On invertible inflection theory

+merlan #flirora

April 22, 2024

Contents

Version	2024-04-22 4fcbbad
Date	Mon Apr 22 22:46:50Z 2024
Racket	8.6
Pollen	3.2
System	Linux 5.4.109+ x86_64
NodeJS	v12.22.9
f9i	f9i 0.1.0

Table 1: Version information.

Chapter 1

Introduction

Suppose that $\mathscr{K}$ is the set of all possible lexical entries in a language, C is a set of category labels, and L is the set of all phonotactically valid words in the language. Then suppose $f : \mathscr{K} \times C \to L$ is the function that inflects words in the language.

We wish to not only be able to inflect a given lexical entry in a given category but also to perform the reverse task: given an inflected word form, find all lexical entries that it could be derived from, along with the category of the inflected form. That is, we wish to devise an algorithm that computes the preimage $f^{\leftarrow}(\{s\})$ for any $s \in L$.

If we are concerned only with finding matches in a fixed dictionary $K \subseteq \mathscr{K}$, then we can store a mapping for all $k \in K$ and $c \in C$ from $f(k, c)$ to (k, c). This method is conceptually simple and requires no structural knowledge of f, but it takes $O(|K| \cdot |C|)$ entries and requires C to be finite, making it space-intensive for highly inflecting languages.

Because Ŋarâþ Crîþ v9's inflection rules have been complex, this brute-force method has been used for f9i. During the development of sp9 for Project Shiva, however, +merlan #flirora proposed investigating alternative approaches that would not require storing all inflected forms.

1.1 The compositionality of linguistic inflection

Many inflection paradigms are not perfectly fusional. That is, we can express C as a Cartesian product of other sets $C_0 \times C_1 \times \cdots \times C_{r-1}$ and define a sequence of functions $f_i : L_i \times C_i \to L_{i+1}$, where $L_0 = \mathscr{K}$ and $L_r = L$. Then f involves applying each f_i in succession, passing each category label:

$$f(k, (c_0, c_1, \ldots, c_{r-1})) = f_{r-1}(\ldots f_1(f_0(k, c_0), c_1), \ldots, c_{r-1}) \tag{1.1}$$

This structure allows us to compute $f^{\leftarrow}(\{s\})$ by inverting each step of the inflection process in reverse order:

$$S_n = \{(s, \varnothing)\} \tag{1.2}$$

$$S_{i-1} = \bigcup_{(s,c) \in S_i} \{(s', (c', c)) \mid (s', c') \in f_{i-1}^{\leftarrow}(\{s\})\} \tag{1.3}$$

$$f^{\leftarrow}(\{s\}) = S_0 \tag{1.4}$$

Additionally, many parts of speech can be categorized into multiple inflectional classes. In mathematical terms, $\mathscr{K}$ can be partitioned into sets $P^0, P^1, \ldots, P^{\pi-1}$, each of which has a function $f^{j\cdot} : P^j \times C \to L = f|_{P^j \times C}$ that is 'simpler' to implement than f itself. Hence, $f^{\leftarrow}(\{s\})$ can be computed by attempting to match against each of the subsets P^j:

$$f^{\leftarrow}(\{s\}) = \bigcup_{j=0}^{\pi-1} (f^{j\cdot})^{\leftarrow}(\{s\}) \tag{1.5}$$

Alternatively, we can partition the set C into sets $C^0, C^1, \ldots C^{\gamma-1}$, with corresponding functions $f^{\cdot k} : \mathscr{K} \times C^k \to L = f|_{\mathscr{K} \times C^k}$, such that

$$f^{\leftarrow}(\{s\}) = \bigcup_{k=0}^{\gamma-1} (f^{\cdot k})^{\leftarrow}(\{s\}) \tag{1.6}$$

Importantly, each of these decompositions produce a set of tuples $(\mathscr{K}', C', L', f')$ which can be analyzed as a subproblem of the original problem and treated in the same way.

