf{q}_N\}$.

Definition 7.1.10. *A multiset of eigenvalues $\{\lambda_1, \ldots, \lambda_N\} \in \mathbb{C}$ of a matrix $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ is called* spectrum *of $\boldsymbol{A}$.*

Lemma 7.1.2. *If $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ is symmetrical, then all of its eigenvalues are real numbers :*

$$\forall (i,j) = \overline{1 \ldots N} : a_{ij} = a_{ji} \;\Leftrightarrow\; \{\lambda_1, \ldots, \lambda_N\} \in \mathbb{R} \tag{7.19}$$

and have an orthogonal set of eigenvectors :

$$\forall (i,j) = \overline{1 \ldots N} : \left\langle \boldsymbol{q}^{(i)}, \boldsymbol{q}^{(j)} \right\rangle \neq 0 \Leftrightarrow i = j \tag{7.20}$$

Corollary 7.1.1. *Undirected graphs have real spectra (immediately follows from (7.16)).*

Definition 7.1.11. *Two graphs G_1 and G_2 are called* isospectral (cospectral) *if and only if they have identical spectra $\Lambda(G_1) = \Lambda(G_2)$.*

Lemma 7.1.3. *Non-isospectral graphs are necessarily not isomorphic :*

$$\Lambda(G_1) \neq \Lambda(G_2) \Rightarrow \nexists \boldsymbol{P} : \boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^{-1} = \boldsymbol{A}_2 \tag{7.21}$$

Matrix spectra can be efficiently computed using the QR algorithm for relatively small matrices or the Lanczos algorithm for large sparse matrices [9]. Together with the result of Lemma 7.1.3 this provides a practical way of testing for the absence of isomorphism between different graphs.

Spectral graph theory is currently a very active area of research in mathematics, particularly for undirected graphs. Interested reader can be further referred to [6] for an extensive review of the field.

7.2 Shortest Path Criterion

7.2.1 Shortest path problem for unweighted graphs

In exploring graphs it is often interesting to compute the length of a shortest path between a certain pair of vertices or, more generally, between all pairs of vertices.

Definition 7.2.1. *A graph $G = (V, E)$ is called* unweighted *if distances between every pair of adjacent vertices (i, j) are set to be the same $\forall(i, j) = \overline{1 \dots N} : d_{ij} = const$. Otherwise, we will call the graph* weighted.

Without loss of generality, it is often convenient to assume the basic distance in an unweighted graph to be 1.

For unweighted graphs, this problem can be solved in a rather elegant way by computing power series $\mathbf{A}^n$ of the adjacency matrix. $\mathbf{A}^n$ has an interesting property as its entries $a_{ij}^{(n)}$ are equal to the number of paths of length n between vertices i and j [5]. The following algorithm computes the *distance matrix* $\mathbf{D} \in \mathbb{Z}^{N \times N}$ containing distances d_{ij} between vertices i and j.

Algorithme 5 : Computing distance matrix for unweighted graphs

```
   input  : Adjacency matrix A ∈ Z^{N×N}
   output : Distance matrix D ∈ Z^{N×N}
   begin
1      B ← I_N ∈ Z^{N×N} ;
2      D ← ∞_N ∈ Z^{N×N} ;
       for n = 1 to N do
3          C ← BA ;
           for i = 1 to N do
               for j = 1 to N do
                   if c_ij > 0 and b_ij = 0 then
4                      d_ij ← n ;
   end
```

It follows rather obviously that Algorithm 5 has computational complexity of N matrix multiplications or $\mathrm{O}(N^3)$. It works for both directed and undirected graphs and computes a complete set of all-pairs shortest path lengths measured in terms of number of steps between the vertices.

7.2.2 Generalized shortest path problem

Semantic circuits (Section 6.3), however can be much more accurately represented by weighted graphs where weights l_{ij} are assigned in an inversely proportional way to the importance of connection between particular USLs. In order to solve the all-pairs shortest path problem for the weighted graphs let us define the fundamental distance matrix $\mathbf{D}^{(0)} \in \mathbb{R}^{N \times N}$ containing lengths of all the paths of the first order, where arc denotes a function that checks if two vertices are adjacent.