As an example, consider a language that has five classes for nominal inflections, which can be categorized into two broad groups, X and Y. This language also has cases in three groups:

- Group A cases are the most commonly used cases and have distinct declensions per class.
- Group B cases are less commonly used than group A cases and are declined differently between X classes and Y classes, but the declensions are the same within each group of classes.
- Group C cases are the least commonly used and are declined in the same way across all nouns. Additionally, while group A and B cases are coexponential with number, group C cases are monoexponential in that nouns in group C cases have a case ending followed by a number ending.

Then we could decompose the problem $(\mathscr{K}, C, L, f)$ first by partitioning C into $C^A \cup C^B \cup C^C$. Subsequently, $(\mathscr{K}, C^A, L, f^{\cdot A})$ is decomposed by partitioning $\mathscr{K}$ into $P^0 \cup \ldots \cup P^4$, while $(\mathscr{K}, C^B, L, f^{\cdot B})$ is decomposed by partitioning the same set as $P^X \cup P^Y$.

In contrast, $(\mathscr{K}, C^C, L, f^{\cdot C})$ is decomposed by composition into $(\mathscr{K}, C_0^C, L_1, f_0^{\cdot C})$ (adding the case affix) and $(L_1, C_1^C, L, f_1^{\cdot C})$ (adding the number affix). We now have the following simpler problems:

$$(P^0, C^A, L, f^{0.A}) \tag{1.7}$$
$$(P^1, C^A, L, f^{1.A}) \tag{1.8}$$
$$(P^2, C^A, L, f^{2.A}) \tag{1.9}$$
$$(P^3, C^A, L, f^{3.A}) \tag{1.10}$$
$$(P^4, C^A, L, f^{4.A}) \tag{1.11}$$
$$(P^X, C^B, L, f^{X.B}) \tag{1.12}$$
$$(P^Y, C^B, L, f^{Y.B}) \tag{1.13}$$
$$(\mathscr{K}, C_0^C, L_1, f_0^{\cdot C}) \tag{1.14}$$
$$(L_1, C_1^C, L, f_1^{\cdot C}) \tag{1.15}$$

This compositionality was the reason that we did not choose to define the set of possible categories $C(k)$ as dependent on the lexical entry. If we did so, then we could partition $\mathscr{K}$ into equivalence classes on $C(k)$ to yield subproblems that have a fixed set of category labels.

Chapter 2

Regular systems

In this chapter, we introduce a notion of REGULAR SYSTEMS, which are a collection of related regular languages. Specifically, a regular system consists of a finite state machine, yielding a language for each pair of start and end states. If the set of phonotactically valid words in a natural language is a regular language, then substrings of words can be considered to belong in one of the languages yielded by a corresponding regular system.

An example is Ŋarâþ Crîþ's finite state machine (Figure 2.1), which defines five states $\{s, g, o, n, \omega\}$. Each transition carries a payload defining a component of a syllable. For instance, the transition from s to g is labeled with I, the set of all initials, indicating that any initial is sufficient to transition from s to g. A phonotactically valid Ŋarâþ Crîþ word starts from s to ω, but individual morphemes can be between a different pair of states. For instance, the morphemes in *cenv-as-le you write it* have the following structures:

- *cenv-* from s to o. In the case that this has at least one full syllable and does not end with a lenited consonant, as is here, this is called a *stem*.
- *-as* from o to s
- *-le* from s to ω

Formally, we define a LANGUAGE SYSTEM Λ over a set of states Q and an alphabet Σ as a function $Q \times Q \to \mathscr{P}(\Sigma)$ that has the following properties:

- **Identity:** For all $q \in Q$, $\varepsilon \in \Lambda(q, q)$.
- **Transitivity:** For all $p, q, r \in Q$, $\Lambda(p, q) \cdot \Lambda(q, r) \subseteq \Lambda(p, r)$, where $\cdot$ denotes the concatenation of languages. We say that a language system has STRICT TRANSITIVITY if $\Lambda(p, r) = \bigcup_{q \in Q} \Lambda(p, q) \cdot \Lambda(q, r)$.

A regular system is a language system in which all languages $\Lambda(p, q)$ are regular.
Given a subset of states $Q' \subseteq Q$, we can define a SUBSYSTEM $\Lambda' = \Lambda|_{Q' \times Q'}$ over Q' and Σ.

2.1 Semiautomata

A SEMIAUTOMATON[1] (DSA) is a deterministic finite automaton that does not specify a start or accept state. A DSA $M = (Q, \Sigma, \delta)$ yields a regular language $M(p, q)$ for every $p, q \in Q$, as

[1] `https://en.wikipedia.org/wiki/Semiautomaton`

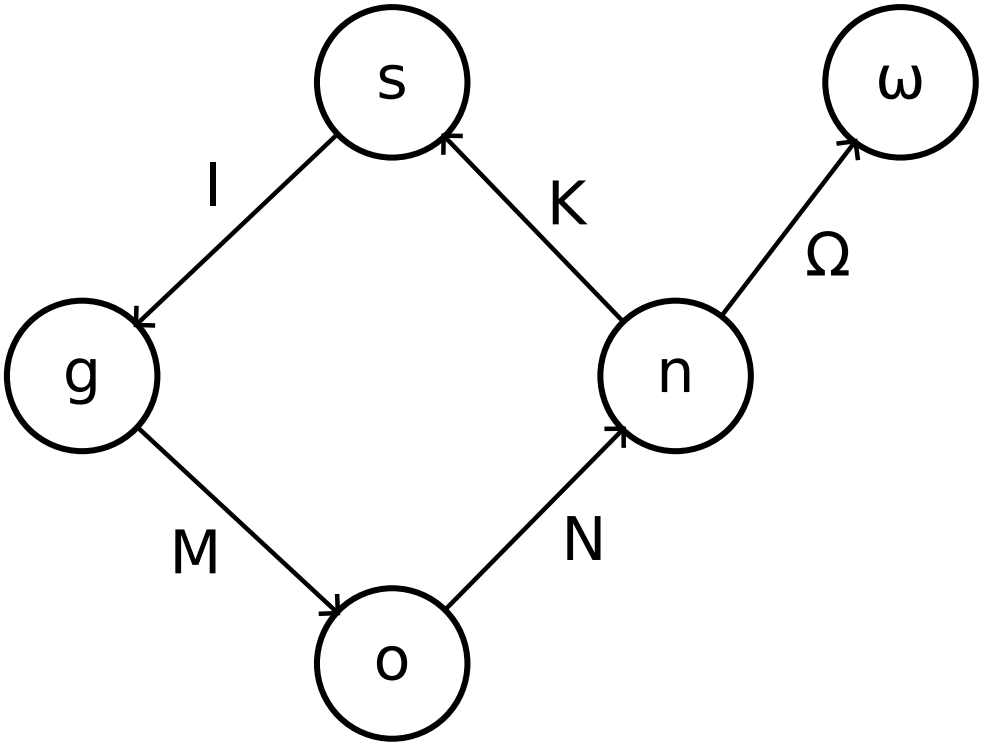

Figure 2.1: The finite ŝtate machine describing the phonotac̄tics of Ŋarâþ Crîþ.

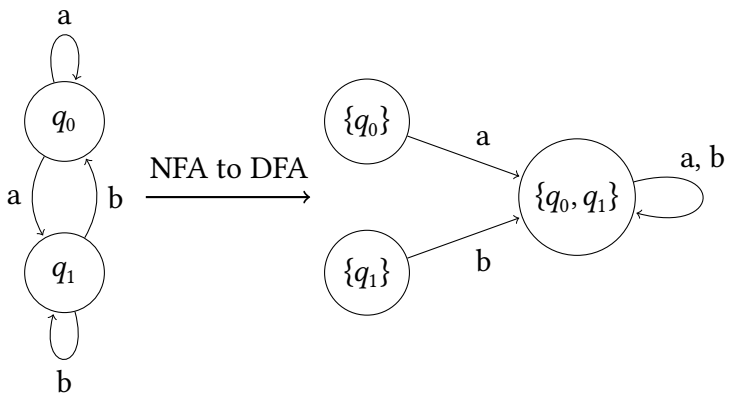

Figure 2.2: A counterexample to the conjecture that every NSA can be translated into a DSA with an equivalent regular system: the q_0 and q_1 states both correspond to multiple accepting states in the DSA.