Definition 7.2.2.

$$\forall(i,j) = \overline{1\dots N} : d_{i,j}^{(0)} := \begin{cases} 0 & \Leftarrow i = j \\ l_{ij} & \Leftarrow \exists\ arc(i,j) \\ \infty & \Leftarrow \nexists\ arc(i,j) \end{cases} \tag{7.22}$$

The following iterative algorithm [2] is attributed to Floyd and Warshall and is widely used in numerous applied fields, such as comparative genetics, circuit analysis and operations research [15].

Algorithme 6 : Computing distance matrix for weighted graphs

input : $\mathbf{P}^{(0)} \in \mathbb{R}^{N\times N}$, $\mathbf{D}^{(0)} \in \mathbb{R}^{N\times N}$ according to (7.22)
output : $\mathbf{D}^{(N)}$, $\mathbf{P}^{(N)}$
begin
  for $i,j = \overline{1\dots N}$ **do**
    if $i = j$ **then**
1     $p_{ij}^{(0)} \leftarrow$ NaN (IEEE not-a-number value) ;
    else
2     $p_{ij}^{(0)} \leftarrow i$;
3 $k \leftarrow 1$;
  repeat
    for *all* $i,j = \overline{1\dots N}$ **do**
4       $d_{ij}^{(k)} \leftarrow \min\left[\left(d_{ij}^{(k-1)}\right), \left(d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)\right]$;
        if $d_{ij}^{(k)} \neq d_{ij}^{(k-1)}$ **then**
5         $p_{ij}^{(k)} \leftarrow p_{kj}^{(k-1)}$;
        else
6         $p_{ij}^{(k)} \leftarrow p_{ij}^{(k-1)}$;
7     $k \leftarrow k + 1$;
  until $k > N$;
end

On completion Algorithm 6 returns matrices $\mathbf{D}^{(N)}$ and $\mathbf{P}^{(N)}$ with $d_{ij}^{(N)}$ containing length of the shortest path between vertices i and j, while matrix $\mathbf{P}^{(N)}$ contains information that makes possible to trace these shortest paths. Computational complexity of the algorithm is equal to $N+1$ iterations of $\mathrm{O}(N^2)$ each or $\mathrm{O}(N^3)$ in total.

Bibliographie

[1] Zippora Arzi-Gonczarowski. Perceive this as that – analogies, artificial perception, and category theory. *Annals of Mathematics and Artificial Intelligence*, 26(1-4) :215–252, 1999.

[2] Mikhail J. Atallah, editor. *Algorithms and Theory of Computation Handbook*. CRC Press, Boca Raton, FL, 1998.

[3] Marie-Pierre Béal and Olivier Carton. Determinization of transducers over finite and infinite words. *Journal of Theoretical Computer Science*, 289(1) :225–251, October 2002.

[4] Gregory Butler. Fundamental algorithms for permutation groups. *Lecture Notes in Computer Science*, 559, 1991.

[5] Gary Chartrand. *Introductory Graph Theory*. Dover Publications, New York, NY, 1984.

[6] Dragos Cvetkovic, Peter Rowlinson, and Slobodan Simic. *Eigenspaces of Graphs*. Cambridge University Press, Cambridge, UK, 1997.

[7] Reinhard Diestel. *Graph Theory*. Springer-Verlag, Heidelberg, electronic edition, 2005.

[8] Arthur Gill. *Introduction to the Theory of Finite-State Machines*. McGraw-Hill Book Company, 1962.

[9] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[10] Frederick C. Hennie. *Finite-State Models for Logical Machines*. John Wiley & Sons, Inc., 1968.

[11] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[12] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2nd edition, 2001.

[13] Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3) :331–378, 1994.

[14] S. Kleene. *Representation of Events in Nerve Nets and Finite Automata*, pages 3–42. Princeton University Press, Princeton, N.J., 1956.

[15] Richard C. Larson and Amedeo R. Odoni. *Urban Operations Research*. Massachusetts Institute of Technology, Cambridge, MA, 1997-99.

[16] Emmanuel Roche. Compact factorization of finite-state transducers and finite-state automata. *Nord. J. Comput.*, 4(2) :187–216, 1997.

[17] Joseph J. Rotman. *Advanced Modern Algebra*. Prentice Hall, Englewood Cliffs, NJ, 2003.

[18] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, Wellesley, MA, 4th edition, 2009.

[19] Eric W. Weinstein. *CRC Concise Encyclopedia of Mathehematics*. CRC Press, Boca Raton, FL, 1999.