Start	End	Language
q_0	q_0	(ab*)*
q_0	q_1	a+(ba*)*
q_1	q_0	b+(ab*)*
q_1	q_1	(ba*)*

Table 2.1: The regular expression for each language of the system described by Figure 2.2.

supplementing starting and accepting states to a semiautomaton yields a DFA. Additionally, these languages have the identity and transitivity properties required for regular systems. These properties (except for perhaps strict transitivity) still hold when we only look at a subset of states $Q' \subseteq Q$.

We also introduce the notion of a NONDETERMINISTIC SEMIAUTOMATON (NSA), which can be seen as a nondeterministic finite automaton without starting or accepting states. More formally, an NSA is a tuple $M = (Q, \Sigma, \delta)$ where $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function. For states $p, q \in Q$, $M(p, q)$ is the set of strings that can be generated by a path from p to q. We also define NONDETERMINISTIC SEMIAUTOMATA WITH ε-MOVES (NSA-ε) similarly, in which transitions without an input symbol are allowed. For similar reasons, NSAs and NSA-εs have the properties of regular systems, and these properties (except for perhaps strict transitivity) still hold when we only look at a subset of states $Q' \subseteq Q$. It follows that any regular system can be converted into a subsystem of an NSA-ε by applying Thompson's construction algorithm between each pair of input states.

However, while it is possible to translate an NFA into a DFA, translating an NSA into a DSA is less trivial. In general, it is not possible to translate an NSA into a DSA with the same regular system if each state of the NSA must correspond to exactly one state in the DSA. A single accepting state in the NSA might correspond to multiple accepting states in the DSA as shown in Figure 2.2.

However, it is possible to map each state of the NSA to a set of states in the DSA. In Figure 2.2, q_0 in the NSA M would correspond to $\{\{q_0\}, \{q_0, q_1\}\}$ in the DSA M' and q_1 would correspond to $\{\{q_1\}, \{q_0, q_1\}\}$. Let D be the function mapping states in M to corresponding sets of states in M'.

However, note that given two states p and q of M, the existence of a path from *some* $p^* \in D(p)$ to $q^* \in D(q)$ in M' accepting a string s does not imply that $s \in M(p, q)$. For example, while

$M(q_0, q_1)$ does not accept the empty string, $M'(\{q_0, q_1\}, \{q_0, q_1\})$ does. Therefore, we require that for all $p^* \in D(p)$, there exists a state $q^* \in D(q)$ such that M' accepts s from the state p^* to q^*.

More generally, we can establish a bijection between the set of abstract states $\mathcal{Q}$ of a regular system Λ and the set of sets of concrete states Q of a semiautomaton M. Then a string s is in $\Lambda(p, q)$ if and only if for all $p^* \in p$, there exists a $q^* \in q$ such that there is a path from p^* to q^* in M for s:

$$\Lambda(p, q) = \bigcap_{p^* \in p} \bigcup_{q^* \in q} M(p^*, q^*) \tag{2.1}$$

In this text, we concern ourselves only with DSAs and their subsystems. This is sufficient for most natural languages if we take the various components of a syllable, including null segments, as symbols themselves and augment each symbol with its source and destination states. For instance, Ŋarâþ Crîþ *tfełor* would be represented as

$$\begin{aligned}
&((\text{tf}, s, g),\\
&(\varepsilon, g, o),\\
&(\text{e}, o, n),\\
&(\varepsilon, n, s),\\
&(\text{ł}, s, g),\\
&(\varepsilon, g, o),\\
&(\text{o}, o, n),\\
&(\text{r}, n, \omega))
\end{aligned} \tag{2.2}$$

We refer to strings represented this way as FULLY SYLLABIFIED STRINGS or ASSEMBLAGES and any element of an assemblage as an ASSEMBLAGE UNIT. A DISAMBIGUATED REGULAR SYSTEM is a regular system Λ over Q in which for each nonempty string s, there is at most one $(p, q) \in Q \times Q$ such that $s \in \Lambda(p, q)$.

Chapter 3

Truncation

In many cases, inflection involves the addition of prefixes or suffixes to a stem. It follows that inverting this process would involve the removal of these affixes. However, concatenation often results in phonological changes to one or both of the stem and the affix. For example, *I meet you* in Ŋarâþ Crîþ would be derived as *cenv-a-ve*, but the co-occurrence of *v* in the resulting word is considered undesirable (*oginiþe cfarðerþ*) and is resolved by replacing the first *v* with an *n*. Thus, the resulting word is not ***cenvave* but rather *cennave*.

Let Σ be an alphabet and $f : \Sigma^* \times \Sigma^* \to \Sigma^* \times \Sigma^*$ be a function describing concatenation rules transforming pairs of strings (a, b). Generally, f can be seen as the composition of some number of individual 'rules' $f_0, f_1, \ldots, f_{n-1} : \Sigma^* \times \Sigma^* \to \Sigma^* \times \Sigma^*$.

Then let $a \overset{f}{\sim} b$ be the concatenation of a and b under f, or f-CONCATENATION, defined as

$$a \overset{f}{\sim} b = a' \cdot b' \text{ where } (a', b') = f(a, b) \tag{3.1}$$

Given $b, w \in \Sigma^*$, we wish to find all $a \in \Sigma^*$ such that $a \overset{f}{\sim} b = w$. If we can compute the preimage of a set over f, then this is simply

$$A = \bigcup_{i=0}^{|w|} \{\alpha \mid (\alpha', \beta') \in f^{\leftarrow}(\{(w[0..i], w[i..|w|])\}) \wedge \beta' = b\} \tag{3.2}$$

That is, we do the following:

- Split w into (α, β) in all possible ways.
- Find all (α', β') such that $f(\alpha', \beta') = (\alpha, \beta)$.
- Collect all such α' where $\beta' = b$.

An analogous problem and its solution can be stated for removing prefixes.

This algorithm is quite general, and certain properties of f admit simpler algorithms. (The solution is trivial if f is the identity function.)

If for all a, b and $(a', b') = f(a, b)$, the strings b and b' have the same length, then f is considered to be RIGHT-ISOMETRIC. In this case, we do not have to try all combinations of (α, β) but rather only the one in which $|\beta| = |b|$.

More generally, let

$$D_f(b) = \{|b'| \mid a \in \Sigma^*, (a', b') = f(a, b)\} \tag{3.3}$$

be the set of possible lengths that f could transform b to become. Then we only have to try combinations of (α, β) where $|\beta| \in D_f(b)$.

A stronger property that right isometry is RIGHT INVARIANCE, which requires $b = b'$; in other words, f is not allowed to change the second string. If f is right-invariant, then we can return an empty set if b is not a suffix of w.

Additionally, concatenation rules in human languages rarely change segments far away from the junction. For $a, b \in \Sigma^*$ and $(a', b') = f(a, b)$, let s be the longest common suffix of b and b', leaving r and r' before it. Then the FORWARD DEXTRAL RADIUS OF INFLUENCE of (a, b) with respect to f is $R_d(f; a, b) = |r|$ and the BACKWARD DEXTRAL RADIUS OF INFLUENCE is $R'_d(f; a, b) = |r'|$. We can also define these radii for the function itself:

$$R_d(f) = \sup_{a,b \in \Sigma^*} R_d(f; a, b) \tag{3.4}$$

$$R'_d(f) = \sup_{a,b \in \Sigma^*} R'_d(f; a, b) \tag{3.5}$$

Note that for right-isometric f, $R_d(f) = R'_d(f)$.

Given knowledge of these values, suppose that for a given (α, β), we compare b and β, aligning them at their ends. Then if b and β differ in the characters at indices $|b| - i$ and $|\beta| - i$, respectively, and if $|b| - i < R_d(f)$ or $|\beta| - i < R'_d(f)$, then we can conclude that $f^{\leftarrow}(\alpha, \beta)$ does not contain any pairs (α', β') such that $\beta' = b$ and can thus discard the pair (α, β).

Unfortunately, Ŋarâþ Crîþ v9e's concatenation rules have a dextral radius of influence of 5, while many of its affixes are shorter than 5 assemblage units long. For that reason, radius-of-influence simplifications are unlikely to be useful for Ŋarâþ Crîþ.

3.1 Examples of concatenation rules

The identity function is right-invariant and has a dextral radius of influence of 0.

Let m and n be nonnegative integers and $g \in \Sigma^m \times \Sigma^n \to \Sigma^* \times \Sigma^*$. Then denote by $\text{peephole}_{m,n}(g)$ the function f such that

$$f(a, b) = \begin{cases} (a[..|a| - m] \cdot a', b' \cdot b[n..]) & \text{if } |a| \geq m, |b| \geq n, (a', b') = g(a[|a| - m..], b[..n]) \\ (a, b) & \text{if } |a| < m \text{ or } |b| < n \end{cases} \tag{3.6}$$

In effect, $\text{peephole}_{m,n}(g)$ is a function that affects only the $m + n$ characters around the juncture.

Then the following statements are true:

$$D_f(b) = \{l + (|b| - n) \mid l \in D_g(b[..n])\} \text{ if } |b| \geq n \tag{3.7}$$

$$R_d(f) = n \tag{3.8}$$

$$R'_d(f) = \max_{a,b} |b'| \text{ where } (a', b') = g(a, b) \tag{3.9}$$

In particular, f is right-isometric if g is.

Often, we wish to replace one substring with another if a juncture occurs anywhere within the substring. For instance, the rule replacing 'cat' with 'dog' can be expressed as the composition of two functions $\text{peephole}_{2,1}(g_2) \circ \text{peephole}_{1,2}(g_1)$ where

$$g_1(a,b) = \begin{cases} (\text{d}, \text{og}) & \text{if } a = \text{c}, b = \text{at} \\ (a,b) & \text{otherwise} \end{cases} \tag{3.10}$$

$$g_2(a,b) = \begin{cases} (\text{do}, \text{g}) & \text{if } a = \text{ca}, b = \text{t} \\ (a,b) & \text{otherwise} \end{cases} \tag{3.11}$$

Suppose that we have a function $h \in \Sigma^k \to \Sigma^k$. If $f = \text{subst}_k(h)$ is such a substitution function, then it can be expressed as a composition of peephole functions $f_{n-1} \circ \cdots \circ f_1$ where

$$f_i = \text{peephole}_{i,k-i}(g_i) \tag{3.12}$$

$$g_i(a,b) = (s'[..i], s'[i..]) \text{ where } s' = h(a \cdot b) \tag{3.13}$$

$\text{subst}_k(h)$ is right-isometric and has a dextral radius of influence of $k-1$.

We have assumed that the function h preserves the length of the substring. We can generalize subst to account for functions that change the length of the input, but we must be careful about where the new juncture is placed.

3.2 Generalizations

In practice, we often want to check a word w against multiple suffixes from a fixed set $B \subseteq \Sigma^*$. This problem can be solved similarly to the single-suffix case by matching any of $(\alpha', \beta') \in f^{\leftarrow}(\ldots)$ where $\beta \in B$. This problem might be simplifiable depending on the properties of f. If f is right-invariant, for instance, then it is possible to use a trie containing the reversed elements of B.

Often, we do not want to find all elements of A but rather its intersection with a 'dictionary set' K_b.

3.2.1 Concatenation on regular systems

The notion of truncation can be generalized to be over any pair of formal languages. In this case, the concatenation rules function may have a different codomain from its domain. Usually, we are interested in languages of a regular system Λ in which the end state of the first string is equal to the start state of the second string. We define an EXTENDED CONCATENATIVE SYSTEM Φ over Λ as a pair of functions (κ, ϕ) where:

- $\kappa : Q \times Q \times Q \to Q$ is a function that takes in three states (p,q,r) and outputs a new middle state q', and
- $\phi : \bigcup_{p,q,r \in Q} [\{(p,q,r)\} \to \Lambda(p,q) \times \Lambda(q,r) \to \Lambda(p, \kappa(p,q,r)) \times \Lambda(\kappa(p,q,r), r)]$ returns for every triple of states (p,q,r) a concatenation rule function for two strings, possibly changing the middle state according to κ.

For $a \in \Lambda(p,q)$ and $b \in \Lambda(q,r)$, the extended concatenation $a \underset{p,q,r}{\overset{\Phi}{\sim}} b = a \overset{\phi(p,q,r)}{\sim} b$ is an element of $\Lambda(p,r)$. If no ambiguity would arise, we omit the state names from the operator and simply write $a \overset{\Phi}{\sim} b$.

3.2.2 *N*-ary concatenation

We have looked at binary concatenation; concatenation of more than two operands is often assumed to be left associative. In other words, concatenating (a, b, c) concatenates a and b first, then the result of that to c.

Another possibility for concatenating multiple morphemes is to apply the juncture rules after all of the concatenations. This means that in this example, any juncture rules applied between a and b would have access to the contents of c. This is more complex than repeated binary concatenation but has the advantage of being able to use word-global information (such as stress or syllable position within a word).

The concept of *N*-ary concatenation itself does not specify the order in which juncture rules are applied to each juncture. For instance, if the juncture rules consist of three subprocesses A, B, and C, then given a word with three junctures labeled 1, 2, and 3 from start to end, then the rules could trigger in any of the following orders (among other possibilities):

- Apply the subprocesses in sequence to each juncture from start to end: A1, B1, C1, A2, B2, C2, A3, B3, C3
- Apply the subprocesses in sequence to each juncture *from end to start*: A3, B3, C3, A2, B2, C2, A1, B1, C1
- Apply each subprocess to all junctures from start to end: A1, A2, A3, B1, B2, B3, C1, C2, C3
- Apply subprocesses A and C from start to end but B from end to start: A1, A2, A3, B3, B2, B1, C1, C2, C3
- Apply subprocess A to all junctures, then B and C to each juncture in sequence: A1, A2, A3, B1, C1, B2, C2, B3, C3

In exchange for this flexibility, *N*-ary concatenation has the disadvantage that truncation requires searching a larger space for possible juncture placements.

3.3 Case study: Ŋarâþ Crîþ v9e

According to the Ŋarâþ Crîþ v9e grammar, concatenation consists of the following processes applied across the juncture:

1. Any new instances of ⟦j⟧ before ⟦i⟧, ⟦î⟧, or ⟦u⟧ are elided.
2. Deduplication rules are applied.
3. Newly formed bridges are canonicalized and repaired.

We assume that we are working with assemblages in a regular system. Therefore, while a substitution function can be regarded as a composition of multiple functions $f_{n-1} \circ \cdots \circ f_1$, most of these functions will have no effect on concatenation at a given juncture state.

The first process, *glide elision*, is a right-invariant substitution function based on

$$h(((\mu, g, o), (\nu, o, n))) = \begin{cases} ((\varepsilon, g, o), (\nu, o, n)) & \text{if } \nu \in \{\text{i}, \text{î}, \text{u}\} \\ ((\mu, g, o), (\nu, o, n)) & \text{otherwise} \end{cases} \tag{3.14}$$

whose preimage is straightforward to compute.

The *deduplication* rules, which resolve instances of *oginiþe cfarðerþ*, work as follows:

1. The onset ⟦f⟧ or ⟦tf⟧ followed by a non-hatted vowel then ⟦f⟧ or ⟦p·⟧ is replaced with ⟦t⟧.
2. The onset ⟦þ⟧ or ⟦cþ⟧ followed by a non-hatted vowel then ⟦þ⟧ or ⟦t·⟧ is replaced with ⟦t⟧. In addition, a preceding ⟦þ⟧ or ⟦cþ⟧ coda is replaced with ⟦s⟧, and a preceding ⟦rþ⟧ coda is replaced with ⟦r⟧.
3. ⟦h⟧ followed by a non-hatted vowel then ⟦h⟧ or ⟦c·⟧ is replaced with ⟦p⟧.
4. ⟦v⟧ followed by a non-hatted vowel then ⟦v⟧ or ⟦m·⟧ is replaced with ⟦n⟧.
5. ⟦ð⟧ followed by a non-hatted vowel then ⟦ð⟧ or ⟦d·⟧ is replaced with ⟦ŋ⟧.
6. ⟦ħ⟧ followed by a non-hatted vowel then ⟦ħ⟧ or ⟦g·⟧ is replaced with ⟦g⟧.

In f9i, these rules are implemented twice: once for the case when the vowel in question is followed by a nonterminal coda (thus capturing the following initial), and once for the case when it is followed by a terminal coda (in which case only rules #1 and #2 are applicable). In both cases, the preceding coda is captured if available.

This process is more involved than glide elision, but its preimage is not too difficult to compute, and the process is right-isometric.

The assemblage form makes these changes difficult: the first consonant following a vowel might belong to the coda of the same syllable or to the initial of the following syllable, and it may be part of a complex coda or initial. It also complicates situations in which changing a letter requires the word to be syllabified differently; for this problem, v9e simply chooses to apply bridge resolution after deduplication, although this has the disadvantage of failing to remove some cases of *oginiþe cfarðerþ*.

Ŋarâþ Crîþ v9e's deduplication rules are quite crude, prompting ad-hoc workarounds to be made in specific instances of inflection. There are plans in Project Shiva to expand the range of *oginiþe cfarðerþ* and thus the scope of deduplication, as well as a desire to avoid changing the initial consonant of a word. An even more challenging problem is propagation: if deduplication changes a consonant such that a new instance of *oginiþe cfarðerþ* arises, then additional invocations of deduplication might be required to resolve it.

The final step of concatenation is *bridge resolution*, which modifies awkward coda–initial pairs to more convenient ones. This process is also used for canonicalizing these pairs according to the maximal onset principle.

Ŋarâþ Crîþ v9e's version of this process is complicated by the fact that although v7 allowed *ŋ* as a coda, v9 does not. When *ŋ* appeared as a coda, it was changed into *r*, modifying the preceding vowel. *-aŋ*, *-oŋ*, and *-uŋ* were changed to *-or*, and *-eŋ* and *-iŋ* were changed to *-jor*. Reflecting this change, bridge resolution first outputs either a true coda or a pseudo-coda of *ŋ*, subsequently resolving the latter case by applying this change to the vowel. That is, a final of *-or* might have arisen from one of *-or*, *-aŋ*, *-oŋ*, or *-uŋ*. Likewise, a final of *-jor* might have arisen from one of *-jor*, *-jaŋ*, *-jeŋ*, *-joŋ*, *-eŋ*, or *-iŋ*.

Since the number of possible bridges is relatively small, tabulation can be used to implement the preimage of the first step.

Because bridge resolution is right-isometric, Ŋarâþ Crîþ v9e's concatenation rules as a whole are as well.

Chapter 4

Into paradigms

We define a PARADIGM Π over (Σ, L) as a triple (C, S, ψ) where:

- C is a set of categories that Π inflects for,
- $S = (S_0, S_1, \ldots, S_{p-1})$ is a sequence of subsets of Σ^*, such that S_i defines the set of values that the ith variable part can take, and
- $\psi : S_0 \times S_1 \times \cdots \times S_{p-1} \times C \to L$ determines the form of an inflected word.

In most cases, we constrain ψ such that $\psi(s_0, \ldots, s_{p-1}, c)$ is the result of concatenating elements of $\{s_0, \ldots s_{p-1}\}$ in a manner that depends only on the value of c. (The particular semantics of concatenation depends on the language being modeled, but we require truncation to be available.)